Graph Embedding for Multi-Robot Path Planning in Complex Environments

Xifeng Gao[1], Zherong Pan[1], and Ruiqi Ni[2]

***Abstract*—We propose a graph-embedding approach to approximate continuous multi-robot path planning (MPP) problems as discrete ones, allowing known discrete planning techniques be exploited in realistic, complex continuous environments. We first design a special pebble graph with a set of conditions, under which MPP problems have the feasibility guarantee. We then develop a mesh optimization algorithm to embed our pebble graph into arbitrarily complex environments. We show that our feasibility conditions can be converted into differentiable geometric constraints, such that our mesh optimizer can find feasible solutions via constrained numerical optimization. Two algorithms can be used to solve MPP problems on our pebble graphs. Conflict-Based Searches (CBS) are preferred for finding (near) optimal solutions. We further introduce a space-time parallel scheduling approach to find sub-optimal solutions for large swarms of congested robots. We have evaluated the effectiveness of our approach on a set of environments with complex geometries, where our method achieves an average of 99.0% free-space coverage and 30.3% robot density, ensuring a large solution space of discrete MPP algorithms executed on the graphs.**

***Index Terms*—Computational Geometry, Path Planning for Multiple Mobile Robots or Agents, Planning, Scheduling and Coordination**

I. Introduction

MPP aims at routing multiple robots from their distinct start to goal positions in a complex environment, which is a unified theoretical model of various applications, including warehouse arrangement, coverage planning for floor vacuuming [1, 2], and inspection planning for search and rescue [3]. These real-world applications often require MPP planners to answer queries in continuous space. However, determining the feasibility of continuous MPP instances have been shown to be PSPACE-hard [4] (for rectangular objects) and strongly NP-hard [5] (for polygonal environments), even for simplified MPP instances, let along environments with complex shapes. It is thus unsurprising that brute-force search algorithms [6, 7, 8] are only practical for tens of robots.

Prior works [9, 10, 11] show that MPP becomes much more tractable in discrete space by assuming that robots can only move on a discrete graph, i.e. robots reside in a discrete set of vertices and move along a discrete set of paths connecting the vertices. MPP in discrete space is also referred as multi-agent path finding (MAPF), for which polynomial-time algorithms exist for checking the feasibility and finding feasible solutions [9, 12, 11]. For special graphs, the sub-optimality of solutions can even be bounded [13]. As a result, a common practice is to convert continuous spaces into discrete problems.

While many works have studied discrete MPP, comparatively fewer work has focused on the quality of discretization and its affect on the quality of discrete MPP solutions. As an example, start and goal positions in continuous space need to be associated with discrete start/goal nodes after discretization. If the positions and nodes are close enough, robots can simply move to the start node without conflict. However, a poor discretization can prevent this. In Figure 1 for example, a lattice-style triangulation is not boundary aligned and can miss start/goal positions in the corners of the star-shaped world. Another example is to capture environments with narrow passages, where regular titling can easily miss the narrow regions.

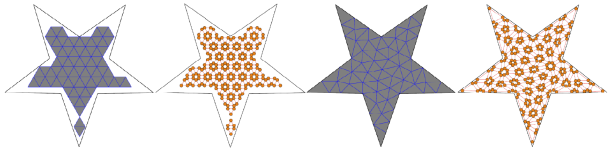

Fig. 1: For an environment with non-axis aligned boundaries, compared to a triangulation in the lattice style (left) and its resulting graph (middle left), our method generates a boundary aligned triangulation (middle right) that can produce a graph with the full coverage of the entire environment (right), capturing the maximum solution space of discrete MPP solutions. Discrete MPPs on the graph are always feasible when robots (orange) are arranged on the graph vertices. The $\mathcal{M}_{\#rc}/\mathcal{D}$/Coverage for regular lattice and ours are 74/0.24/60.5% and 91/0.28/100%, respectively.

This paper focuses on techniques to translate complex, continuous spaces into graph representations in a way that maximizes free space coverage (to ensure problem fidelity), ensures collision-free motion when moving along the graph, and guarantees feasibility of discrete MPP problems posed on the graph. Specifically, we make the following contributions:

- We design a graph with special topology and geometry conditions, on which any MAPF instance is guaranteed to be feasible;
- We prove the designed graph can be topologically identified with a planar triangle mesh, where the feasibility conditions of the discrete MPP instances can be mapped to differentiable geometric constraints;
- We propose a greedy triangle mesh optimization framework to embed the graph on environments with complex boundaries, so that both the robot capacity and the free space coverage of the graph can be optimized under the feasibility conditions;
- For the obtained graph, we propose a space-time search algorithm to quickly answer discrete MPP queries for a large number of congested robots, making use of our guaranteed feasibility.

Manuscript received: February, 24, 2022; Revised April, 15, 2022; Accepted April, 15, 2022.

This paper was recommended for publication by Editor Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments.

[1]First Author and Second Author are with Lightspeed & Quantum Studios, Tencent America {gxf.xisha, zherong.pan.usa}@gmail.com

[2]Second Author is with Department of Computer Science, Florida State University rn19g@my.fsu.edu

Digital Object Identifier (DOI): see top of this page.

We have evaluated our graph-embedding method on a set of complex environments including 2D real floor plans of houses, malls, and gaming maps. Our generated graph embedding not only aligns well with the curved input boundary but also has high free-space coverage ratio of 99.0% and robot density ratio of 30.3% on average. For the generated graphs, we also demonstrate the benefits of the proposed space-time search algorithm in handling highly congested robot swarms.

II. Related Work

We review representative related works.

Graph Embedding applies the discrete graph theory to continuous environments. It has been used in [14, 15, 13] that consider rectangular, triangular, and hexagonal grids. These works do not allow the graph topology or the geometry to be modified. We show that the special feasibility conditions in discrete MPP problems can be preserved during mesh optimization by limiting the set of legal remeshing operators in [16] and using differentiable geometric constraints. **Mesh Optimization** algorithm was originally proposed in [16], which exhaustively applies a set of remeshing operators to monotonically improve a given quality metric. Although this method only achieves local optimality, it is capable of exploring a large portion of the solution space.

Method	Boundary-Aligned	Highly-Congested	Optimality
CBS [8]	N/A	No	Optimal
Divide-and-Conquer [13]	No	Yes	Bounded
Our Method (Section VII)	Yes	Yes	No

TABLE I: Features of different discrete MPP planners in terms of generating boundary-aligned graphs (CBS can handle any graph, so we put N/A), handling highly congested robot swarms, and finding optimal solutions.

MPP problems in a general setting is intractable [4, 5] and practical methods target at problem subclasses. Early works [6, 17] use RRT or PRM to solve continuous MPP problems. However, as the number of robots goes large, e.g. more than 30 in [17], RRT-like methods would incur exponentially increasing running time. When robots are restricted to move on a graph, their motions can be enumerated using CBS [8] or integer programming [18]. These algorithms are practical for finding optimal solutions of a small group of robots. CBS has been extended in several ways to handle more robots [19, 20], some of which sacrifices optimality. However, CBS and its variants become intractable when the environment is highly congested. In the later case, pebble-graph-based feasibility check [9, 12, 11] is the only way to find sub-optimal solutions, to the best of our knowledge. Optimal solutions are intractable to find but sub-optimality can be bounded in some methods [13]. The features of various planning algorithms that can be used on our embedded graphs are summarized in Table I.

III. Problem Statement

Continuous MPP problems for large number of robots are notoriously difficult to solve. Instead, we leverage computer graphics and graph embedding techniques to approximate continuous MPP problems as discrete ones with high fidelity. For a workspace $\mathcal{W} \subset \mathbb{R}^2$ with piecewise linear boundaries, we assume all the robots involved in the discrete MPP problems are disk-shaped with an identical radius r so the ith robot centered at $\mathbf{x}_i \in \mathcal{W}$ takes up the space of a circular region: $C(\mathbf{x}_i) \triangleq \{\mathbf{x} | \|\mathbf{x} - \mathbf{x}_i\| \leq r\} \subset \mathcal{W}$. In this work, we use a graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ that can be embedded into $\mathcal{W}$ to construct the discrete subset MPP problems. $\mathcal{G}$ restricts the robot motions, i.e., robots can only reside on $\mathcal{X}$ and move along $\mathcal{E}$. Here $\mathcal{X}$ is the set of vertices $\mathcal{X} \triangleq \{\mathbf{x}_1, \cdots, \mathbf{x}_{|\mathcal{X}|}\}$. $\mathcal{E}$ is the set of edges connecting vertices in $\mathcal{X}$ and we refer to a direct path connecting two positions in $\mathcal{W}$ and a graph edge interchangeably. Our goal is to compute a graph embedding $\mathcal{G}$ of $\mathcal{W}$ so that: 1) discrete MPP problems on $\mathcal{G}$ is guaranteed to have feasible collision-free solutions, where the ith robot moves from $\mathbf{x}_i$ to $\mathbf{x}_{\sigma(i)}$ with σ being a position permutation; 2) the number of vertices of $\mathcal{G}$ is maximized to enable large swarms of robots in simultaneous navigation; 3) $\mathcal{W}$ is covered as much as possible to provide "good" approximations for continuous MPP problems.

IV. Overview

Our method is based on the observation that a special graph $\mathcal{G}$ not only allows guaranteed feasible collision-free solutions, but also is constructible from a triangulation of $\mathcal{W}$ by enforcing collision-free conditions on the triangle mesh.

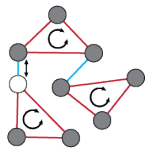

Fig. 2: 3-loops of our designed graph.

As illustrated in Figure 2, the graph we consider consists of simple loops (red edges) with 3 vertices. These 3-loops are connected by extra edges into a single, connected components. Two kinds of moves are available for the robot: A **cyclic move** allows a group of 3 robots to cyclically permute locations along a 3-loop (black, arced arrows). A **vacant move** allows a robot to move along any blue edge $\mathbf{e} \in \mathcal{E}$ connecting $\mathbf{x}_i$ and $\mathbf{x}_j$, as long as $\mathbf{x}_j$ is not occupied by any other robot (a white vacant vertex). We will show that discrete MPP problems are feasible on such graphs, as long as the number of robots is less than $|\mathcal{X}|$.

In the following sections, we first present a list of conditions to embed $\mathcal{G}$ into a triangulation of $\mathcal{W}$, while allowing robots to perform cyclic and vacant moves in a collision-free manner (Section V). We then introduce a mesh optimization algorithm to maximize a metric function $\mathcal{M}(\mathcal{G})$ that encodes different requirements for a "good" $\mathcal{G}$ (Section VI). Finally, we propose a space-time path planning algorithm that schedules simultaneous robot motions on a discrete graph (Section VII).

V. Graph Embedding and Mesh Discretization

In this section, we consider the geometric properties of $\mathcal{G}$ that allows robots to move in a collision-free manner within a $\mathcal{W}$. Our key innovation is a mapping from a mesh discretizing $\mathcal{W}$ to a graph embedded in $\mathcal{W}$. As illustrated in Figure 3 (a), a mesh discretizing $\mathcal{W}$, denoted as $\bar{\mathcal{G}} = \langle \bar{\mathcal{X}}, \bar{\mathcal{E}} \rangle$, is another graph that is also a simplicial-complex, cell-decomposition of $\mathcal{W}$ and we denote all the variables defined on the mesh using a bar

over variables. Given a mesh $\bar{\mathcal{G}}$, we can convert it into a graph using Algorithm 1, which is denoted as a mapping $\Phi(\bar{\mathcal{G}}) = \mathcal{G}$. Algorithm 1 consists of 4 general steps. First, we place robots on inner corner points of each cell, i.e. points inside a cell that are distance-r away from two consecutive edges of the cell. Next, we assume that 3 robots inside the same cell form a loop, along which cyclic moves can be performed, and we add 3 loop edges to $\mathcal{G}$. We then add inter-cell edges to $\mathcal{G}$ between the two pairs of corner points sharing an edge $\bar{\mathbf{e}}$ in $\bar{\mathcal{G}}$, along which vacant move can be performed. Note that some cells may be too small and robots placed on inner corners are not collision-free. Even when corner points are collision-free, robots inside the same cell can still collide when performing cyclic moves. Therefore, we add a fourth and final step to remove from $\mathcal{G}$ all the corner points (and incident edges) in invalid cells. We define a cell as invalid if the 3 corner points or cyclic moves are not collision-free. In the following, we show that the collision-free conditions of $\mathcal{G}$ can be converted to three differentiable geometric constraints on $\bar{\mathcal{G}}$.

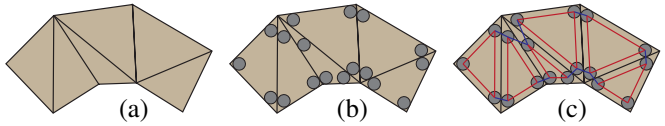

Fig. 3: (a): A mesh discretizing $\mathcal{W}$ is another graph $\bar{\mathcal{G}}$ that is also a cell decomposition with straight-line edges and triangular cells. (b): We map $\bar{\mathcal{G}}$ to $\mathcal{G}$ by first putting robots (gray) on the cell's corner points. (c): We then create loop edges for robots in a single cell (red), and finally create inter-cell edges between robots of neighboring cells (blue).

Algorithm 1: Evaluate $\Phi(\bar{\mathcal{G}})$.

$\mathcal{X} \leftarrow \varnothing$ and $\mathcal{E} \leftarrow \varnothing$
for Each triangular cell in $\bar{\mathcal{G}}$ **do**
  for Corner point $\mathbf{x}_{1,2,3}$ in the cell **do**
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{x}_i\}$
  for $\mathbf{e}$ between two points in $\mathbf{x}_{1,2,3}$ **do**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathbf{e}\}$
for $\mathbf{e}$ between neighboring corner points sharing edge in $\bar{\mathcal{G}}$ **do**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathbf{e}\}$
$\mathcal{G} \leftarrow < \mathcal{X}, \mathcal{E} >$
for Corner points $\mathbf{x}$ of invalid cell **do**
  Remove $\mathbf{x}$ from $\mathcal{G}$
Return $\mathcal{G}$

A. Condition 1: Planner Embedding

Since $\bar{\mathcal{G}}$ is a simplicial complex, each cell is a triangle and we denote the 3 vertices of this triangle as: $\bar{\mathbf{x}}_{1,2,3}$. The distance-r corner points can be computed using the following formula, as illustrated in Figure 4 (a):

$$\mathbf{x}_2 = \bar{\mathbf{x}}_2 + r\frac{(\bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_2)\|\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2\| + (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)\|\bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_2\|}{\|(\bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_2) \times (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)\|}. \quad (1)$$

We only show formula for $\mathbf{x}_2$ and the formulas for $\mathbf{x}_{1,3}$ are symmetric. We will mark the cell as invalid and remove the 3 vertices from $\mathcal{G}$ if there are overlappings between $C(\mathbf{x}_{1,2,3})$.

B. Condition 2: Collision-Free Cyclic Moves

Even when the 3 corner points are not overlapping, robots can still collide when they perform cyclic moves along the 3 loop edges with constant speed, as illustrated in Figure 4 (b). To derive a condition for collision-free cyclic moves, we assume that the 3 robots trace out a trajectory $\tau_{1,2,3}(t)$ where $t \in [0,1]$. At time t, the 3 robots are at positions:

$$\mathbf{x}_{1,2,3}(t) \triangleq \mathbf{x}_{1,2,3}(1-t) + \mathbf{x}_{2,3,1}t,$$

and we have collision-free cyclic moves if:

$$\|\mathbf{x}_{1,2,3}(t) - \mathbf{x}_{2,3,1}(t)\| \geq 2r \quad \forall t \in [0,1]. \quad (2)$$

Here we use cyclic subscripts to denote 3 symmetric conditions. The left-hand side of Equation 2 is quadratic when squared and determining the smallest value of a quadratic equation in $[0,1]$ has closed-form solution, which can be used for determining whether a cell is invalid in Algorithm 1.

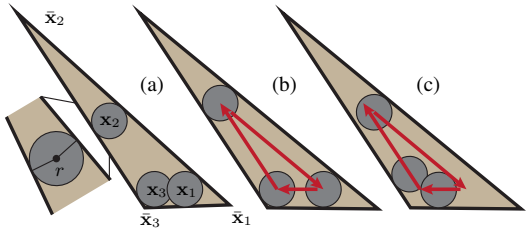

Fig. 4: (a): For a thin triangle, corner points will not be collision-free, violating condition 1. (b): Even when condition 1 holds, robots need to follow the red arrows and perform a cyclic move. (c): If collisions happen during a cyclic move, condition 2 is violated.

In addition to determining the validity of a cell, our mesh optimization algorithm (Section VI) requires an operator that can modify an invalid cell's geometric shape to achieve validity using numerical optimization. To this end, we derive a condition equivalent to Equation 2 but does not contain continuous time variable t, because the variable t can take infinitely many values from $[0,1]$ leading to a difficult semi-infinite programming problem [21]. Taking one of the equations $\|\mathbf{x}_1(t) - \mathbf{x}_2(t)\| \geq 2r$ in Equation 2 for example (the other 2 cases are symmetric), its left-hand-side is a polynomial of a single time variable t. To eliminate t, the Fekete, Markov-Lukaćz theorem [22] can be applied to show that, if Equation 2 holds, then we have:

$$\|\mathbf{x}_1(t) - \mathbf{x}_2(t)\|^2 - 4r^2 = \alpha_1(2t-1)^2 + 2\alpha_2\alpha + \alpha_3 + \alpha_4(4t - 4t^2) \wedge \begin{pmatrix} \alpha_1 & \alpha_2 \\ \alpha_2 & \alpha_3 \end{pmatrix} \geq 0 \wedge \alpha_4 \geq 0,$$

where $\alpha_{1,2,3,4}$ are four unknown variables to be fitted. The 2×2 PSD-cone constraint is equivalent to two quadratic constraints: $\alpha_1\alpha_3 \geq 0$ and $\alpha_1\alpha_3 \geq \alpha_2^2$. By equating coefficients in the quadratic constraints, we can express $\alpha_{1,2,3,4}$ in terms of $\mathbf{x}_{1,2,3}$ to get the following equivalent form:

$$(\frac{1}{4}\|2\mathbf{x}_2 - \mathbf{x}_1 - \mathbf{x}_3\|^2 + \alpha_4)(\frac{1}{4}\|\mathbf{x}_1 - \mathbf{x}_3\|^2 - 4r^2 - \alpha_4) \geq \left[\frac{1}{4}(\mathbf{x}_1 - \mathbf{x}_3)(2\mathbf{x}_2 - \mathbf{x}_1 - \mathbf{x}_3)\right]^2 \wedge \alpha_4 \geq 0, \quad (3)$$

where α_4 is an additional decision variable that cannot be eliminated. But unlike t, we only need to satisfy Equation 3 for a single α_4. We summarize this result below:

Lemma 5.1: A cell in $\bar{\mathcal{G}}$ is valid if its 3 corner points, computed according to Equation 1, satisfy Equation 2 for all

$t \in [0,1]$ or if the 3 corner points satisfy Equation 3 for any positive α_4.

C. Condition 3: Collision-Free Vacant Moves

We show that, as long as condition 2 is satisfied, condition 3 must also be satisfied, so $\Phi(\bar{\mathcal{G}})$ always satisfy condition 3. Condition 3 requires that a robot can move to a vacant position along any $\mathbf{e} \in \mathcal{E}$. We analyze two cases: $\mathbf{e}$ being a loop edge or $\mathbf{e}$ being an inter-cell edge.

Loop edge: If one of the 3 corner points $\mathbf{x}_i$ in a cell is vacant and another robot $\mathbf{x}_j$ is moving along a loop edge to $\mathbf{x}_i$, we show that this move is always collision-free. We prove by contradiction. If this move is not collision-free, then the third corner point $\mathbf{x}_k$ must be in the way between $\mathbf{x}_i$ and $\mathbf{x}_j$. However, in this case, cyclic moves in the cell is not collision-free, which contradicts condition 2.

Inter-cell edge: If two neighboring triangles satisfy condition 2 and share an edge $\bar{\mathbf{e}} \in \bar{\mathcal{E}}$, then cyclic moves inside each triangle are collision-free. As illustrated in Figure 5, if robot $\mathbf{x}_i$ is to be moved to a vacant position $\mathbf{x}_j$ along an inter-cell edge, then we can rotate $\mathbf{x}_i$ (along with the two other robots) clockwise and $\mathbf{x}_j$ counterclockwise until the line-segment $\mathbf{x}_i - \mathbf{x}_j$ is orthogonal to $\bar{\mathbf{e}}$ and the convex hull of $C(\mathbf{x}_i)$ and $C(\mathbf{x}_j)$ does not contain other robots so the vacant move is collision-free. Finally, we rotate $\mathbf{x}_{i,j}$ reversely to undo extra changes.

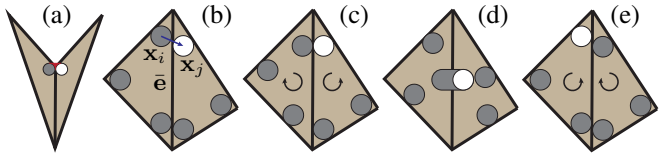

Fig. 5: As illustrated in (a), moving $\mathbf{x}_i$ to vacant $\mathbf{x}_j$ along the straight line can interfere with other triangles (red area) in obtuse angles. If $\mathbf{x}_i$ is to be moved to a vacant position $\mathbf{x}_j$ along a blue inter-cell edge (b), then we can rotate $\mathbf{x}_i$ clockwise and $\mathbf{x}_j$ counterclockwise to align $\mathbf{x}_i$ and $\mathbf{x}_j$ (c), so that the convex hull of $C(\mathbf{x}_i)$ and $C(\mathbf{x}_j)$ does not contain other robots (d). After the vacant move inside the convex hull, we rotate $\mathbf{x}_{i,j}$ reversely to undo extra changes (e).

VI. Mesh Optimization

In this section, we present a variant of mesh optimization algorithm [16, 23] to search for $\bar{\mathcal{G}}$ that maximizes a user given metric $\mathcal{M}(\bar{\mathcal{G}}, \mathcal{G})$. Afterwards, a corresponding $\mathcal{G}$ can be extracted using Algorithm 1 to solve discrete MPP problems.

A. Metric Function

Our metric consists of 4 terms $\mathcal{M}_{\#r}, \mathcal{M}_{\#rc}, \mathcal{M}_{sd}, \mathcal{M}_g$ that can either be a function of the mesh $\bar{\mathcal{G}}$ or the graph $\mathcal{G}$. These four terms measure the robot packing density as well as the area covered by the robots in $\mathcal{W}$. Our first term, $\mathcal{M}_{\#r}$ is defined as: $\mathcal{M}_{\#r}(\mathcal{G}) \triangleq |\mathcal{X}|$, which is the number of vertices contained in $\mathcal{G}$. This term reflects the number of robots that $\mathcal{G}$ can hold, but discrete MPP problems might not be feasible for all these robots because $\mathcal{G}$ might not be connected. $\mathcal{M}_{\#r}$ is a heuristic that guides our algorithm to find more valid cells at an early stage and then try to connect them. Our second term, $\mathcal{M}_{\#rc}$, is the number of robots in the largest connected component of $\mathcal{G}$:

$$\mathcal{M}_{\#rc}(\mathcal{G}) \triangleq \max_{\mathcal{G}' \subseteq \mathcal{G} \text{ connected}} (\mathcal{M}_{\#r}(\mathcal{G}')),$$

which is the maximal number of robots for which discrete MPP problems are feasible. Note that $\mathcal{M}_{\#r,\#rc}$ are discrete, non-differentiable terms that are related to both the topology and geometry of $\bar{\mathcal{G}}$. To increase $\mathcal{M}_{\#r,\#rc}$, we need to update the shape of triangles in $\bar{\mathcal{G}}$ and also update their connectivity so that they are valid. Our third term is the robot packing density averaged over the valid cells:

$$\mathcal{M}_{sd}(\mathcal{G}) \triangleq \frac{\pi r^2 \mathcal{M}_{\#r}(\mathcal{G})}{\sum_{\text{valid cell in } \bar{\mathcal{G}}} |\text{cell}|}.$$

$\mathcal{M}_{sd}$ is a heuristic guidance term that biases our algorithm towards denser robot packing, when only a subset of the workspace is covered by the robot. Our last metric term $\mathcal{M}_g$ is the 2D AMIPS energy [24] using the smallest regular triangle satisfying condition 2 as the target shape. $\mathcal{M}_g$ is differentiable, purely geometric, and related to $\bar{\mathcal{G}}$ only.

Algorithm 2: Optimize $\mathcal{M}$ with respect to $\bar{\mathcal{G}}$.

Input: Initial $\bar{\mathcal{G}}$
Compute $\mathcal{M} \leftarrow \mathcal{M}(\bar{\mathcal{G}}, \Phi(\bar{\mathcal{G}}))$
Set terminate←False
for pass← $1, 2$ **do**
  while Not terminate **do**
    Set terminate←True
    Perform Collapsing, Splitting (if pass = 1), Flipping, Smoothing, LocalOpt, and GlobalOpt sequentially
    if any operations above improves $\mathcal{M}_{\#r} + w\mathcal{M}_{\#rc}$ **then**
      Set terminate←False
Return $\Phi(\bar{\mathcal{G}})$

B. Greedy Maximization of Metric

The greedy algorithm is summarized in Algorithm 2, which exhaustively tries one of the re-meshing operators: edge-flip, edge-split, edge-collapse, vertex-smoothing, local-optimization, and global-optimization. Our re-meshing operators, except for global-optimization, are all local and only affect a small neighborhood of cells. The first 4 re-meshing operators are inherited from [16, 23], as illustrated in Figure 6. Edge-flip is guided by $\mathcal{M}_g$ and is mostly applied to obtuse triangles and turns them into acute triangles. Edge-split and edge-collapse are used to remove tiny edges and split long edges in $\bar{\mathcal{G}}$, respectively. Finally, vertex-smoothing locally optimizes $\mathcal{M}_g$ with respect to the one-ring neighborhood of a single vertex. The four operators combined can turn a low-quality mesh into one with nice element shapes.

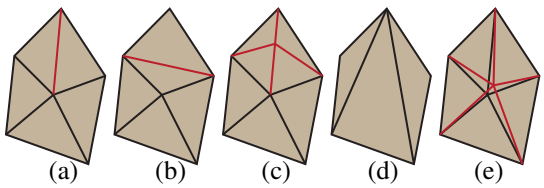

Fig. 6: The first 4 remeshing operators inherited from [16]. (a): original mesh, (b): edge-flip, (c): edge-split, (d): edge-collapse, (e): vertex-smoothing (part of mesh modified by the operator in red).

Our algorithm uses two passes. In the first pass, we allow the algorithm to explore a larger search space by allowing both edge-collapse and edge-split. To ensure the convergence of the first pass, we avoid the cases where consecutive edge-flip and edge-split operators happen for a same edge repeatedly. Allowing edge-split will create more triangles and potentially lead to denser packing of robots. In practice, we split a $\bar{\mathbf{e}}$ when its length is larger than $1.3\times$ of the optimal length $\bar{\mathbf{e}}^*$. Here the optimal length is the edge length of the smallest regular triangles satisfying condition 2, which equals to: $|\bar{\mathbf{e}}^*| = (2\sqrt{3}+4)r$. In the second pass, we are more conservative and disallow edge-split. This pass can be considered as post-processing, merging too small triangles and simplifying the robot layout. Within each pass, we check every possible operator and only accept operators when the weighted metric $\mathcal{M}_{\#r} + w\mathcal{M}_{\#rc}$ monotonically increases and we set $w = 10$. We also reject operators that violate our assumption on $\bar{\mathcal{G}}$ being a cell decomposition, i.e. introducing non-manifold connection and flipped cells. The non-manifoldness of $\bar{\mathcal{G}}$ can happen only in edge-collapse and can be avoided by the link condition check [25]. A flipped cell has a negative area which can be checked by the following constraint:

$$(\bar{\mathbf{x}}_2 - \bar{\mathbf{x}}_1) \times (\bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_1) \geq 0. \tag{4}$$

C. Local- and Global-Optimization Operators

Aside from the four remeshing operators inherited from [16, 23], we introduce one local- and one global-optimization operators. These operators use the primal-dual interior point method to optimize the geometric shape of the cells, taking condition 2 as hard constraints. At the same time, we maximize $\mathcal{M}_{sd}$ by minimizing the area of valid cells, leading to a denser robot packing. Specifically, for a cell in $\bar{\mathcal{G}}$ with vertices $\bar{\mathbf{x}}_{1,2,3}$, we solve the following problem in the local-optimization operator:

$$\begin{aligned} &\underset{\bar{\mathbf{x}}_{1,2,3}}{\text{argmin}} \sum_{\text{valid cell in } \bar{\mathcal{G}}} |\text{cell}| \\ &\mathbf{s.t.} \quad Eq.3 \ \forall \text{valid cell in } \bar{\mathcal{G}} \quad Eq.4 \ \forall \text{cell in } \bar{\mathcal{G}}, \end{aligned} \tag{5}$$

where we require that the entire mesh to be a cell decomposition, so we add Equation 4 for all cells. We also require that those originally valid cells stay valid, so we add Equation 3 for all valid cells. This optimization can be solved efficiently because it is local. Since we only treat the 3 vertices of a single cell in $\bar{\mathcal{G}}$ as decision variables, most of the terms in the objective function and constraints are outside the 1-ring neighborhood of $\bar{\mathbf{x}}_{1,2,3}$ and not influenced by the decision variables, which can be omitted. Note that Equation 3 is expressed in terms of the corner points $\mathbf{x}_{1,2,3}$, instead of mesh vertices $\bar{\mathbf{x}}_{1,2,3}$. When the optimizer requires partial derivatives of Equation 3 against $\bar{\mathbf{x}}_{1,2,3}$, we use the chain rule on the relationship Equation 1.

The global-optimization operator is very similar to the local-optimization operator, which also takes the form of Equation 5, but we set all the variables $\bar{\mathbf{x}}$ as decision variables. The global-optimization is more expensive than all other operators, but we found that this last operator can significantly improve robot packing density and it is more efficient than applying the local-optimization operator to each of the cell.

VII. Discrete MPP on Loop Graphs

In this section, we present a method to find feasible solutions to discrete MPP instances for a large-scale highly congested robot swarm. Our method is capable of using vacant graph vertices to schedule parallel robot motions.

A. Discrete MPP Feasibility

We show that discrete MPP instances restricted to the graph with the topology as illustrated in Figure 9 are always feasible:

***Lemma** 7.1:* If $\mathcal{G}$ has more than 1 loop, then any discrete MPP problem with $N < |\mathcal{X}|$ is feasible.

We only give a sketch of proof. Any permutation σ of robot positions can be decomposed into a set of pairwise swaps. To swap the location of two robots, $\mathbf{x}_i$ and $\mathbf{x}_j$, we first find a path in $\mathcal{G}$ connecting $\mathbf{x}_i$ and $\mathbf{x}_j$, which is always possible as $\mathcal{G}$ is connected, denoted as $\mathbf{x}_i, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K, \mathbf{x}_j$. We can further decompose the swap into a series of sub-swaps:

$$\mathbf{x}_i \leftrightarrow \mathbf{x}_1, \cdots, \mathbf{x}_i \leftrightarrow \mathbf{x}_K, \mathbf{x}_i \leftrightarrow \mathbf{x}_j, \mathbf{x}_K \leftrightarrow \mathbf{x}_j, \cdots, \mathbf{x}_1 \leftrightarrow \mathbf{x}_j,$$

where each pair of positions in a sub-swap are connected directly by some $\mathbf{e} \in \mathcal{E}$, where $\mathbf{e}$ is either a loop edge or an inter-cell edge. In either case, the sub-swap can be performed with the help of a nearby vacant vertex as illustrated in Figure 9. Finally, such vacant vertex must exist because our number of robots is strictly less than the number of vertices and the vacant position can be moved anywhere via vacant moves.

B. Parallel Robot Moves

We measure the quality of a discrete MPP solution via makespan, which is the number of timesteps used by the entire scheduling process, where each timestep involves one or more robots moving simultaneously.In the worst case, the makespan of above mentioned discrete MPP solutions is $\mathcal{O}(|\mathcal{X}|^2)$. This is because we have only one vacant vertex, which is needed to perform every sub-swap. When more vacant vertices are available, we can parallelize the sub-swaps and improve the makespan. We present a space-time search algorithm to find parallel sub-swaps, which is similar to [13] in principle but adapted to graphs of more general topology.

As illustrated in Figure 8 right, we first use the CLINK algorithm [26] to cluster the loops into a binary tree. This algorithm starts by treating each loop as a separate cluster, and then iteratively merge two smallest clusters (with the smallest number of vertices) until only one cluster is left. We then iteratively merge small leaf nodes until all leaf nodes have more than $K > 1$ loops ($K = 4$ in the inset), where K is a user specified parameter determining the congestion-level. In the worst case, this method requires $\lceil |\mathcal{X}|/(3K) \rceil$ vertices to be left vacant. Next, we introduce a vacant vertex for each leaf node. As a result, cyclic moves and vacant moves can be performed parallelly within a leaf node and two pairs of connected leaf nodes can perform two robot position swaps in parallel. Finally, we adopt the divide-and-conquer algorithm proposed in [13]. For every internal binary tree node, we consider their two children, which are two connected sub-graph. We swap robots between the two sub-graphs until $\mathbf{x}_i$

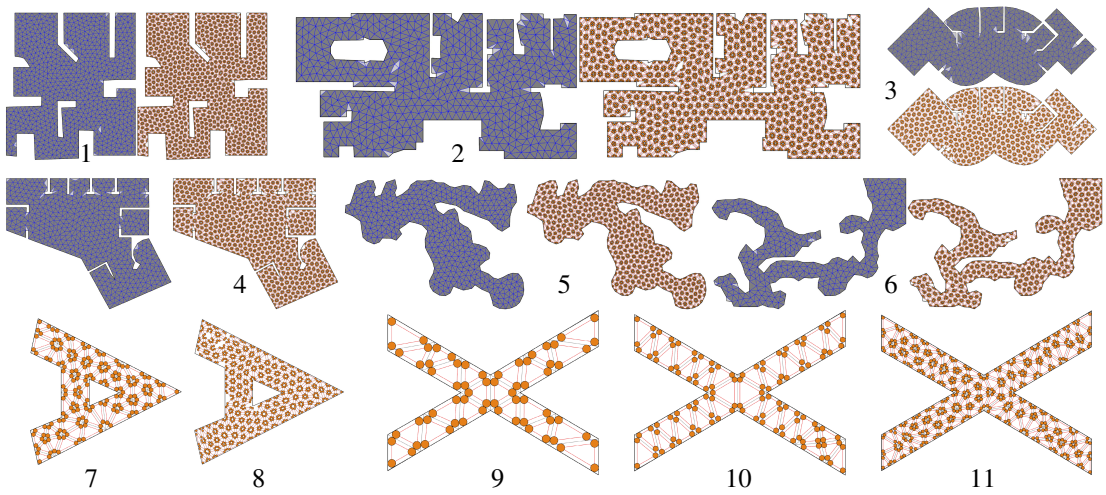

Fig. 7: Gallery of graph embeddings of robot number ranging from 66 to 4467. For each example (1-6), left is our optimized triangulation where cells colored in light gray are invalid, right is the converted graph. We also show two simple examples (7-11) with varying robot sizes.

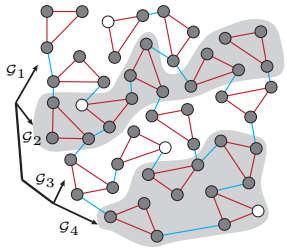

Fig. 8: A binary tree is generated by clustering the 3-loops.

and $\mathbf{x}_{\sigma(i)}$ belongs to the same sub-graph for all $i = 1, \cdots, N$. This procedure is performed recursively from the root node to all the leaves, where the swaps within different sub-trees are performed in parallel.

C. Scheduling Robot Swaps between Sub-Graphs

The major challenging in the above algorithm lies in the scheduling of parallel robot position swaps between sub-graphs, for which we use a space-time data structure. We denote $\mathcal{G}_i$ as the ith leaf node and we assume there are N_1 leaf nodes in one of the sub-graph and N_2 in the other ($N_1 + N_2 \leq \lceil |\mathcal{X}|/(3K) \rceil$). The inter-sub-graph position swaps are decomposed into several rounds, where each round consists of several parallel position swaps between two connected leaf nodes. We denote as $\mathcal{G}_{i,t}$ as the state of $\mathcal{G}_i$ at the beginning of tth round. We further connect $\mathcal{G}_{i,t}$ and $\mathcal{G}_{j,t+1}$ using a space-time edge $\mathbf{e}_{ij,t}$ if two conditions hold: 1) $\mathcal{G}_i$ and $\mathcal{G}_j$ are connected by some inter-loop edge $\mathbf{e} \in \mathcal{E}$; 2) $\mathcal{G}_i$ and $\mathcal{G}_j$ belongs to the same sub-graph, i.e., $i, j \leq N_1$ or $N_1 < i, j \leq N_1 + N_2$. Our space-time data structure is defined as the space-time graph $\langle \{\mathcal{G}_{i,t} | t \leq t_0 + 1\}, \{\mathbf{e}_{ij,t} | t \leq t_0\} \rangle$, where t_0 is the maximal allowed number of rounds. Note that the space-time graph is directed with each $\mathbf{e}_{ij,t}$ directing from $\mathbf{e}_{j,t+1}$ to $\mathbf{e}_{i,t}$. The inter-sub-graph position swaps can be accomplished by multi-rounds of swaps between leaf nodes, which in turn corresponds to selecting a set of space-time edges $\mathbf{e}_{ij,t}$. Our algorithm greedily select a set of space-time edges leading to an inter-sub-graph position swap in the earliest round. After each selection, we remove the conflicting space-time edges before selecting the next set of edges, until all the robots belong to the correct sub-graph. When no position swaps can be found for the given t_0, we introduce a new round and increase t_0 by one. This procedure is guaranteed to terminate.

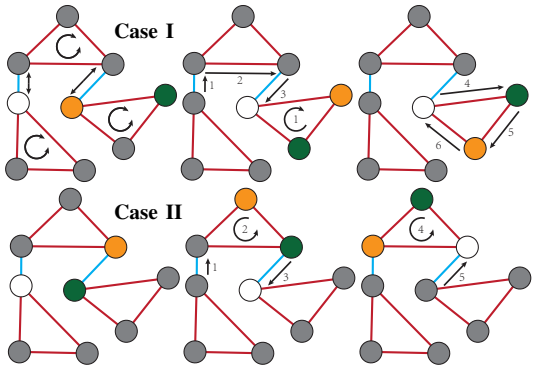

Fig. 9: A discrete MPP problem solved using a series of sub-swaps between two connected vertices. (white: vacant vertex; gray: vertices whose positions should remain the same; yellow/green: vertices whose position should be swapped; the ith arrow means a move at the ith round; arrows with same the number mean moves that can be performed at the ith round in parallel) **Case I:** If yellow/green vertices belong to the same loop, then we can move the vacant vertex to the loop (1,2) and swap the two vertices via 3 vacant moves (4,5,6). **Case II:** If two vertices belong to two connected loops, then we can use (1,2,3,4) to reduce to **Case I**. After the swaps, all other robots' position are made unaltered using a reversed sequence of moves.

Our method is outlined in Algorithm 3, which is composed of two main steps. First, we consider all pairs of connected leave nodes $\langle \mathcal{G}_{a,t}, \mathcal{G}_{b,t} \rangle$ belonging to different sub-graphs. We find the two shortest paths connecting from $\mathcal{G}_{a,t}$, $\mathcal{G}_{b,t}$ to some leaf node containing a robot that belongs to a different sub-graph. Specifically, we run Dijkstra's algorithm with all-one edge weights from both $\mathcal{G}_{a,t}$ and $\mathcal{G}_{b,t}$. We terminate the Dijkstra's algorithm at any $\mathcal{G}_{i,t}$ satisfying the following condition:

$$\min_{t'=t,\cdots,t_0} (\#\mathcal{G}_{i,t'}) > 0, \tag{6}$$

where we define $\#\mathcal{G}_{i,t}$ as the number of robots within $\mathcal{G}_{i,t}$ that belongs to a different sub-graph from $\mathcal{G}_{i,t}$ at the beginning of the tth round. Intuitively, Equation 6 ensures that swapping one more robot away from $\mathcal{G}_i$ during the tth round will not cause conflict with the number of to-be-swapped robots

required by other swapping operations in the future rounds. Finally, note that we consider nodes pairs in a time-ascending order. The first pair of leaf nodes, for which such paths can be found, are selected, which corresponds to the position swap in the earliest round.

Our second step would remove three types of conflicting edges: 1) If an edge $\mathbf{e}_{ij,t}$ is selected, then all the edges $\{\mathbf{e}_{kj,t}, \mathbf{e}_{ik,t} | k = 1, \cdots, N_1 + N_2\}$ are removed because they conflict with the edges along the two paths; 2) all the edges $\{\mathbf{e}_{ak,t}, \mathbf{e}_{bk,t} | k = 1, \cdots, N_1 + N_2\}$ are removed because they conflict with the inter-sub-graph swap between $\mathcal{G}_{a,t}$, $\mathcal{G}_{b,t}$; 3) Let's define $\#_{/}\mathcal{G}_{j,t}$ as the number of robots belonging to the same sub-graph as $\mathcal{G}_j$ at the beginning of tth round ($\#\mathcal{G}_{j,t} + \#_{/}\mathcal{G}_{j,t} + 1$ equals to the number of vertices in $\mathcal{G}_j$), then all the edges:

$$\{\mathbf{e}_{kj,t} | \min_{t'=t,\cdots,t_0} \#_{/}\mathcal{G}_{j,t'} = 0 \wedge k = 1, \cdots, N_1 + N_2\},$$

should be removed. This is because selecting edge $\mathbf{e}_{kj,t}$ implies that a robot $\mathbf{x}_a \in \mathcal{G}_{k,t}$ must be swapped with another robot $\mathbf{x}_b \in \mathcal{G}_{j,t}$ at tth round. We must have $\mathbf{x}_b$ belong to the same sub-graph as $\mathcal{G}_j$ at the beginning of tth round, because otherwise the shortest path would stop at $\mathbf{x}_b$ instead of moving on to $\mathbf{x}_a$. However, $\min_{t'=t,\cdots,t_0} \#_{/}\mathcal{G}_{j,t'} = 0$ implies that, if $\mathbf{x}_b$ was swapped out of $\mathcal{G}_j$ during tth round, there will not be enough to-be-swapped robots (that belongs to the same sub-graph as $\mathcal{G}_j$) during some future rounds. For illustrative purpose, the attached video includes an animation of step-wise robots motion scheduled by our approach for the 10th example in Figure 7.

VIII. Results and Analysis

We implement our algorithm and conduct experiments on a single desktop machine with Intel i7-9700 CPU. The input to our algorithm is a vector image in the svg format representing $\mathcal{W}$. The vector image is manually traced in Adobe Illustrator from online png images of real building floor plans and maps used in games. Our main Algorithm 2 starts from an initial guess of $\bar{\mathcal{G}}$, which can be generated using constrained Delaunay Trianglation. Our mesh optimization algorithm then turns a triangle mesh (Figure 10 left), with only a few triangles satisfying the feasibility conditions, into a regular triangle mesh (Figure 10 right), with conditions satisfied for most triangles. We evaluate our algorithm on a set of complex workspaces and we summarize the statistics of $\mathcal{M}_{\#r}$, robot density $\mathcal{D}$ (the ratio between the area of robots and the area of $\mathcal{W}$), free space coverage (the ratio between the area of all valid cells and the area of $\mathcal{W}$), and computational time (minutes) in Figure 7 and Table II. Note that, since $\mathcal{M}_{\#r}$, $\mathcal{M}_{\#rc}$ are the same for all the tested examples in Table II, we list $\mathcal{M}_{\#r}$ only. While our graph embedding cannot support real-time computation for large environments, for a given size robot and environment, we just need to compute it once and it can support repeated discrete MPP queries.

Algorithm 3: Space-Time Scheduling of Robot Swaps

Input: Sub-graphs A: leaves $G_{1,\cdots,N_1}$ and $G_{N_1+1,\cdots,N_1+N_2}$
$t_0 \leftarrow 2$
Build space-time graph $< \{\mathcal{G}_{i,t} | t \le t_0 + 1\}, \{\mathbf{e}_{ij,t} | t \le t_0\} >$
while $\sum_{i=1,\cdots,N_1+N_2} \#\mathcal{G}_{i,t_0} > 0$ **do**
  Found←False
  for $t = 1, \cdots, t_0$ **do**
    for Connected leaf nodes $\mathcal{G}_a, \mathcal{G}_b$ **do**
      Run Dijkstra's algorithm from $\mathcal{G}_{a,t}, \mathcal{G}_{b,t}$
      until some $\mathcal{G}_{i,t'}$ satisfying Equation 6 is found
      if Found both shortest paths **then**
        Select edges on the two shortest paths
        for Selected edge $\mathbf{e}_{ij,t}$ **do**
          Remove $\{\mathbf{e}_{kj,t}, \mathbf{e}_{ik,t}\}$
          for $t' = 1, \cdots, t_0$ **do**
            Remove
            $\left\{\mathbf{e}_{ki,t'}, \mathbf{e}_{kj,t'} \middle| \min_{t''=t',\cdots,t_0} \#_{/}\mathcal{G}_{j,t''} = 0\right\}$
        Remove $\{\mathbf{e}_{ak,t}, \mathbf{e}_{bk,t}\}$
        Found←True
  if Found←False **then**
    $t_0 \leftarrow t_0 + 1$
    Build space-time nodes $\mathcal{G}_{i,t_0+2}$, edges $\mathbf{e}_{ij,t_0+1}$

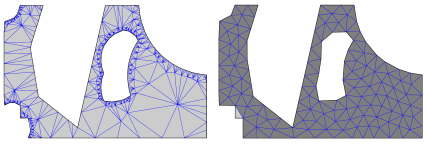

Fig. 10: Initialization (left) and the final result (right) of our graph embedding on the same workspace.

Measures \ Models	1	2	3	4	5	6	7	8	9	10	11
$\mathcal{M}_{\#r}$ ↑	1489	1061	782	1021	756	569	60	257	22	40	150
$\mathcal{D}$ ↑	0.31	0.29	0.30	0.30	0.29	0.28	0.28	0.30	0.24	0.19	0.26
Coverage ↑	1.00	0.98	0.98	0.98	1.00	0.99	1.00	1.00	1.00	1.00	1.00
time (min) ↓	236	43	74	98	47	54	0.55	4.35	0.10	0.20	0.77

TABLE II: Statistics of $\mathcal{M}_{\#r}$, robot density $\mathcal{D}$, free space coverage, and computational time (minutes) for the graph generation of the workspaces listed in Figure 7 (indexed from top to bottom, left to right). ↑ means the larger value the better, while ↓ represents the opposite.

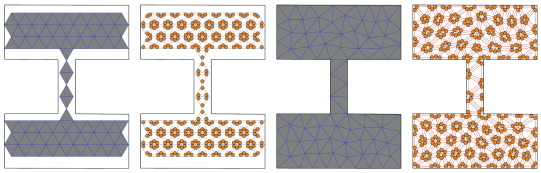

Fig. 11: Our method (right) allows robots to pass through narrow regions, while the prior regular lattice graph embedding methods (left) fail to capture such critical regions. The $\mathcal{M}_{\#rc}$/$\mathcal{D}$/Coverage for regular lattice and ours are 52/0.19/61.5% and 131/0.29/100%, respectively.

Comparison with Regular Embedding: We also compare our method with the regular-pattern baseline, illustrated in Figure 1 and Figure 11. Our approach exhibits three benefits: 1) boundary alignment, while the regular pattern method introduces undesirable zig-zag boundary coverage; 2) full coverage of the work space, while prior works leaves large spaces near the boundary; and 3) captures narrow passages that can be missed by regular tiling patterns, leading to infeasible discrete MPP problems.

Varying Robot Radius: We also profile the influence of different robot radius in Figure 12. Using our arrangement of robots in Section V, we can use regular triangles with the optimal area A^* to cover $\mathcal{W}$ and then remove triangles that are outside $\mathcal{W}$. We change the radius r, and compare the curve with the ideal curve $3|\mathcal{W}|/A^*$ by plotting the change of $\mathcal{M}_{\#r}$ against r. Our method closely matches the ideal reference curve for different r.

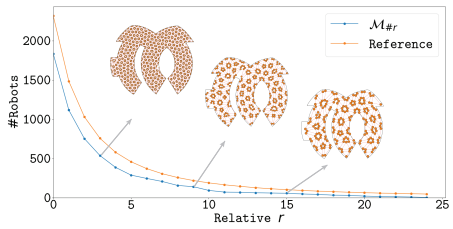

Fig. 12: By varying r, we plot $\mathcal{M}_{\#r}$ and produced by our boundary aligned approach and the ideal reference $3|\mathcal{W}|/A^*$.

$\#Robot$	10	11	12	13	14	15	16	17	18	19	20	25	30
time (s)	0.001	0.047	-	-	0.217	-	0.007	0.344	15.7	-	-	-	-

TABLE III: Computational time of the CBS algorithm [27] executed for discrete MPP instances with varying number of robots on the graph shown in Figure 13 left with 1128 nodes. "-" denotes no solution found.

Discrete MPP Algorithm Comparison: Our generated graph embedding $\mathcal{G}$ allows prior CBS algorithm [27] to solve discrete MPP instances. Although there are more efficient extensions to CBS [19, 20], we cannot find open-source implementations for general graph topology to conduct experiments. By testing on the graph of Figure 3, from Table III, we found that the computing time of such algorithms vary greatly w.r.t. the number of robots, e.g., CBS cannot find optimal solutions for a discrete MPP instance with 20 robots even after hours computing on the graph shown in Figure 3. Our proposed parallel discrete MPP scheduler Algorithm 3 process all discrete MPP instances with thousands of robots within 6mins (Figure 13), while the solution is sub-optimal. Figure 13 left illustrates such a sub-optimal path. Each MAPF problem is randomly generated and we run 5 times to obtain a sound statistics. Although our approach does not ensure an optimal path as visualized in Figure 13 left, it guarantees to deliver a valid solution for large number of robots. When more and more vacant vertices (corresponding to smaller K where more parallelizable clusters can be computed to speed-up the scheduling process), the rate of acceleration of our approach is more obvious (Figure 13).

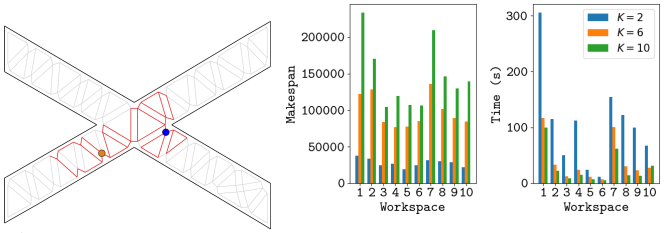

Fig. 13: Left: 1 (red) out of 66 paths scheduled by our approach, where orange and blue nodes are start and goal positions, respectively; Middle and right: the average makespan and the running time (s) for 10 environments under three different K, where for most environments, the makespan reduces linearly with K.

IX. Conclusion and Limitations

We present a method to solve graph-restricted discrete MPP problems in complex environments. Our method uses a special graph topology to ensure discrete MPP feasibility. The special graph topology can be further identified with triangular meshes, which in turn is optimized via mesh optimization and embedded into arbitrarily complex environments. Using a divide-and-conquer algorithm with space-time scheduling, we can parallelize the robot motions and greedily improve the makespan. In the future, we plan to conduct experiments with more efficient discrete MPP algorithms such as [19, 20].

References

[1] J. M. Palacios-Gasós, Z. Talebpour, E. Montijano, C. Sagüés, and A. Martinoli, "Optimal path planning and coverage control for multi-robot persistent coverage in environments with obstacles," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1321–1327.

[2] K. M. Hasan, Abdullah-Al-Nahid, and K. J. Reza, "Path planning algorithm development for autonomous vacuum cleaner robots," in *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, 2014, pp. 1–6.

[3] J. L. Baxter, E. K. Burke, J. M. Garibaldi, and M. Norman, "Multi-robot search and rescue: A potential field based approach," in *Autonomous Robots and Agents*, S. C. Mukhopadhyay and G. S. Gupta, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 9–16.

[4] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects: Pspace-hardness of the" warehouse-man's problem"," *The international journal of robotics research*, vol. 3, no. 4, pp. 76–88, 1984.

[5] P. Spirakis and C. K. Yap, "Strong np-hardness of moving many discs," *Information Processing Letters*, vol. 19, no. 1, pp. 55–59, 1984.

[6] G. Sánchez and J.-C. Latombe, "On delaying collision checking in prm planning: Application to multi-robot coordination," *The International Journal of Robotics Research*, vol. 21, no. 1, pp. 5–26, 2002.

[7] D. Le and E. Plaku, "Cooperative multi-robot sampling-based motion planning with dynamics," in *ICAPS*, 2017.

[8] 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.

[9] V. Auletta, A. Monti, M. Parente, and P. Persiano, "A linear-time algorithm for the feasibility of pebble motion on trees," *Algorithmica*, vol. 23, no. 3, pp. 223–245, 1999.

[10] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *International Joint Conferences in Artificial Intelligence (IJCAI-11)*, Barcelona, Spain, 2011, pp. 294–300.

[11] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms," in *Algorithmic foundations of robotics XI*, Springer, 2015, pp. 729–746.

[12] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *25th Annual Symposium onFoundations of Computer Science, 1984.*, 1984, pp. 241–250.

[13] J. Yu, "Average case constant factor time and distance optimal multi-robot path planning in well-connected environments," *Autonomous Robots*, vol. 44, no. 3, pp. 469–483, 2020.

[14] Shuai D. Han, Edgar J. Rodriguez, and Jingjin Yu, "Sear: A polynomial- time multi-robot path planning algorithm with expected constant-factor optimality guarantee.," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, IEEE, 2018.

[15] R. Chinta, S. D. Han, and J. Yu, "Coordinating the motion of labeled discs with optimality guarantees under extreme density," in *The 13th International Workshop on the Algorithmic Foundations of Robotics*, 2018.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993, pp. 19–26.

[17] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," in *Robotics: Sciences and Systems*, Jun. 2009, pp. 2–3.

[18] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[19] J. Li, W. R. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 12 353–12 362, 2021.

[20] G. Gange, D. D. Harabor, and P. J. Stuckey, "Lazy cbs: Implicit conflict-based search using lazy clause generation," in *ICAPS*, 2019, pp. 155–162.

[21] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[22] M. Laurent, "Sums of squares, moment matrices and optimization over polynomials," in *Emerging applications of algebraic geometry*, Springer, 2009, pp. 157–270.

[23] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo, "Tetrahedral meshing in the wild," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, Jul. 2018.

[24] X.-M. Fu, Y. Liu, and B. Guo, "Computing locally injective mappings by advanced mips," *ACM Trans. Graph.*, vol. 34, no. 4, Jul. 2015.

[25] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev, "Topology preserving edge contraction," *Publ. Inst. Math. (Beograd) (N.S*, vol. 66, pp. 23–45, 1998.

[26] D. Defays, "An efficient algorithm for a complete link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, 1977.

[27] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 39–45.