Competitive Connected Multi-robot Exploration of Unknown Graphs

Dolev Mutzari[1] , **Yonatan Aumann**[1] , **Sarit Kraus**[1]
[1]Department of Computer Science, Bar-Ilan University
dolevmu@gmail.com, aumann@cs.biu.ac.il, sarit@cs.biu.ac.il

Abstract

Multi-robot graph exploration is a central problem in robotics, planning, and multi-agent systems. In this work, we consider the problem of exploring an unknown n-node graph by k robots that must remain connected throughout the process. Such a connectivity is frequently required for safety reasons, and naturally arises in real-world applications such as search-and-rescue and maintenance operations. We study the *overhead* imposed by not knowing the graph in advance, measured in terms of the *competitive ratio* of the number of exploration rounds necessary when the graph is unknown (versus the case that it is known).

We introduce a novel exploration procedure, DFS-BGS, to tackle the problem, and analyze its performance both theoretically and experimentally. On the theoretical end, DFS-BGS provably achieves a competitive ratio $\tilde{\mathcal{O}}(k^{1/3})$, for the case $n \leq k$. Empirically, we compare our online DFS-BGS to COCTA, the SOTA algorithm for trees that are known in advance. Examining the performance of the algorithms on real-world hotel floor plans as well as random graphs over a wide range of parameters, DFS-BGS incurs only a small slowdown, even with hundreds of robots and thousands of nodes.

1 Introduction

Multi-robot exploration and coverage are central problems in robotics, planning, and multi-agent systems. In applications such as search-and-rescue, environmental monitoring, and infrastructure inspection, a team of robots must visit every location in an environment, naturally modeled as a graph whose vertices correspond to locations and edges to traversable paths. When the graph is known in advance, the goal is to schedule robot motion so as to minimize the number of rounds needed to visit all vertices. This problem is called multi-robot, or collective, *graph coverage*. Notably, the problem is NP-hard even for trees [Fraigniaud *et al.*, 2006].

In many settings, however, the graph is unknown, so robots must discover and cover it online. Upon first visiting a vertex, its incident edges are revealed. This is *Collective Graph Exploration (CGE)* [Fraigniaud *et al.*, 2006]. In this setting, the central question is: what is the additional cost of online discovery, beyond pure coverage? This is captured by the *competitive ratio*, also known as *overhead*, defined as the ratio between online exploration performance and an optimal offline coverage [Dessmark and Pelc, 2004; Fraigniaud *et al.*, 2006]. Existing competitive strategies for CGE crucially rely on the ability of robots to disperse arbitrarily throughout the graph. The best existing algorithm for this problem achieves a competitive ratio of $\mathcal{O}(k/\exp(\sqrt{\log 2 \log k}))$ [Cosson, 2024], improving on the previous best algorithm of $\mathcal{O}(k/\log k)$. In contrast, the best known lower bound on the competitive ratio is $\Omega(\log k/\log\log k)$ [Disser *et al.*, 2020; Dynia *et al.*, 2007].

Connected Collective Graph Exploration (CCGE). Standard graph coverage and exploration models allow arbitrary dispersion, usually assuming *global communication*: robots share information regardless of their positions. In many applications, however, robots must remain in close proximity for communication, safety, supervision, or cooperative operation. Examples include rescue teams, swarms relying on multi-hop connectivity, or equipment transport teams. Such requirements significantly complicate parallel exploration, as robots cannot independently explore distant regions.

We therefore study *Connected Collective Graph Exploration (CCGE)*, a constrained variant of CGE in which occupied vertices must induce a connected subgraph at all times. Connectivity constraints significantly restrict robot mobility and coordination, making the exploration task qualitatively more challenging. Such constraints appear in related settings, e.g. multi-robot connected graph coverage (MRCGC) [Sinay *et al.*, 2017; Mutzari *et al.*, 2024], tethered robot Coverage Path Planning (CPP) [Mechsy *et al.*, 2017], and Multi-Agent Path Finding (MAPF) under continuous connectivity [Dutta *et al.*, 2019]. However, to the best of our knowledge, connectivity constraints have not been studied in the context of competitive online graph exploration.

Unlike standard CGE with global communication, connectivity allows us to replace the long-distance communication with local message passing through the connected robot team. Since the robots remain connected, even a communication radius of $k \ll n$ suffices, and in fact even a sublinear radius suffices in our proposed algorithm. Moreover, within $\mathcal{O}(k)$ communication rounds, or the diameter of the occupied

subgraph, all robots can communicate by forwarding messages between neighboring robots. Moreover, although we present centralized algorithms, they can be implemented distributively: robots may exchange information when gathering at a node, and can use local randomness to distribute among unexplored regions.

Our Contributions. We introduce CCGE, a new exploration model in which robots explore an unknown graph while remaining connected. Unlike prior work on collective exploration, our model prohibits arbitrary dispersion of robots and rules out most existing competitive strategies. We then present DFS-BGS, and analyze it theoretically and experimentally. For $k \geq n$, it achieves competitive ratio $\mathcal{O}(\min(\sqrt{D}, k^{1/3})\log(k))$, where D is the radius of the graph. Experimentally, we compare the online DFS-BGS to COCTA [Sinay *et al.*, 2017], the state-of-the-art offline algorithm. On random graphs and over a wide range of parameters, as well as real hotel floor plans from [Mutzari *et al.*, 2024], DFS-BGS exhibits only a small slowdown relative to COCTA, even for large values of k and n.

This shows that sublinear-overhead exploration is possible under connectivity constraints, narrowing the gap between theoretical exploration models and practical deployments.

Organization. In Section 3 we present our problem CCGE. We then gradually motivate our exploration algorithm as follows. In Section 4.1 we derive Picaboo, an exploration algorithm for bounded radius, in Section 4.2 we derive BGS, an exploration algorithm under bounded graph size and in Section 5 we derive DFS-BGS, an exploration algorithm for CCGE under no restrictions. Finally, Section 6 exhibits our experimental comparison with COCTA on a wide parameter range and families of trees.

2 Related Work

Multi-robot exploration and coverage have been studied extensively in robotics and multi-agent systems. We review the most relevant lines of work and clarify how they differ from our setting.

The study of online multi-robot graph exploration was initiated by [Fraigniaud *et al.*, 2006], who introduced Collective Tree Exploration (CTE), later generalized to graphs (CGE) [Dereniowski *et al.*, 2015]. In this model, robots explore an initially unknown graph under synchronous rounds and global communication. The objective is to minimize exploration time, typically analyzed via overhead, the ratio between the online algorithm and an optimal offline traversal. Even without connectivity constraints, minimizing exploration time is NP-hard, and much of the literature focuses on competitive guarantees and approximation algorithms. The work of [Cabrera-Mora and Xiao, 2012] studied variants with additional traversal restrictions and decentralized coordination using landmarks, but without explicit proximity constraints between robots. Subsequent works studied exploration in the regime of many robots, including the case $k \geq n$ [Dereniowski *et al.*, 2015].

However, these works allow robots to disperse arbitrarily in the graph, without connectivity or proximity constraints. Connectivity-constrained multi-robot coverage was introduced by [Sinay *et al.*, 2017] for offline coverage of trees; due to NP-hardness, they analyzed speedup relative to a single robot. This work is limited to trees, does not compare against optimal multi-robot solutions, and does not address online exploration. [Charrier *et al.*, 2020] studied coverage and multi-agent path finding with separate movement and communication edges, requiring connectivity to a base station, but focused on PSPACE-completeness results without algorithmic guarantees. More recently, [Mutzari *et al.*, 2024] generalized this framework to heterogeneous robots under broad proximity and movement constraints, providing FPT and PTAS offline algorithms for bounded-treewidth graphs. Overall, these works assume a known environment and focus on offline planning rather than online exploration.

In robotics, multi-robot exploration is often studied in the context of mapping and model reconstruction, see the survey by [Almadhoun *et al.*, 2019]. [Brass *et al.*, 2011] studied exploration of unknown graphs by multiple robots, but without connectivity constraints. [Banfi *et al.*, 2016] allowed robots to disconnect from the base station for arbitrarily long periods, reconnecting only to transmit information.

Another related line of work is Multi-Robot Coverage Path Planning (mCPP), which studies coverage of continuous environments. Works such as [Tang *et al.*, 2021; Lu *et al.*, 2023] consider physical constraints but do not enforce connectivity between robots. [Jensen and Gini, 2018] compared coverage algorithms under varying communication assumptions, while [Mechsy *et al.*, 2017] studied tethered robots with bounded cable length. Connectivity constraints are also central in swarm robotics. [Panerati *et al.*, 2018] and [Siligardi *et al.*, 2019] studied maintaining swarm connectivity during area coverage. More recent work considers UAV swarms with limited perception or repeated coverage in dynamic environments. However, swarm models typically rely on local rules, limited communication, and asynchronous operation, and do not allow centralized coordination or global planning.

To the best of our knowledge, no prior work provides competitive-overhead guarantees for online multi-robot exploration of unknown graphs under strict connectivity constraints. Our work fills this gap by providing the first sublinear-overhead exploration algorithms in this setting.

3 Model and Problem Definition

Consider a rooted, connected, undirected graph $\langle G, r\rangle$ where $r \in V(G)$, parameterized by its radius $D_r(G) = \max_{v\in V(G)} d(r,v) < \infty$, where $d(u,v)$ is the shortest path length between $u, v \in V(G)$. An edge $e = \{u,v\} \in E(G)$ exists if a robot can move directly from u to v.

A *configuration* of robots $\mathbf{x} : V(G) \to \mathbb{N}$ specifies how many robots occupy each vertex. In a *connected* configuration, the set of *occupied* vertices $\mathsf{Occupied}(\mathbf{x}) := \{v \in V(G) : x(v) > 0\}$ forms a connected induced subgraph of G. A simple interpretation is line-of-sight. A *transition* is a pair of connected configurations $(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}'$ can be reached from $\mathbf{x}$ by moving each robot along at most one incident edge. In particular, multiple robots are allowed to occupy a vertex simultaneously or pass through a common edge.

A *t-traversal* of G is a sequence of $t+1$ connected configurations $\mathcal{X} = (\mathbf{x}^0, \ldots, \mathbf{x}^t)$, which form t sequential transitions, where each vertex $v \in V(G)$ is visited at least once by at least one robot. In addition, in $\mathbf{x}^0$ all robots are at the root r.

In CGE, a team of k robots is initially located at the root r of an unknown graph G with edges of unit length. When a robot visits a node $v \in V(G)$ for the first time, the whole team becomes aware of its degree and of all unexplored edges adjacent to v, called *dangling edges*. When there are no more dangling edges, the entire team knows that all edges have been explored, and meets at the root to declare termination of exploration. The runtime of an exploration algorithm $\mathcal{E}$ is defined on a graph G as the number of rounds before termination and is denoted by $\mathsf{Runtime}(\mathcal{E}, k, G)$. By extension, for any two integers $n \geq D$, $\mathsf{Runtime}(\mathcal{E}, k, n, D)$ is the maximum number of rounds required before termination on any graph with n nodes and radius D. We refer to [Cosson *et al.*, 2023] for more details. Additionally, we define $\mathsf{Runtime}^*(k, G)$ as the minimal number of rounds required before termination if G is known in advance. The *overhead* of an exploration algorithm $\mathcal{E}$ is then defined as $Q(\mathcal{E}, k) := \sup_G \frac{\mathsf{Runtime}(\mathcal{E},k,G)}{\mathsf{Runtime}^*(k,G)}$. With slight abuse of notation, $Q(\mathcal{E}, k, D)$ can be defined by taking the supremum over graphs of radius D, and by definition, $Q(\mathcal{E}, k) = \sup_D Q(\mathcal{E}, k, D)$.

The *Connected Collective Graph Exploration (CCGE)* Problem is defined similarly. Given a rooted graph $\langle G, r\rangle$, a number of robots $k \in \mathbb{N}$ initially located at $r \in V$, find an *exploration algorithm* $\mathcal{E}$ with minimal overhead. The difference is that throughout the exploration, all configurations must be connected, and $\mathsf{Runtime}^*(k, G)$ is the minimal number of rounds required before termination if G is known in advance and robots must stay connected.

Notably, our approach is limited to undirected graphs, allowing robots to go back, regroup, and redistribute. Additionally, a connected team of robots is less effective in the case of sparse and deep graphs, as the connectivity constraint hinders effective coverage of the graph, even if the graph is known in advance.

4 CCGE with $k \geq n$ Robots

For the case of $k \geq n$, there are enough robots to occupy the entire graph simultaneously, and therefore an optimal vertex traversal will take precisely $D = \mathsf{radius}\langle G, r\rangle$ rounds. As for edge traversal, it would still take at least D rounds as all vertices will be visited.

However, in the exploration setting, D is unknown. To address this, we derive an exploration algorithm for the case $k \geq n$ in two steps. In Section 4.1 we derive Picaboo, an exploration algorithm parameterized by a bound Δ on the radius of the unknown graph, with an overhead Δ in case $\Delta > D$. In the following Section 4.2, we derive Baby-Giant Step, which uses Picaboo as a subroutine, and applies to any graph of size $n \leq k$. It achieves an overhead of $\tilde{\mathcal{O}}(\sqrt{D})$, which can be $\tilde{\mathcal{O}}(\sqrt{k})$ for graphs of radius $\mathcal{O}(k)$. We then utilize an early stopping technique to get the overhead complexity down to $\tilde{\mathcal{O}}(k^{1/3})$.

Algorithm 1 Picaboo(r, k, Δ)

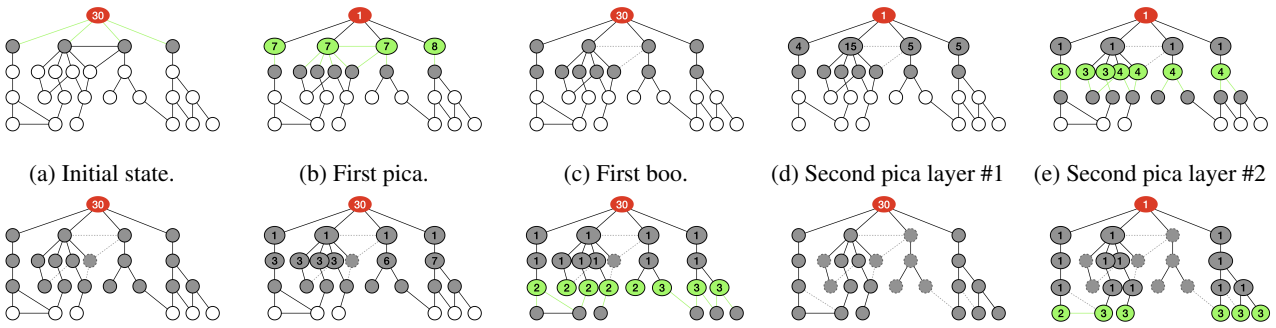
Input: A starting node $r \in V(G)$ of an unknown graph;
$k \in \mathbb{N}$ the number of robots;
$\Delta \in \mathbb{N}$ an upper bound on $\mathsf{radius}(G)$;
Output: A connected partial traversal $\mathcal{X}$ of the graph with k robots;
Initialize $\mathcal{X}$ with a single configuration with all robots at r;
for $i = 1, \ldots, \Delta$ **do**
Let G_i be the subgraph of size $n_i = |V(G_i)|$ discovered so far;
Let T_i be a spanning tree of G_i of depth i;
Pica: Send to each vertex in $v \in V(T_i)$ k_v robots, the number of vertices in the subtree of T_i rooted at v. If there are any robots remaining, divide them equally among the leaves of $\langle T_i, r\rangle$, and update $\mathcal{X}$;
Boo: Return with all robots back to the root r, and update $\mathcal{X}$;
return $\mathcal{X}$; // Either T is traversed or max depth Δ is reached

4.1 Bounded Radius

Essentially, the idea behind Picaboo (Algorithm 1) is to explore the graph in BFS, keeping connectivity by occupying vertices from previous layers with single robots. The problem is that since the graph is unknown, robots cannot know how to distribute the i^{th} layer in order to effectively continue to the $i+1^{\text{th}}$ layer. The algorithm therefore works iteratively, where in the i^{th} iteration the subgraph of radius $i-1$ from r is visited, and so the subgraph of radius i is known. To this end, the algorithm chooses a spanning tree for the subgraph of radius i of depth i, and sends to each discovered vertex the size of its discovered subtree number of robots. When the robots reach radius i, they all go back to the root r in i more rounds. At the end of the i^{th} iteration, which therefore takes exactly $2i$ rounds, all vertices in the subgraph of radius i are visited, and the subgraph of radius $i+1$ is observed, allowing advancement to the next iteration.

Figure 1 demonstrates Picaboo on a toy graph. $k = 30$ robots start at the root r marked in red in 1a, where dangling edges are colored green, and their incident unvisited vertices are colored in gray. In the first pica step, the robots are divided equally among the gray vertices 1b, after which the subgraph of radius two is observed. Then comes boo, where robots then gather back at the root 1c. We mark edges that are not part of the spanning tree in dashed gray, and Picaboo ignores them throughout the exploration. Then in the next two rounds 1d,1e, pica is used to occupy the layer at radius two from r. Exploration ends after reaching $D(G) = 4$ in 1j and gathering at r. Overall, the exploration time of this algorithm is therefore quadratic in the graph radius, and so its overhead is $\Theta(D)$. This can be a prohibitive $\Theta(k)$ overhead in case $D = \Theta(k)$. To bound the overhead, we parameterize the algorithm with a parameter Δ and let it stop if it reaches a radius of Δ from r. The first termination condition ensures the algorithm has an overhead at most Δ, and the second ensures there are enough robots to cover the tree in a single connected configuration. As a result, the algorithm may fail to explore the tree, but if successful, the overhead is bounded by Δ. This parameterized algorithm will serve as a subroutine for the unbounded-radius case.

Lemma 1. *Let $n, k, \Delta \in \mathbb{N}$ and let $\langle G, r\rangle$ be a tree of radius*



(a) Initial state. (b) First pica. (c) First boo. (d) Second pica layer #1 (e) Second pica layer #2

(f) Gather in 2 rounds. (g) Spread to layer #3. (h) Reach radius 3 from r. (i) Gather in 3 rounds. (j) Reach radius 4 from r.

Figure 1: A demonstration of Picaboo with $k = 30$ robots on a toy graph with $n = 26$ vertices and 33 edges.

D and size $n \geq k$. Algorithm Picaboo(r, k, Δ) maintains a connected configuration at all times. Moreover, after the i-th iteration, all vertices at a distance of at most i from r are visited. If $\Delta \geq D$, the algorithm explores G completely in $\Theta(D^2)$ rounds; otherwise, it terminates after $\Theta(\Delta^2)$ rounds having explored the subgraph of radius Δ.

4.2 Unbounded Radius

Next, we extend Picaboo and derive an algorithm that traverses any graph of size $n \leq k$ and depth $\leq D$, with overhead $\mathcal{O}((\Delta + \frac{D}{\Delta})\log(k))$. The algorithm starts by calling Picaboo(r, k, Δ). In case the traversal is done or a graph of size $> n$ is discovered, it returns. Otherwise, the entire subgraph of G of radius Δ has been traversed, its size is smaller than n, and we must continue exploration until either we discover more than k vertices, or the subgraph of depth D is successfully explored. However, we cannot continue with Picaboo using a larger Δ, as this would result in an overhead potentially greater than Δ and proportional to the actual radius of G. There are two reasons why, intuitively, it is possible to maintain the overhead of Δ. The first one is that the size of the graph G remains bounded, and the second is that we can exploit the information gained by discovering the subgraph of radius Δ. One may think of it as a baby-step giant-step algorithm, where in each baby step we discover more nodes in the graph and increase the observed radius by at most Δ using Picaboo, and each giant step redistributes the robots equally among the unexplored vertices with dangling edges. Intuitively, the giant steps therefore take $\Delta(1 + \ldots + (D/\Delta)) = \mathcal{O}(D^2/\Delta)$ rounds, or a D/Δ overhead as getting to radius Δj takes at least Δj rounds. The baby steps contribute to the established Δ overhead.

Figure 2 demonstrates BGS (Algorithm 2) on a toy graph, using Picaboo as a subroutine. $k = 35$ robots start at the root r marked in red in 2a. In the first picaboo step 2b, the subgraph of radius $\Delta = 5$ is visited, and all vertices in radius 6 are discovered, and marked in gray. Then, in the first giant step 2c, all robots gather at the root and then spread equally among the unvisited vertices, marked in green, except for robots needed to maintain connectivity at the internal vertices marked in gray. Next, Picaboo is executed in parallel in 2d from both green vertices, using 13 robots in each group. This results with complete exploration of the shallow subtree. Then in 2e a second giant step moves all robots to the chain side of the graph, where in 2f a final Picaboo is executed to fully explore the tree, and gather at the root 2g.

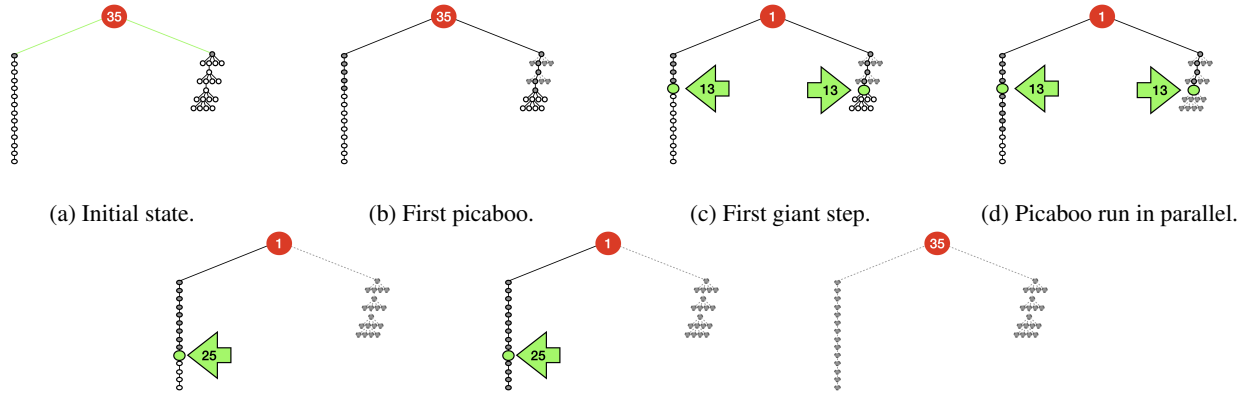
Algorithm 2 BabyGiantStep (BGS)(r, k, Δ)

Input: The root $r \in V(G)$ of a graph of size $n = |V(G)| \leq k$;
$k \in \mathbb{N}$ the number of robots;
$\Delta \in \mathbb{N}$ a parameter for baby-steps;
Output: A connected traversal $\mathcal{X}$ of G with k robots;
for $j = 0, \ldots, 4\lceil k/\Delta \rceil + \log_{4/3}(k) + 1$ **do**
Let G_j be the subgraph of size $n_j = |V(G_j)|$ discovered so far;
Let T_j be a spanning tree of G_j;
Let $L_j = L(T_j)$ be its set of $|L_j| = \ell_j$ leaves;
repeat $\lceil \log_2(k) \rceil$ times:
Giant-step: Group all robots at r, and then send one robot to every interior vertex of T_j, and divide the rest of the $k - n_j + \ell_j$ robots equally among the leaves L_j (in Round-Robin), and update $\mathcal{X}$;
Baby-step: Call Picaboo$(v, \cdot, \Delta)$ in parallel on all leaves $v \in L_j$, ignoring backward edges to $V(G_j) \setminus \{v\}$, and update $\mathcal{X}$;
return $\mathcal{X}$;

In what follows, we formally analyze the overhead of the BabyGiantStep (BGS) algorithm (Algorithm 2).

Lemma 2. *If $|V(G)| = n \leq k$, BGS explores the graph G, with overhead $\mathcal{O}((\Delta + \log(k) + D/\Delta)\log(k))$.*

Proof. Consider some iteration j, that starts with a discovered subgraph G_j. The first baby-step on each leaf $v \in L(G_j)$ may result in three potential outcomes:
1. Picaboo succeeds, and so the connected component of v in $G \setminus G_j$ is fully explored.
2. Picaboo fails, and so the subgraph of radius Δ from v contains more vertices than the number of assigned robots. In this case, a subgraph of size equal to the number of assigned robots, of the connected component of v in $G \setminus G_j$, is discovered.
3. Picaboo terminates, and the subgraph of radius Δ from v is discovered.



(a) Initial state. (b) First picaboo. (c) First giant step. (d) Picaboo run in parallel.

(e) Second giant step. (f) Last picaboo. (g) Gathering at root.

Figure 2: A demonstration of BGS with $k = 35$ robots on a toy graph with $n = 34 \leq k$ vertices, with $\Delta = 5 = \lfloor \sqrt{k}. \rfloor$.

However, after applying a second giant step, the leaves with small connected components will not be traversed again. Let $\ell_j^1 + \ell_j^2 + \ell_j^3$ be the number of picaboo calls in Cases 1, 2 and 3. If $\ell_j^1 \geq \ell_j^2 + \ell_j^3$, the number of leaves is reduced by a factor of two in each such repetition, and so after $\lceil \log_2(k) \rceil$ repetitions, $\ell_j^1 = 0$. Otherwise, $\ell_j^2 + \ell_j^3 > \ell_j/2$. We again divide this case into two cases. If $\ell_j^2 > \ell_j^3$, then $\ell_j^2 > \ell_j/4$. Let $k_j = |V(T_j)|$ be the number of robots occupying T_j for connectivity. Since all the rest of the $k - k_j$ robots are allocated to leaves, in this case, at least $(k - k_j)/4$ vertices are discovered. In particular, in every iteration j that this case happens, we have $k_{j+1} > k_j + (k - k_j)/4 = 3k_j/4 + k/4$. This is a linear recurrence, solving it with $k_1 = 1$ yields $k_j = k - (k-1)(3/4)^{j-1}$, and therefore this can also happen at most $1 + log_{4/3}(k-1)$ iterations. Finally, we are left with the case where $\ell_j^3 > \ell_j^2$ and therefore $\ell_j^3 > \ell_j/4$. In this case, a quarter of the leaves discover the radius Δ subgraph of their connected component in $G \setminus G_j$. Therefore, this happens at most $4D/\Delta$ times. As $D \leq n \leq k$, this concludes the correctness of the algorithm.

It remains to bound the algorithm's overhead relative to the optimal traversal. Since the tree is of size $n \leq k$, the optimal traversal time equals D, the radius of the graph. The analysis above shows that the number of iterations is bounded by $\mathcal{O}(D/\Delta + \log(k))$. In each iteration, $\mathcal{O}(\log(k))$ repetitions, and in each repetition, we make one baby step and one giant step. As for the giant step, the radius of the discovered subgraph G_j increases by at most Δ in each iteration (we can avoid the $\log(k)$ repetitions for leaves of Case 3), and therefore the radius $\mathsf{radius}(G_j) \leq \delta j$. In particular, the j^{th} giant step takes at most $2\delta j \log(k)$ rounds, and summing over j from 1 to $\mathcal{O}(D/\Delta + \log(k))$ gives overall $\mathcal{O}(D^2/\Delta \log(k) + \log^2(k))$ rounds. As for baby steps, each one takes at most Δ^2 rounds by construction, and therefore the overall number of rounds for baby steps is $\mathcal{O}((D/\Delta + \log(k)) \cdot \log(k) \cdot \Delta^2) = \mathcal{O}(D\delta \log(k) + \log(k)^2 \Delta^2)$. To conclude, the number of rounds is bounded by $\mathcal{O}((D^2/\Delta + (D + \log(k))\Delta)\log(k))$. Since any traversal of the graph takes at least the radius D, the overhead is bounded by $Q(k, D) = \mathcal{O}((D/\Delta + (1 + \frac{\log(k)}{D}\Delta)\log(k))$, or $\tilde{Q}(k, D) = \tilde{\mathcal{O}}(D/\Delta + \Delta)$ up to logarithmic factors in k. □

Corollary 3. *There exists an exploration algorithm with overhead* $Q(k, D) = \mathcal{O}(\max(\sqrt{D}, \log(k))\log(k)) = \tilde{\mathcal{O}}(\sqrt{D})$*, for* $k \geq n$.

Intuitively, this directly follows from Lemma 2 by setting $\Delta = \sqrt{D}$. However, D is not known to the exploration algorithm in advance. This can be tackled by running $\mathsf{BGS}(r, k, \Delta_i = 2^i)$ for increasing powers of two, until $\Delta_i > \sqrt{D}$, which would take at most $\lceil \log_2(D) \rceil$ BGS calls. Importantly, each such call will do Δ_i iterations, and not run up to k/Δ_i, effectively "guessing" that the radius of the graph is $\leq \Delta_i^2$. In particular, the number of rounds may double each BGS call, and so the geometric sum over the rounds in each call converges to a constant factor.

Theorem 4. *There exists an exploration algorithm with overhead* $Q(k) = \mathcal{O}(k^{1/3}\log(k))$.

Notably, plugging in $D = \Theta(k)$ as the worst case bound in Corollary 3 yields an overhead of $Q(k) = \mathcal{O}(k^{1/2}\log(k))$. However, one can improve by noticing that for such deep graphs, of depth $D = \Omega(k^{2/3})$, an optimal algorithm that knows the graph in advance still takes $\Theta(D) = \Omega(k^{2/3})$ rounds, which is what a DFS exploration with a single robot would take, and therefore the overhead compared to having all robots as one unit traverse the tree is only $\mathcal{O}(k^{1/3})$. We can therefore plug in $D = k^{2/3}$ and $\Delta = k^{1/3}$ and run BGS, and if the graph is not fully explored, simply traverse it with all robots as one unit in DFS. This yields an overhead of $\tilde{Q}(k) = \tilde{\mathcal{O}}(k^{1/3})$.

5 CCGE Under Unbounded Graph Size

The key idea is that under connectivity constraints, the exploration overhead should not depend on the graph size n for a fixed number of robots $k \ll n$. For instance, reaching distance $2k$ from r requires all robots to be at distance $> k$, so the

Algorithm 3 GraphCover

Input: A rooted graph $\langle G, r\rangle$;
Parameter $0 < \varepsilon < 1$;
Output: The induced subgraph G_τ remained to be covered, of size $|V(G_\tau)| < \frac{1}{\varepsilon}$;
A family $\mathcal{V}$ of induced subgraphs of G, with $|\mathcal{V}| \le n\varepsilon$, where the size of each subgraph G_τ is in $|V(G_\tau)| \in [\frac{1}{\varepsilon}, \frac{2}{\varepsilon}]$, and $\mathcal{P}(\mathcal{V} \cup \{\tau\})$ is an ε-tree-cover of T;

```
if r has no neighbors then return (({r}, ∅), r), ∅;
for u ∈ neighbors(r) do
    G_u, V_u ← GraphCover((G ∖ {r}, u), ε);
    Remove V_u and G_u from G; // avoid double coverage
Sort neighbors(r) by radius(G_u, u);
Initialize size ← 1;
for u ∈ neighbors(r) do
    size += |V(G_u)|;   G_r.add_subgraph.(G_u);
    V_r.add(V_u);
    if size > 1/ε then
        size ← 1; V_r.add(G_r, r); G_r ← ({r}, ∅);
return G_r, V;
```

radius-k neighborhood of r may effectively be traversed by the robots acting as a single unit. Accordingly, our algorithm partitions the discovered graph into subgraphs of size $\Theta(k)$, which are explored independently. Fully explored subgraphs are never revisited, while subgraphs found to be larger than expected are recursively partitioned. Each subgraph is explored using the BGS algorithm, which also detects whether its size exceeds k.

5.1 Graph Cover

First, we formalize the notion of covering the currently discovered subgraph with a collection of smaller subgraphs.

Definition 5. *A* graph-cover $\mathcal{P} = (\mathcal{T}, \mathcal{V})$ *of* G *is a tree* $\mathcal{T}$ *and a family* $\mathcal{V} = ((G_\tau, u))_{\tau \in V(\mathcal{T})}$ *of induced rooted subgraphs of* G *such that:*

1. ***Coverage****: It covers* V*, that is,* $\bigcup_{\tau \in V(\mathcal{T})} V(G_\tau) = V(T)$.
2. ***Connectivity****: For any* $\tau \neq \tau' \in V(\mathcal{T})$*,* $\{\tau, \tau'\} \in E(\mathcal{T})$ *iff* $V(G_\tau) \cap V(G_{\tau'}) \neq \emptyset$.

Notably, $\mathcal{V}$ uniquely identifies the graph-cover. Therefore, we can denote it by $\mathcal{P}(\mathcal{V})$. Next, we parameterize a graph-cover with the maximal size of a subgraph in the coverage. This is because we essentially want to divide the graph into $\Theta(n/k)$ subgraphs of size $\Theta(k)$:

Definition 6. *Let* $\langle G, r\rangle$ *be a rooted graph and* $\varepsilon > 0$*. An* ε*-graph-cover* $\mathcal{P} = (\mathcal{T}, \mathcal{V})$ *is a graph-cover where each subgraph* $G_\tau \in \mathcal{V}$ *satisfies* $|V(G_\tau)| \le \frac{2}{\varepsilon}$*, and* $|\mathcal{V}| \le n\varepsilon + 1$.

GraphCover (Algorithm 3) efficiently computes an ε-graph-cover. In particular, an ε-graph-cover always exists:

Lemma 7. *GraphCover runs in time* $\mathcal{O}(n \log(1/\varepsilon))$*, and if* G *is connected, returns a graph-cover* $\mathcal{P}(\mathcal{V} \uplus \{G_r\})$ *of* G.

Proof. As for complexity, GraphCover is a DFS search over G starting from the root r. After each visit, basic operations such as adding (a pointer to) a subgraph, adding (pointers/indices) of subgraph to the cover, and adding up / comparing $\mathcal{O}(\log(\frac{1}{\varepsilon}))$-bit numbers.

Coverage. Since DFS will scan the whole graph, eventually all vertices will appear in a subgraph in $\mathcal{V} \cup \{\tau\}$.

Connectivity. Each subgraph $G_u \in \mathcal{V} \uplus \{G_r\}$ is associated with a root $u \in V(G) \cap V(G_u)$. A path between G_u and G_r is ensured by induction, as each component G_u is connected, and the root u of G_u belongs to $V(G_{u'})$ of the caller GraphCover(G', u'). Furthermore, $\mathcal{T}$ is a tree, because we remove $\mathcal{V}_u$ and G_u from the graph before going to the next neighbor of r. □

In addition, GraphCover outputs an ε-graph-cover:

Lemma 8. *If* G *is connected,* GraphCover *returns an* ε*-graph-cover* $\mathcal{P}(\mathcal{V} \uplus \{\tau\})$ *of* G.

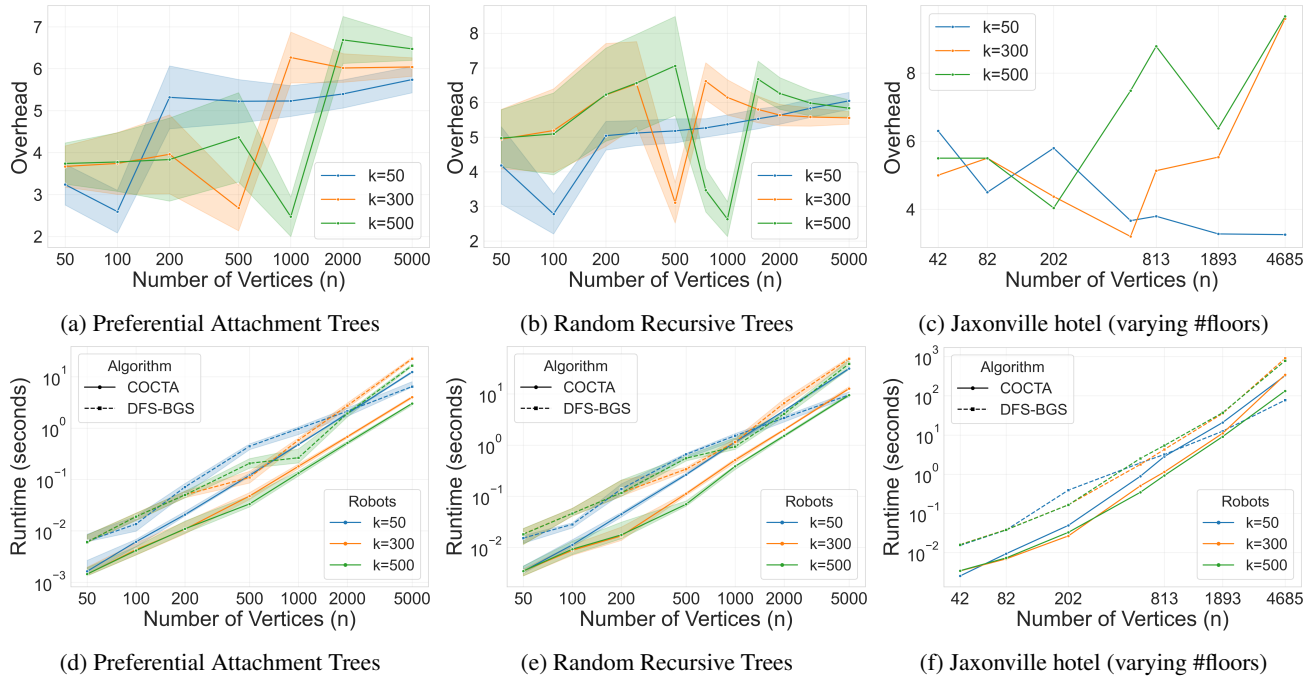
Proof. By Lemma 7, $\mathcal{P}$ is a graph-cover. The size of each added subgraph always coincides with size by induction. It is initialized as 1 to account for the current root r of G_r, and then as long as it doesn't pass $\frac{1}{\varepsilon}$, it is updated by adding the sizes of the subgraphs of each neighbor of u, which are correct by induction. It is important to avoid double-counting by removing $\mathcal{V}_u$ and G_u from the graph before partitioning and covering the remaining of the graph. Once a graph is added to $\mathcal{V}$, all vertices except the root are forgotten and the size of the tree is reset to 1.

Therefore, since subgraphs are added to $\mathcal{V}_r$ only after checking their size is greater than $\frac{1}{\varepsilon}$, we have that $|\tau| < \frac{1}{\varepsilon}$, as otherwise it would have been added to $\mathcal{V}$. Now, suppose in contradiction that there is a subgraph $G_u \in \mathcal{P}$ rooted at u of size greater than $\frac{2}{\varepsilon}$. Then there must be a neighbor of u for which GraphCover returned a tree of size $\ge \frac{1}{\varepsilon}$. But this is a contradiction, since we proved that the size of G_u is always strictly less than $\frac{1}{\varepsilon}$. Finally, denote by $P := |\mathcal{V}_r|$. Then $|V| = n = |V(G_r)| + \sum_{\tau \in \mathcal{V}} |V(G_\tau)| \ge 0 + P \cdot \frac{1}{\varepsilon}$. Therefore, $P \le n\varepsilon$. □

5.2 The Algorithm

We are now ready to formally specify our heuristic for graphs of unbounded size n, potentially $n \gg k$. Intuitively, GraphCover is used to divide the currently discovered graph into small graphs of (discovered) size $\Theta(k) \in [k/4, k/2]$. On each such subgraph, we call BGS to uncover $\Theta(k)$ new nodes (or to complete the exploration of the induced subgraph), and then recursively call the exploration algorithm to ensure the subgraph is explored completely.

DFS-BGS (Algorithm 4) is a recursive algorithm that first applies a BGS to uncover a subgraph of size $\ge k$, and then computes a graph cover of it, and recursively explores each subgraph in the graph cover in a similar fashion. Intuitively, this recursive behavior ensures the robots never traverse the same area of the graph G twice, and since each subtree in the tree cover is explored with overhead $Q(k)$ independent of n, and since they are of size $\Theta(k)$ which prevents exploring



(a) Preferential Attachment Trees (b) Random Recursive Trees (c) Jaxonville hotel (varying #floors)

(d) Preferential Attachment Trees (e) Random Recursive Trees (f) Jaxonville hotel (varying #floors)

Figure 3: Overhead and runtime of DFS-BGS vs COCTA for different families of trees, and varying number of robots k and tree size n. Each point is an average over ten samples.

Algorithm 4 DFS-BGS(r, k)

Input: A starting node $r \in T$ of an unknown rooted graph G;
$k \in \mathbb{N}$ the number of robots;
Output: A connected traversal $\mathcal{X}$ of the graph with k robots;
Call BGS$(r, k, \Delta = k^{1/3})$ and initialize $\mathcal{X}$.
Let G be the subgraph discovered so far;
Let $\mathcal{P}$ be an $\frac{4}{k}$ graph-cover of G;
Traverse the tree induced by the roots $\{r_\tau\}_{\tau \in \mathcal{T}(\mathcal{P})}$ in DFS, and update $\mathcal{X}$.
Whenever the robots reach r_τ where τ is a leaf in $\mathcal{T}$, call DFS-BGS$(r_\tau, k, \Delta = k^{1/3})$, removing all edges between G_τ and G, and update $\mathcal{X}$;
Return with all robots back to r_τ and update $\mathcal{X}$, and continue following the DFS, while updating $\mathcal{X}$;
Return with all robots back to r, and update $\mathcal{X}$;
return $\mathcal{X}$;

more than $\mathcal{O}(1)$ of them in parallel under connectivity, the overall overhead of the algorithm is expected to be $Q(k)$ as well. Unfortunately, since an optimal traversal that is given the tree in advance could explore the graph in a completely different way, this is left as a heuristic that we analyze with experiments in Section 6. From our observations, the decompositions strategy is most effective when the tree has shallow and dense components, as BGS detects them effectively with a relatively low overhead, and GraphCover effectively partitions and focuses on those components one at a time, exploiting the connected team of robots.

6 Experiments

We compare our algorithm to COCTA [Sinay *et al.*, 2017], the current state-of-the-art for *online* tree coverage. We use it as a *lower* bound on the overhead of our algorithm. Experiments are conducted on an Apple M2 Pro, 16GB RAM, single 3.5GHz CPU thread. Source code, datasets, synthetic graphs and raw experimental results are all available in the MRFGC GitHub repo.

We deploy two optimizations in our experiments: (i) In Picaboo, if robots can advance to the next level, we do so before gathering at the root; and (ii) when applying a BGS on a subtree from a tree cover, we exploit the fact that part of this subtree is already discovered, and we start with a giant step that deploys all robots to the leaves right away.

We consider two families of random trees: (i) Preferential Attachment Trees (PATS) [Albert and Barabási, 2002], where a leaf is added to a node with probability proportional to its degree. This results in relatively dense trees of low depth with a power-law distribution on the degrees of the nodes, where connected collective robot coverage can be most effective; and (ii) Rapidly-exploring random trees (RRTs) [LaValle and Kuffner, 2001], commonly used in robotics for motion planning and path finding, which grow by randomly sampling points in space and connecting them to the nearest node, producing low-degree low-depth trees.

We tested the algorithms for varying numbers of robots, $k = 50, 300$ and 500, and varying numbers of nodes $50 \leq n \leq 5,000$, thus examining the performance of the algorithm both for $k > n$ and $k < n$. The results are depicted in Figure 3 (note that the x axis is in logarithmic scale). Within the range we tested, the observed overhead was $2-6$. Notably, the compet-

PAT

n	$k = 50$	$k = 300$	$k = 500$
50	3.24 ± 0.14	3.67 ± 0.14	3.74 ± 0.14
100	2.59 ± 0.15	3.74 ± 0.22	3.78 ± 0.21
200	5.32 ± 0.22	3.96 ± 0.28	3.84 ± 0.29
500	5.22 ± 0.15	2.68 ± 0.16	4.37 ± 0.31
1000	5.23 ± 0.11	6.27 ± 0.18	2.47 ± 0.14
2000	5.40 ± 0.10	6.02 ± 0.10	6.69 ± 0.17
5000	5.74 ± 0.09	6.04 ± 0.07	6.47 ± 0.08

RRT

n	$k = 50$	$k = 300$	$k = 500$
50	4.19 ± 0.22	4.94 ± 0.17	4.97 ± 0.16
100	2.78 ± 0.11	5.19 ± 0.24	5.09 ± 0.23
200	5.04 ± 0.08	6.22 ± 0.29	6.23 ± 0.27
500	5.18 ± 0.07	3.11 ± 0.12	7.06 ± 0.28
1000	5.37 ± 0.05	6.15 ± 0.10	2.63 ± 0.10
2000	5.63 ± 0.05	5.64 ± 0.06	6.26 ± 0.09
5000	6.05 ± 0.05	5.56 ± 0.04	5.84 ± 0.05

Table 1: A 95% confidence interval for the average overhead on PAT and RRT trees.

itive ratio does not behave as $\sqrt{k}$, and indeed, it is not even clear if it is monotone in k, even for $k \geq n$. Empirically, the overhead exhibits two regimes with a transition around $n \approx k$. (i) When $n \leq k$, BGS is applied without graph decomposition. In this regime, COCTA incurs an additive overhead that BGS does not have, that was also observed in [Mutzari *et al.*, 2024]. As n grows, this additive cost becomes relatively smaller, leading to a mild increase in the measured overhead. (ii) When $n > k$, a phase transition occurs: BGS alone no longer suffices and the graph must be decomposed into subgraphs. For $n \sim k$, the initial BGS phase often terminates early due to the detection of a subtree of size k. Together with the optimization that increases the search radius without regrouping when possible, this reduces the number of costly iterations, resulting in a decrease in overhead. As n grows much larger than k, this advantage diminishes. The algorithm repeatedly applies the same BGS and GraphCover subroutines, and the relative overhead stabilizes.

We additionally tested our algorithm on the Jaxonville hotel floor plan studied in [Mutzari *et al.*, 2024], where the overhead seems to be within $2-10$. The overhead for this graph is expected to be larger, as its tree is highly *imbalanced*: along each hall many rooms come out as single leaf nodes. This means an offline coverage algorithm can send there single robots and continue exploring the hall, whereas an exploration algorithm cannot distinguish in advance an edge leading to a room from the continuation of the main hall. In our case, the optimized Picaboo will divide the robots equally between the continuation of the hall and the room edges, and so the number of available robots after each edge along a hall decreases exponentially, compared to linearly for COCTA.

Notably, the observed variance decreases with n. For $n > k$ the standard deviation of the overhead is typically between 0.3 and 0.5. For smaller n's, the variance is larger due to the variance of a single BGS run on small, dense and shallow trees. However, 100 samples for each data point are sufficient to obtain a strong statistical confidence. Accordingly, Table 1 reports a 95% CI of *mean* overhead per data point.

7 The Visibility and Proximity Spectrum

In this paper, we study multi-robot collective connected graph exploration (CCGE), generalizing multi-robot graph coverage (MRGC) along two axes: *visibility* and *proximity*. Visibility captures far ahead robots observe: in MRGC the entire graph is known in advance (visibility $D(G)$), whereas in collective graph exploration (CGE) robots observe only dangling edges. Proximity captures how far robots may separate: in MRGC robots may disperse arbitrarily (proximity $D(G)$), while in multi-robot connected graph coverage (MRCGC) they must remain connected. CCGE combines the constraints of CGE and MRCGC, setting both visibility and proximity to one. Figure 4 summarizes the relationships among these variants.

In general, one may define CCGE(δ, ε) with parameterized visibility and proximity constraints: robots observe vertices within distance δ and may separate up to distance ε. For $\delta = D(G)$ and variable ε, this yields a special case of multi-robot formation graph coverage (MRFGC) [Mutzari *et al.*, 2024]. Notably, our DFS-BGS algorithm naturally extends to arbitrary (δ, ε): in Picaboo, visibility δ allows increasing the explored radius by δ in each iteration, while proximity ε implies that only every ε-th layer must remain occupied, freeing additional robots for exploration.

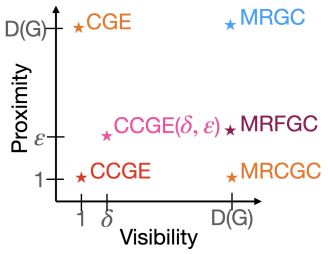


Figure 4: A diagram of existing multi-robot exploration problems.

8 Summary

We prove that multi-robot connected exploration has sublinear overhead over offline connected coverage for $n \leq k$. Empirically, we observe that overhead might be a small factor in practice, even for $n > k$. However, we stress that our experiments compare against an algorithm with no theoretical upper bound. One potential explanation could be that connectivity restricts both online and offline coverage and reduces the gap between the two. However, this proposition requires further investigation. Following this line, we believe that getting a theoretical upper bound for CCGE under $n \gg k$ is an important milestone left for future work.

Acknowledgments

This research has been partly supported by the Israel Science Foundation (ISF) under grants 3007/24 and 2544/24.

References

[Albert and Barabási, 2002] Réka Albert and Albert-lászló Barabási. Statistical mechanics of complex networks. In *Rev. Mod. Phys*. Citeseer, 2002.

[Almadhoun *et al.*, 2019] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, and Yahya Zweiri. A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, 1:1–24, 2019.

[Banfi *et al.*, 2016] Jacopo Banfi, Alberto Quattrini Li, Nicola Basilico, Ioannis Rekleitis, and Francesco Amigoni. Asynchronous multirobot exploration under recurrent connectivity constraints. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5491–5498. IEEE, 2016.

[Brass *et al.*, 2011] Peter Brass, Flavio Cabrera-Mora, Andrea Gasparri, and Jizhong Xiao. Multirobot tree and graph exploration. *IEEE Transactions on Robotics*, 27(4):707–717, 2011.

[Cabrera-Mora and Xiao, 2012] Flavio Cabrera-Mora and Jizhong Xiao. A flooding algorithm for multirobot exploration. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):850–863, 2012.

[Charrier *et al.*, 2020] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. Complexity of planning for connected agents. *Autonomous Agents and Multi-Agent Systems*, 34:1–31, 2020.

[Cosson *et al.*, 2023] Romain Cosson, Laurent Massoulié, and Laurent Viennot. Breadth-first depth-next: Optimal collaborative exploration of trees with low diameter. *arXiv preprint arXiv:2301.13307*, 2023.

[Cosson, 2024] Romain Cosson. Breaking the k/log k barrier in collective tree exploration via tree-mining. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4264–4282. SIAM, 2024.

[Dereniowski *et al.*, 2015] Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pajak, and Przemysław Uznański. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015.

[Dessmark and Pelc, 2004] Anders Dessmark and Andrzej Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326(1):343–362, 2004.

[Disser *et al.*, 2020] Yann Disser, Frank Mousset, Andreas Noever, Nemanja Škorić, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theoretical Computer Science*, 811:70–78, 2020. Special issue on Structural Information and Communication Complexit.

[Dutta *et al.*, 2019] Ayan Dutta, Anirban Ghosh, and O Patrick Kreidl. Multi-robot informative path planning with continuous connectivity constraints. In *2019 international conference on robotics and automation (ICRA)*, pages 3245–3251. IEEE, 2019.

[Dynia *et al.*, 2007] Miroslaw Dynia, Jakub Łopuszański, and Christian Schindelhauer. Why robots need maps. In *International Colloquium on Structural Information and Communication Complexity*, pages 41–50. Springer, 2007.

[Fraigniaud *et al.*, 2006] Pierre Fraigniaud, Leszek Gasieniec, Dariusz R Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks: An International Journal*, 48(3):166–177, 2006.

[Jensen and Gini, 2018] Elizabeth A Jensen and Maria L Gini. Online multi-robot coverage: Algorithm comparisons. In *AAMAS*, pages 1974–1976, 2018.

[LaValle and Kuffner, 2001] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[Lu *et al.*, 2023] Junjie Lu, Bi Zeng, Jingtao Tang, Tin Lun Lam, and Junbin Wen. Tmstc*: A path planning algorithm for minimizing turns in multi-robot coverage. *IEEE Robotics and Automation Letters*, 2023.

[Mechsy *et al.*, 2017] LSR Mechsy, MUB Dias, W Pragithmukar, and AL Kulasekera. A novel offline coverage path planning algorithm for a tethered robot. In *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, pages 218–223. IEEE, 2017.

[Mutzari *et al.*, 2024] Dolev Mutzari, Yonatan Aumann, and Sarit Kraus. Heterogeneous multi-robot graph coverage with proximity and movement constraints. *arXiv preprint arXiv:2412.10083*, 2024.

[Panerati *et al.*, 2018] Jacopo Panerati, Luca Gianoli, Carlo Pinciroli, Abdo Shabah, Gabriela Nicolescu, and Giovanni Beltrame. From swarms to stars: Task coverage in robot swarms with connectivity constraints. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7674–7681. IEEE, 2018.

[Siligardi *et al.*, 2019] Luca Siligardi, Jacopo Panerati, Marcel Kaufmann, Marco Minelli, Cinara Ghedini, Giovanni Beltrame, and Lorenzo Sabattini. Robust area coverage with connectivity maintenance. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2202–2208. IEEE, 2019.

[Sinay *et al.*, 2017] Mor Sinay, Noa Agmon, Oleg Maksimov, Sarit Kraus, and David Peleg. Maintaining communication in multi-robot tree coverage. In *IJCAI*, pages 4515–4522, 2017.

[Tang *et al.*, 2021] Jingtao Tang, Chun Sun, and Xinyu Zhang. Mstc*: Multi-robot coverage path planning under physical constrain. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2518–2524. IEEE, 2021.