2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

Multi-Robot Path Planning Using Medial-Axis-Based Pebble-Graph Embedding

Liang He[1†], Zherong Pan[3], Kiril Solovey[2], Biao Jia[4], and Dinesh Manocha[4]

***Abstract*— We present a centralized algorithm for labeled, disk-shaped Multi-Robot Path Planning (MPP) in a continuous planar workspace with polygonal boundaries. Our method automatically transform the continuous problem into a discrete, graph-based variant termed the pebble motion problem, which can be solved efficiently. To construct the underlying pebble graph, we identify inscribed circles in the workspace via a medial axis transform and organize robots into layers within each inscribed circle. We show that our layered pebble-graph enables collision-free motions, allowing all graph-restricted MPP instances to be feasible. MPP instances with continuous start and goal positions can then be solved via local navigations that route robots from and to graph vertices. We tested our method on several environments with high robot-packing densities (up to 61.6% of the workspace). For environments with narrow passages, such density violates the well-separated assumptions made by state-of-the-art MPP planners, while our method achieves an average success rate of 83%.**

I. Introduction

We propose a centralized approach for Multi-Robot Path Planning (MPP) for planar workspaces with polygonal obstacles. This problem has a wide range of applications such as warehouse management [1], computer games [2], and crowd modeling [3], which require the coordination of a large swarm of robots within a limited computational budget. MPP plans must be computed within a couple of milliseconds for an interactive game and an automatic warehouse needs to answer thousands of queries on a daily basis. Unfortunately, solving general MPP problems is NP-hard [4, 5, 6] and practical algorithms based on sampled roadmaps [7] and conflict-based search [8] quickly become intractable for more than a few robots. Follow-up research relies on additional assumptions on environment shapes and/or robot arrangements to attain practical performance.

A. Related Work

We review the three assumptions most relevant to our work. To begin with, graph pebbling [9, 10, 11] lays the theoretical foundation of discrete structure in MPP problems. It assumes that robots are restricted to vertices and move along the edges of a graph. Prior work established fast algorithms to verify and construct feasible solutions for sub-classes of pebble-graphs [9, 11]. Certain regular grids are endowed with complete results [12, 13, 8, 14] and near-optimal solutions [15, 16] although exact optimality is intractable to attain [17]. However, building a pebble-graphs for workspaces with complex boundaries is still challenging, because regular grids cannot cover narrow passages and sharp features as illustrated in Figure 1 (b).

Some methods [18, 19, 20, 21, 22, 23] use random sampling to build discrete graphs that capture the structure of the continuous problem. Some of these methods have completeness guarantees, that is, if a graph leading to feasible solutions exists, they can ultimately build one. However, to the best of our knowledge, those approaches do not scale well with the number of robots.

Lastly, the well-separated assumption [24, 25, 26, 27] partially addresses the shortcomings of the above methods. By assuming robots (as well as their goal positions) are sufficiently separated from each other and the obstacles, each robot can find paths to their goal regardless of the order of movements of other robots. Unlike graph-pebbling, the well-separated assumption can be verified and established for an arbitrary workspace easily, after which feasibility can be guaranteed. On the downside, the large separation distance can limit the number of robots.

B. Main Results

We propose a new method to construct pebble-graphs for arbitrary workspaces without using sampling-based methods. As illustrated in Figure 1 (c), our method relies on the medial-axis transform [28] to identify a series of inscribed circles with center points connected into skeleton curves. We then organize robots into discrete layers inside each inscribed circles, as illustrated in Figure 1 (d). We show that our robot arrangement allows both translational and rotational pebble motions. Following an idea similar to [11], we prove that all the MPP instances restricted to our pebble-graph are feasible as long as the number of graph vertices is larger than the number of robots. Finally, our method can inherently extend to continuous start and goal positions by moving robots to and from closest graph vertices via local navigation algorithms [29].

Our method works well in workspaces with complex obstacles and boundary shapes. We have conducted systematic comparisons with [16] in 5 different environments, each having $30-160$ randomized MPP instances. For the benchmark with narrow passages, our method achieves a 100% success rate on an average robot density between $30-71\%$ of the workspace, while such a high density violates the well-separated assumption in [25] and the method in [16] can fail whenever agents fall inside these narrow passages.

II. Problem Statement and Approach Overview

We formalize 2D labeled MPP problems. We define $\mathcal{W} \subseteq \mathbb{R}^2$ as the 2D workspace and assum that the boundary of workspace, $\partial\mathcal{W}$, is piecewise linear. We have a set of N disk-shaped robots initially centered at $\mathbf{s}^1, \ldots, \mathbf{s}^N$ with identical

[1]Liang He is with Department of Computer Science, University of North Carolina at Chapel Hill. {lianghe.hust@gmail.com} [2]Kiril Solovey is with the Technion, Israel Institute of Technology. {kirilsol@technion.ac.il} [3]Zherong Pan is with Lightspeed & Quantum Studio, Tencent America. {zrpan@tencent.com} [4]Biao Jia and Dinesh Manocha are with Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park. {biao,dm@cs.umd.edu}

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-7927-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IROS47612.2022.9981807

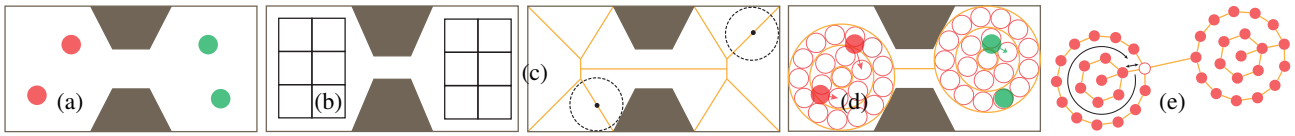

Fig. 1: (a): We have two robots moving from start (red) to target (green) positions. (b): Prior work such as [30] assumes that robots can move on a regular grid (black edges). (c):To extract the skeletons (yellow), our method first computes a medial axis transform, each point on which is the center of an inscribed circle. (d): Our method embeds the pebble-graph into $\mathcal{W}$ by picking two inscribed circles, in which robots are arranged into loops (separated by yellow loop boundaries). We further use subsets of the skeleton as paths connecting inscribed circles. Generally speaking, $\mathbf{s}^i$ and $\mathbf{t}^i$ do not coincide with the planned positions in the inscribed circles and we use local navigation to move robots there (arrows). (e): The topology of the embedded graph consists of six connected loops, on which a robot can move to a neighboring vacant position and robots on a loop can perform rotational motions (black edges).

unit radii, and we denote $B(\bullet)$ as a unit circle centered at $\bullet$. The robots have distinct goal positions $\mathbf{t}^{1,\cdots,N}$. Our method aims at finding a continuous path for each robot connecting $\mathbf{s}^i$ and $\mathbf{t}^i$, where the disk-shaped regions of any two robots do not overlap for any given time instance and no robot collides with obstacles.

Having defined the problem, we provide an overview of our approach. The first step of our algorithm is pebble-graph embedding (Section III). The pebble-graph is embedded into $\mathcal{W}$ with the help of Blum's medial axis analysis, which can be performed through robust algorithms such as [31]. The medial axis analysis extracts two crucial pieces of information from a continuous workspace: skeleton lines and inscribed circles, as illustrated in Figure 1 (c). An inscribed circle, denoted as $\mathcal{C}$, is defined as a circular subset of the workspace that touches the boundary of the workspace at least two points, and skeleton lines are derived by connecting centers of inscribed circles. These two pieces of information, skeleton lines and inscribed circles, are crucial because they allow the structure in $\mathcal{W}$ to move and accommodate robots: robots can reside in inscribed circles and move along subpaths of skeleton lines. As illustrated in Figure 1 (d), we design an algorithm to select a subset of inscribed circles where robots can be arranged into connected loops such that they can move to nearby vacant positions or rotate along loops in a collision-free manner as illustrated in Figure 1 (e). By construction, our pebble-graph is topologically identified with the setting considered in [11]. Our second step is motion planning (Section IV), where we compute a series of motions to answer arbitrary MPP queries. In other words, robots can perform arbitrary position permutations restricted to the graph vertices. Finally, start and goal positions might not coincide with graph vertices, and we use unlabeled, local navigations to move robots between their start/goal positions and graph vertices (Section V).

III. Pebble-Graph Embedding

The first step of our method finds $M > N$ graph vertex positions using the greedy Algorithm 1. These vertices should be close to robots' start positions, so that robots can be moved to vertices with a high success rate via local navigation algorithms such as [29]. We further assume each robots' goal position set is close to its start position set, so goal positions can be ignore in the graph construct step. Our method maintains: 1) a set of considered robot start positions $\mathbb{X}$ initialized to be empty; 2) a set of remaining positions initialized to be $\{\mathbf{s}^1,\cdots,\mathbf{s}^N\}$; 3) a set of considered inscribed circles $\mathbb{C}$ initialized to be empty. During each iteration, an arbitrary position in the remaining set is picked, e.g. $\mathbf{s}^i$. By the definition of inscribed circles, $B(\mathbf{s}^i)$ must be contained in at least one inscribed circle, and we select the circle whose center is closest to $\mathbf{s}^i$, denoted as $\mathcal{C}(\mathbf{s}^i)$. (To find $\mathcal{C}(\mathbf{s}^i)$, we sample the skeleton lines at regular intervals and extract inscribed circles centered at sample points, giving a discrete circle set $\bar{\mathbb{C}}$.) We then move $\mathbf{s}^i$ from the remaining set to the considered set $\mathbb{X}$ and move $\mathcal{C}(\mathbf{s}^i)$ into the circle set $\mathbb{C}$. Further, if there is another $\mathbf{s}^j$ in the remaining set such that $B(\mathbf{s}^j) \subset \mathcal{C}(\mathbf{s}^i)$, we also move $\mathbf{s}^j$ into the considered set. We terminate when the remaining set is empty. Note the success of our method depends on the order of choosing positions in the remaining set. To increase the success rate, we propose executing the algorithm multiple times using randomly shuffled start positions and return the first successful result. The resulting circle set would be used to construct our discrete loop graph using a BuildGraph function.

Remark 1. *The result of Algorithm 1 is a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with M vertices corresponding to $\mathcal{V} = \{\mathbf{v}^{1,\cdots,M}\}$ and edges $\mathcal{E}$ constructed in the BuildGraph function. If $\mathcal{G}$ is not connected, has only one loop, or $M \leq N$, then we immediately return failure because our pebble motion planner (Section IV) relies on these pre-conditions. Otherwise, we will use local navigation (Section V) to move robots from $\mathbf{s}^{1,\cdots,N}$ to N out of M positions. If robots get stuck during local navigation, we also return failure.*

In the following, we analyze the relationship between loops and derive geometric conditions of a graph embedding, such that robot motions are collision-free. Since two loops can belong to different inscribed circles or different layers of the same inscribed circle, we use the subscript to index circles and superscript to index loops inside a circle. For example, $\mathcal{C}_a^i$ indicates the ith loop (starting from the innermost one) of the ath circle.

A. A Single Circle

We refine the details of the BuildGraph function, which arranges robots into loops within inscribed circles, ensuring

Algorithm 1: Convert $\mathcal{W}$ into $\mathcal{G}$

1: Perform Blum's medial axis analysis [31] for $\mathcal{W}$
2: A discrete candidate circle set $\bar{\mathbb{C}}$
3: Initialize considered set $\mathbb{X} \leftarrow \emptyset$
4: Initialize selected circle set $\mathbb{C} \leftarrow \emptyset$
5: Initialize $\mathcal{G} \leftarrow < \emptyset, \emptyset >$
6: **for** $i = 1, \cdots, N$ **do**
7: **if** $\mathbf{s}^i \notin \mathbb{X}$ **then**
8: Find $\mathcal{C}(\mathbf{s}^i) \leftarrow \underset{B(\mathbf{s}^i) \subseteq \mathcal{C} \in \bar{\mathbb{C}}}{\text{argmin}} \|\text{center}(\mathcal{C}) - \mathbf{s}^i\|$
9: $\mathbb{C} \leftarrow \mathbb{C} \cup \{\mathcal{C}(\mathbf{s}^i)\}$
10: $\mathcal{G} \leftarrow$BuildGraph($\mathbb{C}$)
11: **for** $j = 1, \cdots, N$ **do**
12: **if** $\mathbf{s}^j \notin \mathbb{X} \wedge B(\mathbf{s}^j) \subseteq \mathcal{C}(\mathbf{s}^i)$ **then**
13: $\mathbb{X} \leftarrow \mathbb{X} \cup \{\mathbf{s}^j\}$
14: **if** Disconnected($\mathcal{G}$) $\vee$ $|\mathcal{V}| \leq N \vee \#\text{Loop}(\mathcal{G}) \leq 1$ **then**
15: **if** Maximal trial number not reached **then**
16: Random shuffle $\mathbf{s}^i$, $\mathbb{X} \leftarrow \emptyset, \mathcal{G} \leftarrow < \emptyset, \emptyset >$
17: Goto Line 5
18: **else**
19: Return Failure
20: **else**
21: Return Success

that translational and rotational motions are collision-free. We denote the ith loop as $\mathcal{C}^i \in \mathbb{R}^2$, where the 0th loop is the innermost loop. We use subscripts to distinguish different circles and superscripts to distinguish different loops. We have an upper bound on the number of loops $\mathcal{C}$ can hold as $0 \leq i \leq \lfloor (r(\mathcal{C}) + 1)/2 \rfloor$, where $r(\mathcal{C})$ is the radius. In addition, we denote $\#(R)$ as the number of prescribed positions that can be put into some subset region $R \subseteq \mathbb{R}^2$ without any collisions between positions or with $\partial\mathcal{W}$.

We begin with the simplest case where there is only one $\mathcal{C}$ in $\mathcal{G}$. Although a single robot can be put into $\mathcal{C}^0$, we let $\#(\mathcal{C}^0) = 0$ because our pebble motion planner requires each loop to have at least 3 positions (see Section IV for more details). For other loops, we have $\#(\mathcal{C}^1) = 6$ and for $i > 1$ we have:

$$\#(\mathcal{C}^i) = \left\lfloor (2\pi - 2\sin^{-1}\frac{1}{i})/(2\sin^{-1}\frac{1}{2i}) \right\rfloor + 2. \quad (1)$$

As illustrated in Figure 2, Equation 1 allows a capsule-shaped region between $\mathcal{C}^i$ and $\mathcal{C}^{i-1}$ to contain only the two positions, allowing a pebble motion to be performed between the two loops. A pebble or rotational motion inside a single $\mathcal{C}^i$ can be performed within $\mathcal{C}^i$ without affecting any other loops by having all the positions trace out a circular arc along the centerline of $\mathcal{C}^i$. In summary, the procedure to convert a single inscribed circle $\mathcal{C}$ into a graph $\mathcal{G}$ is Algorithm 2. Note that Algorithm 2 requires that $\lfloor (r(\mathcal{C}) + 1)/2 \rfloor \geq 2$, since our pebble motion planner requires more than one loop in $\mathcal{G}$. In addition, Equation 1 allows $\#(\mathcal{C}^i)$ positions to be placed along the centerline of $\mathcal{C}^i$ that are at least 2 apart, positions of different layers are non-overlapping.

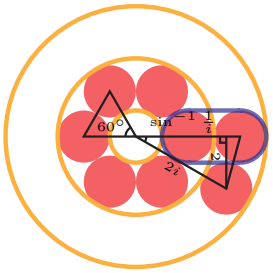

Fig. 2: We illustrate the angles defining $\#(\mathcal{C}^1)$ and $\#(\mathcal{C}^i)$ for $i > 1$. Our goal is to make sure the capsule-shaped region (blue) contains only the two robots.

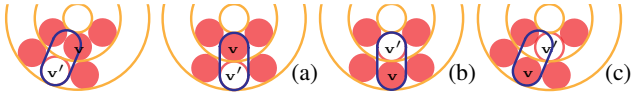

Fig. 3: To perform a pebble motion along $\mathbf{v} \leftrightarrow \mathbf{v}'$, we first move robots in $\mathcal{C}^i$ continuously to: (a) align the capsule region (blue); (b) perform the swap; (c) move robots in $\mathcal{C}^i$ backward.

Each pebble or rotational motion in $\mathcal{C}^i$ can be performed by moving robots along the centerline of $\mathcal{C}^i$ without affecting other layers. To realize pebble motions between two robots $\mathbf{v}$ and $\mathbf{v}'$ of neighboring layers (Line 9 of Algorithm 2), we first rotate robots continuously in $\mathcal{C}^i$ to make sure that the capsule-shaped region between $B(\mathbf{v})$ and $B(\mathbf{v}')$ does not contain other robots, as illustrated in Figure 3. After the pebble motion, we rotate robots in $\mathcal{C}^i$ back to their original positions.

Algorithm 2: BuildGraph for a single $\mathcal{C}$

1: **if** $\left\lfloor \frac{r(\mathcal{C})+1}{2} \right\rfloor < 2$ **then**
2: Return $\mathcal{G} = < \emptyset, \emptyset >$
3: **for** $1 \leq i \leq \left\lfloor \frac{r(\mathcal{C})+1}{2} \right\rfloor$ **do**
4: ▷ Pick positions that are at least 2 apart
5: Pick $\#(\mathcal{C}^i)$ positions along the centerline of $\mathcal{C}^i$
6: Insert the $\#(\mathcal{C}^i)$ positions into $\mathcal{V}$
7: Insert $\#(\mathcal{C}^i) - 1$ edges into $\mathcal{E}$
8: **if** $i > 1$ **then**
9: Choose $\mathbf{v}, \mathbf{v}'$ belonging to $i, i-1$th loop
10: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
11: Return $\mathcal{G}$

B. Two Overlapping Circles

Next, we discuss the case where two circles $\mathcal{C}_{a,b}$ are overlapping, which might result in loop sharing. Since the interiors of different layers in the same circle are disjointed, we always have the following decomposition of the domain:

$$\#(\mathcal{C}_a \cup \mathcal{C}_b) \geq \textstyle\sum_i \#(\mathcal{C}_a^i - \mathcal{C}_b) + \sum_j \#(\mathcal{C}_b^j - \mathcal{C}_a) + \sum_{i,j} \#(\mathcal{C}_a^i \cap \mathcal{C}_b^j), \quad (2)$$

where the inequality can be strict since we exclude robots that can cross the boundary of sub-domains. We choose to construct our graph using the lower bound on the righthand side. As illustrated in Figure 4, the three terms on the righthand side can be derived analytically by considering 4 different cases. These four cases are distinguished by the distance between center vertices of $\mathcal{C}_a$ and $\mathcal{C}_b$, denoted as D. In addition, we require that the center of $\mathcal{C}_a$ is outside $\mathcal{C}_b$ and vice versa:

$$D \geq \max(r(\mathcal{C}_a), r(\mathcal{C}_b)). \quad (3)$$

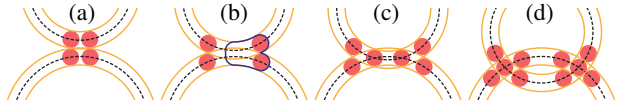

Fig. 4: We illustrate 4 critical cases between two loops of overlapping inscribed circles.

1) Case I: The first case is illustrated in Figure 4 (a), and it happens if the following condition holds:

$$D \geq \sqrt{(2i+1)^2-1} + \sqrt{(2j+1)^2-1} \tag{4}$$

In this case, we cannot fit any circle in the overlapping area, i.e. $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j) = 0$. The capacity of $\mathcal{C}_a^i$ is reduced to:

$$\#(\mathcal{C}_a^i - \mathcal{C}_b^j) = \left\lfloor \frac{2\pi - 2\cos^{-1}\frac{D^2+(2i)^2-(2j+2)^2}{4iD}}{2\sin^{-1}\frac{1}{2i}} \right\rfloor + 1, \tag{5}$$

and a symmetric equation applies to $\mathcal{C}_b^j$. Although the two loops overlap, they are not connected in $\mathcal{G}$ because $\mathcal{C}_a^i \cap \mathcal{C}_b^j$ is too narrow to perform pebble motions. It is trivial to show that rotational motions can be performed in either $\mathcal{C}_a^i$ or $\mathcal{C}_b^j$. We can convert this case by building two loops for $\mathcal{C}_a^i$ and $\mathcal{C}_b^j$ without adding edges to $\mathcal{E}$.

2) Case II: The second case is illustrated in Figure 4 (b), which happens if Equation 4 does not hold but we have:

$$D \geq 2i + 2j. \tag{6}$$

In this case, we still have $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j) = 0$ and Equation 5 holds. However, $\mathcal{C}_a^i \cap \mathcal{C}_b^j$ is now wide enough to allow an robot to travel in the blue region of Figure 4 (b) to swap with a vacant position, which also utilizes the blue region. Therefore, we can insert an edge into $\mathcal{E}_\mathcal{I}$ between two loops.

3) Case III: The third case is illustrated in Figure 4 (c), which happens if Equation 6 does not hold but we have:

$$D \geq 2i + 2j - 2. \tag{7}$$

In this case, we have $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j) = 2$ and we have:

$$\#(\mathcal{C}_a^i - \mathcal{C}_b^j) = (\text{Equation } 5) - 2.$$

Moreover, the two robots in $\mathcal{C}_a^i \cap \mathcal{C}_b^j$ can be swapped if one of them is vacant and rotational motions can be performed in either $\mathcal{C}_a^i$ or $\mathcal{C}_b^j$. Therefore, we can construct two loops for $\mathcal{C}_a^i$ and $\mathcal{C}_b^j$, let them share two robots in $\mathcal{C}_a^i \cap \mathcal{C}_b^j$, and the edge between them.

4) Case IV: The last case is illustrated in Figure 4 (d), and it happens if we have:

$$\max(r(\mathcal{C}_a), r(\mathcal{C}_b)) \leq D < 2i + 2j - 2. \tag{8}$$

In this case, we have $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j) = 2$, but $\#(\mathcal{C}_a^i - \mathcal{C}_b)$ has a new expression:

$$\#(\mathcal{C}_a^i - \mathcal{C}_b^j) = \left\lfloor \frac{2\cos^{-1}\frac{D^2+(2i)^2-(2j-2)^2}{4iD}}{2\sin^{-1}\frac{1}{2i}} \right\rfloor + 1 + \text{Equation } 5. \tag{9}$$

Similar to case III, we can construct two loops for $\mathcal{C}_a^i$ and $\mathcal{C}_b^j$, let them share two robots in $\mathcal{C}_a^i \cap \mathcal{C}_b^j$, but the two loops do not share any edges.

Note that we have computed $\#(\mathcal{C}_a^i - \mathcal{C}_b^j)$ in the four cases but we need $\#(\mathcal{C}_a^i - \mathcal{C}_b)$ in Equation 2, which can be computed by excluding each layer $\mathcal{C}_b^j$ from $\mathcal{C}_a^i$ (the details are similar to the four cases and omitted for brevity). We summarize our method to convert $\mathcal{W}$ to $\mathcal{G}$ for two overlapping circles in Algorithm 3. This algorithm inserts vertices corresponding to each term of the righthand side of Equation 2 in Line 2 and Line 5, respectively, and then inserts edges to connect loops. Note that we only need to insert inter-loop edges in Case II, as is done in Line 14.

Algorithm 3: BuildGraph for two circles $\mathcal{C}_{a,b}$

1: **for** $1 \leq i \leq \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ and $1 \leq j \leq \left\lfloor \frac{r(\mathcal{C}_b)+1}{2} \right\rfloor$ **do**
2: Insert $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j)$ vertices into $\mathcal{V}$
3: **for** pass=1, 2 **do**
4: **for** $1 \leq i \leq \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ **do**
5: Insert $\#(\mathcal{C}_a^i - \mathcal{C}_b)$ vertices into $\mathcal{V}$
6: Add $\#(\mathcal{C}_a^i - \mathcal{C}_b) + \sum_j \#(\mathcal{C}_a^i \cap \mathcal{C}_b^j)$ edges to $\mathcal{E}$
7: **if** $i > 1$ **then**
8: Choose $\mathbf{v}, \mathbf{v}'$ belonging to $i, i-1$th loop
9: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
10: Swap a, b
11: **for** $1 \leq i \leq \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ and $1 \leq j \leq \left\lfloor \frac{r(\mathcal{C}_b)+1)}{2} \right\rfloor$ **do**
12: **if** Case II holds **then**
13: Choose $\mathbf{v}, \mathbf{v}'$ from i, jth loop of $\mathcal{C}_{a,b}$, respectively
14: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
15: Return $\mathcal{G}$

Informally, we justify the correctness of Algorithm 3. First, since each summand in Equation 2 corresponds to pairwise disjointed regions, the embedding is collision-free. Second, it is trivial to show that pebble or rotational motions within a single circle can be performed in the same way as in Section III-A. In addition, pebble motions between two overlapping loops are only required in Case II, which can also be safely performed, as shown in Figure 4 (b). Note that we might not get a valid pebble graph in two scenarios: 1) when some loop might not have 3 vertices; 2) when two circles are not connected, leading to disconnected $\mathcal{G}$.

C. More Than Two Circles

We can combine the two previous cases to derive the final BuildGraph procedure. We assume that K inscribed circles $\mathcal{C}_{1,\cdots,K}$ are used and our algorithm is based on the assumption that, for any three (pairwise distinct) circles $\mathcal{C}_a, \mathcal{C}_b, \mathcal{C}_c$, we have:

$$\mathcal{C}_a \cap \mathcal{C}_b \cap \mathcal{C}_c = \emptyset. \tag{10}$$

As a result, we have the following inequality:

$$\#(\textstyle\bigcup_{a=1}^K \mathcal{C}_a) \geq \sum_{a,i} \#(\mathcal{C}_a^i - \bigcup_{b\neq a} \mathcal{C}_b) + \sum_{b<a} \sum_{i,j} \#(\mathcal{C}_a^i \cap \mathcal{C}_b^j), \tag{11}$$

which is valid only when Equation 10 holds. We can compute each of the terms in Equation 11 analytically. For a term of type $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j)$, we can compute it using the four cases in Section III-B. For a term of type $\#(\mathcal{C}_a^i - \bigcup_{b\neq a} \mathcal{C}_b)$, we use a procedure illustrated in Figure 5, where we first identify all the tangent cases (red circle) using triangular relationships (black), then find the angles between tangent cases (θ), and finally compute the number of spheres

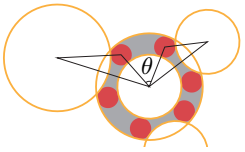

Fig. 5: We illustrate the procedure to compute $\#(\mathcal{C}_a^i - \bigcup_{b\neq a} \mathcal{C}_b)$.

that can be put into the interval between neighboring tangent cases as $\lfloor \theta/(2\sin^{-1}1/(2i)) \rfloor + 1$.

Algorithm 4: BuildGraph for $\mathcal{C}_{1,\cdots,K}$

1: **for** $1 \le a < b \le K$ **do**
2: **for** $1 \le i \le \left\lfloor \frac{r(\mathcal{C}_a)+}{2} \right\rfloor$ and $1 \le j \le \left\lfloor \frac{r(\mathcal{C}_b)+1}{2} \right\rfloor$ **do**
3: Insert $\#(\mathcal{C}_a^i \cap \mathcal{C}_b^j)$ positions into $\mathcal{V}$
4: **for** $1 \le a \le K$ **do**
5: **for** $1 \le i \le \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ **do**
6: Insert $\#(\mathcal{C}_a^i - \bigcup_{b\neq a} \mathcal{C}_b)$ positions into $\mathcal{V}$
7: Insert $\#(\mathcal{C}_a^i - \bigcup_{b\neq a} \mathcal{C}_b) + \sum_{b\neq a}\sum_j \#(\mathcal{C}_a^i \cap \mathcal{C}_b^j)$
8: edges into $\mathcal{E}$
9: **if** $i > 1$ **then**
10: Choose $\mathbf{v}, \mathbf{v}'$ belonging to $i, i-1$th loop
11: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
12: **for** $1 \le a < b \le K$ **do**
13: **for** $1 \le i \le \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ and $1 \le j \le \left\lfloor \frac{r(\mathcal{C}_b)+1}{2} \right\rfloor$ **do**
14: **if** Case II holds **then**
15: Choose $\mathbf{v}, \mathbf{v}'$ from i, jth loop of $\mathcal{C}_{a,b}$
16: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
17: **if** There is τ satisfying Equation 12 **then**
18: Set $i = \left\lfloor \frac{r(\mathcal{C}_a)+1}{2} \right\rfloor$ and $j = \left\lfloor \frac{r(\mathcal{C}_b)+1}{2} \right\rfloor$
19: Choose $\mathbf{v}, \mathbf{v}'$ from i, jth loop of $\mathcal{C}_{a,b}$
20: Insert $\mathbf{v} \leftrightarrow \mathbf{v}'$ into $\mathcal{E}$
21: Return $\mathcal{G}$

Fig. 6: A tunnel along the medial axis is added between two distant loops (a), so that an agent can be moved to a nearby vacant position by first rotating in the two loops and then moving the agent along the tunnel (b).

However, as shown in Section III-B, two scenarios might lead to invalid pebble-graphs that also apply for multiple circles. First, there may be invalid loops with less than 3 positions. In Section III-B, we eliminate this case by having two circles' centers outside each other, but this cannot be done for multiple circles. Second, two circles might be too far apart, leading to disconnected $\mathcal{G}$. This later scenario can be mitigated with the help of a medial axis. For two circles $\mathcal{C}_{s,t}$, their centers are on the medial axis. If we can find a sub-path $\tau : [0,1] \to \mathbb{R}^2$ along the medial axis such that:

$$\tau(0) \in \mathcal{C}_a \wedge \tau(1) \in \mathcal{C}_b \wedge$$
$$(\tau \oplus B(0)) - (\mathcal{C}_s \cup \mathcal{C}_t) \subseteq \mathcal{W} - \bigcup_{a=1}^{K} \mathcal{C}_a, \tag{12}$$

then we can insert an edge into $\mathcal{E}_\mathcal{I}$ between the outermost loops of $\mathcal{C}_s$ and $\mathcal{C}_t$. Here $\oplus$ is the Minkowski Sum, i.e. we require the path to have no interference with any obstacle or other circle. When performing pebble motions along τ, we follow the procedure illustrated in Figure 6. The motions can be performed in a collision-free manner when Equation 12 holds. The procedure to convert the K circles into $\mathcal{G}$ is summarized in Algorithm 4. This algorithm adds vertices corresponding to each term of the righthand side of Equation 10 in Line 3 and Line 6, respectively. It then adds three kinds of edges: 1) loop edges (Line 8); 2) inter loop edges (Line 11 and Line 16); and 3) medial axis edges (Line 20).

IV. Pebble Motion Planning

We allow robots to perform two types of movements on the pebble graph: 1) cyclic permutation of robots in a single loop; 2) movement of robot to an adjacent vacant position (either in the same loop or in an adjacent loop). This setting is similar to prior work in [17], but that method focuses on checking the feasibility, while we ensure feasibility of arbitrary robot configuration change on the graph under the assumption that the graph vertices are not fully occupied. Our pebbling motion planner does not differentiate between loops of the same circle or loops of different circles (because such differences have been taken care of in Section III). Therefore, we only use a global superscript to index loops. For the ith loop, the set of vertices is denoted as $\mathcal{V}_\mathcal{L}^i$. These vertices are connected by a set of edges denoted as $\mathcal{E}_\mathcal{L}^i$.

Formally, the topology of a pebble-graph $\mathcal{G} =< \mathcal{V}, \mathcal{E} >$ successfully constructed from Algorithm 1 is simple, undirected, and connected. Different loops can be interpreted as a partition of $\mathcal{V}$ as follows:

$$\mathcal{V} = \bigcup_{i=1}^{K} \mathcal{V}_\mathcal{L}^i, \quad |\mathcal{V}_\mathcal{L}^i \cap \mathcal{V}_\mathcal{L}^j| \in \{0, 2\}. \tag{13}$$

The partition of vertices induces a partition of edge set $\mathcal{E}$ as follows:

$$\mathcal{E} = \mathcal{E}_\mathcal{I} \cup \bigcup_{i=1}^{K} \mathcal{E}_\mathcal{L}^i, \quad \mathcal{E}_\mathcal{L}^i \cap \mathcal{E}_\mathcal{I} = \emptyset, \quad |\mathcal{E}_\mathcal{L}^i \cap \mathcal{E}_\mathcal{L}^j| \in \{0, 1\}. \tag{14}$$

Specifically, we assume that the vertices can be partitioned into $K > 1$ loops, where K is the number of loops in the graph. We also require that $\mathcal{E}_\mathcal{L}^i$ is the unique loop connecting $\mathcal{V}_\mathcal{L}^i$ and that $\mathcal{E}_\mathcal{L}^i$ is a simple loop (with no repeated vertices). Note that, under these definitions, each loop must have at least 3 vertices, i.e. $|\mathcal{V}_\mathcal{L}^i| \ge 3$. This is necessary because our motion planner moves robots by swapping the positions of two neighbors connected by a graph edge, where we need a third position as a buffer to accommodate temporary robots (see our extended version [32] for more details). Finally, we allow two loops to overlap; however, if the ith loop and the jth loop overlap, we require that they share exactly 2 vertices. As a result, two loops can also share a common edge (this is why we have $|\mathcal{V}_\mathcal{L}^i \cap \mathcal{V}_\mathcal{L}^j| \in \{0, 2\}$ and $|\mathcal{E}_\mathcal{L}^i \cap \mathcal{E}_\mathcal{L}^j| \in \{0, 1\}$). This corresponds to Case III of Section III-B. In summary, a successful graph returned by Algorithm 1 satisfies the following conditions:

Definition 1 (Pebble Graph)**.** *A pebble graph $\mathcal{G}$ is a simple, undirected, and connected graph satisfying Equation 13 and Equation 14 with $K > 1$ such that, through each group of vertices in $\mathcal{V}_\mathcal{L}^i$ ($i = 1, \cdots, K$), there is a simple, closed path formed by edges in $\mathcal{E}_\mathcal{L}^i$.*

On such a graph, we could follow similar reasoning as [9, 11] to verify feasibility for all the MPP instances and construct the corresponding motion plans, as summarized in the following result:

Theorem 1 (Pebble-Graph Feasibility)**.** *On a pebble-graph $\mathcal{G}$ with $|\mathcal{V}| = M > N$, we assume $\mathbf{s}^i = \mathbf{v}^i$ and $\mathbf{t}^i = \mathbf{v}^\sigma(i)$, where $\sigma(\bullet)$ is an arbitrary permutation. there exists a finite sequence of pebble or rotational motions to move robots from $\mathbf{s}^i$ to $\mathbf{t}^i$ for all $i = 1, \cdots, N$, and the length of the motion sequence is $\mathcal{O}(|\mathcal{V}|^2)$.*

We prove Theorem 1 constructively in our extended version [32] by converting the permutation into a sequence of pairwise position swaps between two neighboring vertices, and we use the proof to construct a planning algorithm to solve MPP instances in our extended version [32].We then show that each swap can be accomplished by a $\mathcal{O}(|\mathcal{V}|)$-sequence of motions. We further show that the amortized length of motion sequence for permuting the position of two arbitrarily distant vertices is also $\mathcal{O}(|\mathcal{V}|)$. Therefore, the total length of motion sequence to solve the pebbling problem is $\mathcal{O}(|\mathcal{V}|^2)$. To accomplish each position swap, we need to move the vacant vertex, which is not occupied by any robot, near the to-be-swapped vertices. We could optimize the motion plan by moving the vacant vertex along the shortest path. It is convenient to pre-compute the all-pair shortest paths, which costs $\mathcal{O}(|\mathcal{V}|^3)$. This step dominates the complexity of computing the motion sequence.

Bench.	Metric / Method	Precomp.	Planning	Total	Succ.	Makespan	Traj. Length
Figure 7 (a)/600	Ours	2	313	316	100%	73×600	178×600
	[16]	-	333	333	100%	81×600	171×600
	[25]	-	258	258	-	-	-
Figure 7 (b)/710	Ours	2	451	453	100%	82×710	163×710
	[16]	-	432	432	100%	76×710	166×710
	[25]	-	357	357	-	-	-
Figure 7 (c)/170	Ours	4	212	216	100%	33×170	122×170
	[16]	-	208	208	82%	28×170	118×170
	[25]	-	205	205	-	-	-
Figure 7 (d)/130	Ours	3	128	131	100%	26×130	132×130
	[16]	-	122	122	100%	23×130	136×130
	[25]	-	107	107	-	-	-

TABLE I: We compare the performance of different techniques. From left to right: benchmark/robots number, algorithm name, time to construct the pebble graph (in seconds); time to compute the MPP motion plan (in seconds); total computation time (in seconds); rate of success; makespan (average makespan of each robot × number of robots); trajectory length (average trajectory length of each robot × number of robots). All numbers are averaged over 50 random trials. Note the makespan (measured in the number of pebble steps) is incomparable with the trajectory length (measured in the unit length traveled by each robot).

V. Local Navigation

In general, robot start/goal positions do not coincide with the graph vertices and we use local navigations to move robots to/from graph vertices. To this end, we propose a heuristic method based on RVO [29] and CAPT [14]. The RVO algorithm resolves local collisions between robots, and CAPT further uses the Hungarian algorithm to solve unlabeled navigation problems by assigning robots to goal positions. Since our graph vertices do not coincide with robot start/goal positions, we use a similar technique to assign each $\mathbf{s}^i/\mathbf{t}^i$ to some graph vertex. This assignment can be arbitrary in our method because robots can be moved to any graph vertices and permuted later, while we propose computing an as-close-as-possible assignment via optimal transport by solving the following mixed integer linear programming:

$$\underset{z_{\mathbf{s}}^{ij}\in\{0,1\}}{\text{argmin}} \sum_{i=1}^{N}\sum_{j=1}^{M} z_{\mathbf{s}}^{ij}\|\mathbf{s}^i - \mathbf{v}^j\|$$
$$\text{s.t.} \sum_{j=1}^{N} z_{\mathbf{s}}^{ij} = 1 \wedge \sum_{i=1}^{N} z_{\mathbf{s}}^{ij} = 1,$$

where $z_{\mathbf{s}}^{ij} = 1$ implies assigning $\mathbf{s}^i$ to $\mathbf{v}^j$. After the assignment is computed, we can move each $\mathbf{s}^i$ to $\mathbf{v}^j$ using RVO. An identical procedure is used to assign $\mathbf{t}^i$ to $\mathbf{v}^j$ with decision variables denoted as $z_{\mathbf{t}}^{ij}$. Finally, the graph vertex permutation can be determined from $z_{\mathbf{s}}^{ij}$ and $z_{\mathbf{t}}^{ij}$. We set $\sigma(j) = j'$ if $z_{\mathbf{s}}^{ij} = 1$ and $z_{\mathbf{t}}^{ij'} = 1$ for some i.

VI. Experiments

We evaluate the performance of our method on a set of 5 benchmarks. The algorithm is implemented in C++ and tested on a desktop machine with an Intel Core i7 CPU running at 3.30GHz with 16GB of RAM. We have also compared our algorithm with [16, 25], which are recently proposed methods for centralized motion planning in continuous workspaces. Our method can compute motion plans for up to 200 robots in less than 10 seconds shown in Figure 7 (a) and (b), where the cost of pebble-graph embedding (Algorithm 1) is marginal and the majority of computation is spent on scheduling pebble motions on the graph. The detailed timing and trajectory qualities of the three methods are summarized in Table I. These results are derived by performing 50 random trials and taking the average. For each random trial, the robots' start positions are randomly sampled.

Our first benchmark is illustrated in Figure 7 (a), where we compare our method and prior work [16] under different robot densities. The robots' start positions are shown in red and their goal positions are derived by randomly permuting the starts. For 100 robots, our method takes 8 seconds to compute the motion plan (including pebble-graph construction and pebble motion planning, but not local navigation), while it takes 10 seconds to compute the motion plan using [16]. We then increase the number of robots to 200 in Figure 7 (a), where our algorithm still takes 8 seconds to find a motion plan, while the computational cost of [16] is 18 seconds using a rectangular grid as the graph. We can further increase the number of robots up to 600, resulting in robots occupying 47% of $|\mathcal{W}|$, and the motion plan can be computed within 313 seconds. We have tried a similar benchmark (Figure 7 (b)) with a different obstacle setup, where the overall computation time of our method is 2 seconds for 70 robots. We can increase the number of robots up to 710, occupying 55% of $|\mathcal{W}|$, for which our method computes a motion plan within 453 seconds. Our third benchmark in Figure 7 (c) involves a maple-shaped boundary with a narrow passage and our forth benchmark contains irregular obstacles. Both [16, 25] fail for these irregular cases. This is because the regular grid used by [16] does not fit into the narrow space and the well-separated assumption of [25] does not hold, In contrast, our method succeeds in computing a motion plan

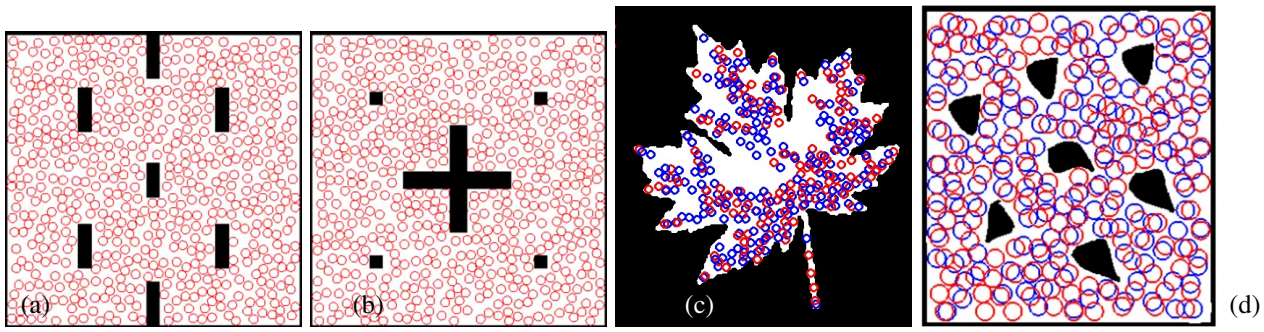

Fig. 7: We highlight the 4 most challenging benchmarks. (a): A rectangular workspace with 600 robots; (b): Another obstacle setup with 710 robots. The two benchmarks (ab) are also used in [16], where robots' goal positions are derived by permuting their start positions. (c): A maple-shaped workspace with highly irregular boundaries and 100 robots; (d): A rectangular workspace with irregular obstacles and 130 large robots. For (c) and (d), robots' start positions are in red and goal positions are in blue.

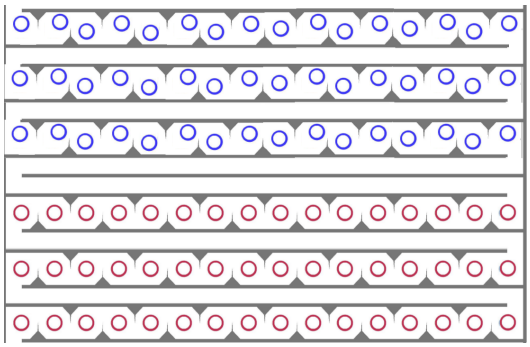

Fig. 8: A similar scenario to one used in [25], where robots need to give way to each other by revolving. The robots' start and end positions are in red and blue, respectively.

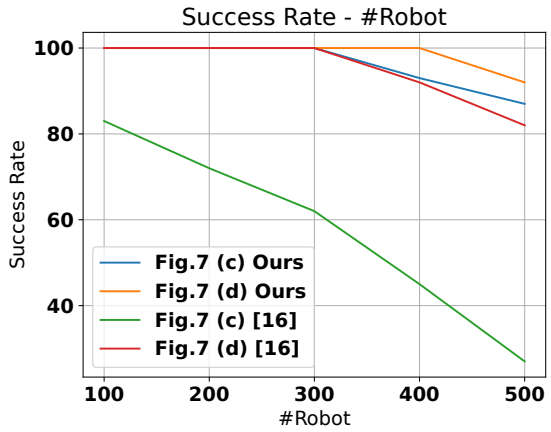

Fig. 9: We compare the changes in success rate over 50 random runs of our method and [16], under different robot densities. In cases of high density, our method exhibits a much higher success rate, especially with a narrow passage, as seen in Figure 7 (c).

within 220 seconds for 170 robots in Figure 7 (c) and 130 robots in Figure 7 (d), with a success rate of 100% over all 20 executions.

Furthermore, we implement a similar scenario to [25], but unlike their experiments, the robots in our experiments are randomly placed. As a results, some robots do not have sufficient free space to accomplish a revolving motion, which is needed in [25] to find feasible motion plans. However, our method successfully moves robots into inscribed circles via local navigation, and revolving motions can still be performed. These behaviors are illustrated in the video and the overall computation time is less than 3 seconds.

In Figure 9, we have profiled the success rate of our method and [16] under different scenarios and robot densities. Note that our method relies on a randomized, greedy Algorithm 1 to construct the pebble graph, so it is possible to improve our success rate by running Algorithm 1 multiple times using different random seeds until a solution is found, as in Line 15 to Line 17 of Algorithm 1.

VII. Conclusion and Limitations

We presented a new method to bridge the gap between continuous MPP planning and discrete pebble-graph motion. We first use medial axis analysis to extract critical information from the workspace, i.e. skeleton lines and inscribed circles. Using this information, we convert the free space into a pebble-graph via embedding. We show that motion planning on the pebble-graph is always feasible under mild assumptions and general MPP instances can be reduced to graph pebbling problems via local navigation. We conduct experiments using a set of 5 challenging benchmarks and achieve a 100% success rate under robot densities less than 37%. In Figure 10, the robots take up 68% of the free space, which implies that our method can work under extreme robot densities. The major limitation of our method lies in the overly conservative conditions derived in Section III which can leave some gap regions between robots. In large open areas, regular grids can have better space coverage [16]. Our future work would consider hybrid graph embedding techniques that combine multiple

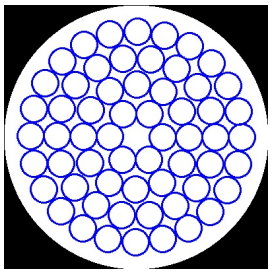

Fig. 10: In the extreme case, densely packed robots take up 68% of $\mathcal{W}$.

space tiling patterns, and we plan to derive improved boundary conditions for neighboring inscribed circles to accommodate more robots. Another limitation is that, our Algorithm 1 only considers robots' start positions, and ignores their goals. This works well if the goal set is derived by permuting the start positions (as in Figure 7 (ab)), but the local navigation can fail if goal sets are far from the start positions (as in Figure 7 (cd)).

REFERENCES

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

[2] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2186–2188.

[3] A. Malinowski, P. Czarnul, K. Czurylo, M. Maciejewski, and P. Skowron, "Multi-agent large-scale parallel crowd simulation," *Procedia Computer Science*, vol. 108, pp. 917–926, 2017.

[4] J. Hopcroft, J. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; pspace-hardness of the "warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[5] R. A. Hearn and E. D. Demaine, "Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theor. Comput. Sci.*, vol. 343, no. 1–2, 72–96, Oct. 2005.

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

[7] D. Le and E. Plaku, "Cooperative multi-robot sampling-based motion planning with dynamics," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, no. 1, pp. 513–521, 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] D. Moews, "Pebbling graphs," *Journal of Combinatorial Theory, Series B*, vol. 55, no. 2, pp. 244–252, 1992.

[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] T. S. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[13] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Meta-agent conflict-based search for optimal multi-agent path finding.," *SoCS*, vol. 1, pp. 39–40, 2012.

[14] M. Turpin, N. Michael, and V. Kumar, "Concurrent assignment and planning of trajectories for large teams of interchangeable robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 842–848.

[15] J. Yu, "Average case constant factor time and distance optimal multi-robot path planning in well-connected environments," *Autonomous Robots*, 2019.

[16] J. Yu and D. Rus, "An effective algorithmic framework for near optimal multi-robot path planning," in *Robotics research*, Springer, 2018, pp. 495–511.

[17] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," *ArXiv*, vol. abs/1507.03289, 2015.

[18] K. Solovey and D. Halperin, "K-color multi-robot motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 82–97, 2014.

[19] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. Bekris, "Rearranging similar objects with a manipulator using pebble graphs," in *2014 IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2014, pp. 1081–1087.

[20] M. Otte and N. Correll, "Dynamic teams of robots as ad hoc distributed computers: Reducing the complexity of multi-robot motion planning via subspace selection," *Autonomous Robots*, vol. 42, no. 8, pp. 1691–1713, 2018.

[21] A. Atias, K. Solovey, O. Salzman, and D. Halperin, "Effective metrics for multi-robot motion-planning," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1741–1759, 2018.

[22] D. Dayan, K. Solovey, M. Pavone, and D. Halperin, "Near-optimal multi-robot motion planning with finite sampling," in *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, IEEE, 2021, pp. 9190–9196.

[23] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3, pp. 443–467, 2020.

[24] A. Adler, M. De Berg, D. Halperin, and K. Solovey, "Efficient multi-robot motion planning for unlabeled discs in simple polygons," in *Algorithmic foundations of robotics XI*, Springer, 2015, pp. 1–17.

[25] I. Solomon and D. Halperin, "Motion planning for multiple unit-ball robots in rd," in *International Workshop on the Algorithmic Foundations of Robotics*, Springer, 2018, pp. 799–816.

[26] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," in *Robotics: Sciences and Systems*, 2015.

[27] S. Tang and V. Kumar, "A complete algorithm for generating safe trajectories for multi-robot teams,"

[28] J. Giesen, B. Miklos, and M. Pauly, "The medial axis of the union of inner voronoi balls in the plane," *Computational Geometry*, vol. 45, no. 9, pp. 515–523, 2012.

[29] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, Springer, 2011, pp. 3–19.

[30] 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.

[31] H. Blum and R. N. Nagel, "Shape description using weighted symmetric axis features," *Pattern recognition*, vol. 10, no. 3, pp. 167–180, 1978.

[32] L. He, Z. Pan, B. Jia, and D. Manocha, "Efficient multi-agent motion planning in continuous workspaces using medial-axis-based swap graphs," *ArXiv*, vol. abs/2002.11892, 2020.