Optimal Constrained Graph Exploration

CHRISTIAN A. DUNCAN

University of Miami

STEPHEN G. KOBOUROV

University of Arizona

AND

V. S. ANIL KUMAR

Virginia Tech

Abstract. We address the problem of constrained exploration of an unknown graph $G = (V, E)$ from a given start node s with either a tethered robot or a robot with a fuel tank of limited capacity, the former being a tighter constraint. In both variations of the problem, the robot can only move along the edges of the graph, for example, it cannot jump between nonadjacent nodes. In the tethered robot case, if the tether (rope) has length l, then the robot must remain within distance l from the start node s. In the second variation, a fuel tank of limited capacity forces the robot to return to s after traversing C edges. The efficiency of algorithms for both variations of the problem is measured by the number of edges traversed during the exploration. We present an algorithm for a tethered robot that explores the graph in $\Theta(|E|)$ edge traversals. The problem of exploration using a robot with a limited fuel tank capacity can be solved with a simple reduction from the tethered robot case and also yields a $\Theta(|E|)$ algorithm. This improves on the previous best-known bound of $O(|E| + |V| \log^2 |V|)$. Since the lower bound for the graph exploration problems is $\Omega(|E|)$, our algorithm is optimal within a constant factor.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Graph and tree search strategies*

A preliminary version of this article appeared in the *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, ACM, New York, 2001.

Some of this work was done while the authors were at the Max-Planck-Institut für Informatik.

S. Kobourov was partially supported by National Science Foundation (NSF) under Grant CCR-9625289.

Authors' addresses: C. Duncan, Department of Computer Science, University of Miami, Miami, FL, e-mail: duncan@cs.miami.edu; S. Kobourov, Department of Computer Science, University of Arizona, Tucson, AZ, e-mail: kobourov@cs.arizona.edu; V. S. A. Kumar, Department of Computer Science and Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA, e-mail: akumar@vbi.vt.edu.

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Computational learning theory, graph exploration

1. *Introduction*

Robot navigation is often modeled as the problem of exploring an unknown graph, $G = (V, E)$. The robot explores the graph beginning at a fixed start node s and performing edge traversals. As the robot explores G it constructs a map of the known portions of the graph. That is, once the robot has visited a node or traversed an edge, it can recognize them if it encounters them again; this can be done by marking the nodes [Fraigniaud et al. 2004] or using pebbles [Bender et al. 2002]. The graph $G = (V, E)$ is explored once all the nodes in V have been visited and all the edges in E have been traversed; we assume that the graph to be explored is connected. The performance *cost* of a search algorithm is generally measured in terms of the total number of *exploration steps* used by the robot to traverse all of the edges. Let $\ell(e) \geq 0$ denote the length of edge e. We say that the cost of exploring e is $\ell(e)$. If for all edges e, $\ell(e) = 1$, we have the unweighted case, where a trivial lower bound for the cost of an exploration algorithm measured by the number of edge traversals is $|E| + |V|$. The lower bound in the weighted case is $|V| + \sum_e \ell(e)$. Almost all the results on graph exploration only pertain to unweighted graphs. For the unconstrained exploration problem, the classical search algorithms, breadth-first search (BFS) and depth-first search (DFS) (e.g., see Cormen et al. [1990]), are optimal within a constant factor.

The piecemeal, or fuel-constrained, search model, introduced by Betke et al. [1995], adds two natural constraints to the graph exploration problem:

Continuity. The robot must traverse a continuous path in the graph, that is, it cannot jump from one point to another. This constraint is relevant in problems where the traversal corresponds to physical exploration of terrain.

Interruptibility. The robot must return to the start node after traversing C steps. C is typically at least $2(1 + \beta)r$, where r is the distance to the farthest node from the start node s and β is any nonnegative constant.

In this article, we consider another seemingly even more restricted problem of graph exploration: exploration by a tethered robot with a rope of length l. In this model, the robot is tied to the start node by a rope and is forced to match every forward traversal of an edge with a backward traversal, rewinding the rope, in a first-in last-out stack order. In practical terms, the rope can be a fuel line, a communication line, or a safety line.

Although the tethered robot is not constrained to return to s periodically, it might be forced to backtrack a great deal even to visit an adjacent node. However, the two models, the fuel-constrained and tethered search, are equivalent within constant factors, as shown in Section 2, that is, for appropriate rope and fuel constraints, the performance costs are equivalent within constant factors.

Note that neither the fuel-constrained search problem nor the tethered robot search problem can be solved using the classical search algorithms. BFS violates the continuity constraint and DFS violates either the interruptibility constraint or the maximum rope length, as shown in the examples of Awerbuch and Kobourov

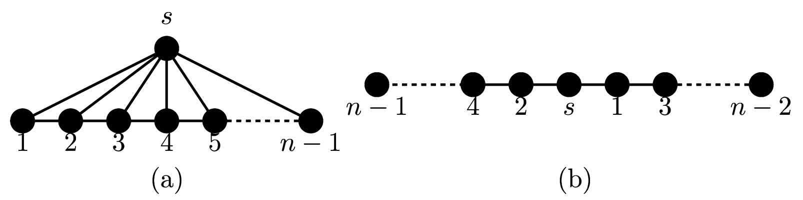

FIG. 1. Two example graphs illustrating failure of the DFS and BFS algorithms for constrained graph exploration. In both graphs, the start node is labeled s, and the other nodes are labeled by their traversal order. (a) A wheel graph with radius one. A Depth-First Search on this graph would require a rope of length n. (b) A single path with start node at the center of the path. A Breadth-First Search on this graph requiring continuity would take $\Theta(n^2)$ exploration steps.

[1998]; see Figure 1. We show how to solve both the piecemeal search and the tethered robot search problems by a conceptually simple Closest First Exploration (CFX) algorithm.

This article expands on the results presented in the extended abstract [Duncan et al. 2001]. In addition to providing detailed proofs for all the claims, we introduce a new version of the problem involving unknown graphs with weighted edges.

1.1. PREVIOUS WORK.

Exploration and navigation in unknown terrains is a well-studied problem; see the survey of Rao et al. [1993]. Exploration in geometric settings has been studied by Blum et al. [1997], Bar-Eli et al. [1994], Deng et al. [1998], and Hoffmann et al. [2002]. Papadimitriou and Yanakakis [1991] consider finding shortest paths in various unknown graphs and Blum and Chalasani [2000] give a “k-trip” shortest path algorithm.

Rivest and Schapire [1993] study a finite state model for exploration, while Deng and Papadimitriou [1999] and Albers and Henzinger [1999] model it on a directed graph. Bender and Slonim [1994] consider the problem with two cooperating robots. Randomized robot navigation was addressed by Berman et al. [1996]. Exploration by multiple robots has also been studied [Averbakh and Berman 1996, 1997; Fraigniaud et al. 2004].

In situations corresponding to physical settings the exploration is modeled by an undirected graph. Without the two piecemeal constraints (continuity and interruptibility) an undirected graph can be explored with cost $\Theta(|E|+|V|)$ using either BFS or DFS. Panaite and Pelc [1999] address the problem of minimizing the constants in such linear-cost explorations, achieving a bound of $|E| + \Theta(|V|)$.

The problem of piecemeal (or fuel-constrained) exploration of an undirected graph was introduced by Betke et al. [1995] where an $O(|E| + |V|)$ algorithm is presented for two classes of graphs: city-block graphs and grid graphs with rectangular obstacles. Awerbuch et al. [1999a] present an $O(|E|+|V|^{1+\epsilon})$ algorithm for general graphs and pose the open problem of finding a linear-cost algorithm for general graphs. Awerbuch and Kobourov [1998] employ sparse neighborhood covers [Awerbuch et al. 1999b] in an $O(|E| + |V| \log^2 |V|)$ algorithm.

The problem of finding a linear-cost algorithm has been open not only for general graphs but even for special classes of graphs such as planar graphs and grid graphs with nonconvex obstacles.

1.2. OUR RESULTS.

We settle the open problem posed in Awerbuch et al. [1999a] by presenting an algorithm for fuel-constrained and tethered robot exploration that runs with cost $\Theta(|E|)$. We show that G can be explored by a tethered robot with cost $\Theta(|E|+|V|/\alpha)$[1] using a rope of length $(1+\alpha)r$, where $\alpha > 0$ and r is the *radius*, that is, the distance to the farthest node from the start node s. We show in Section 2 that this implies an $O(|E|/\beta)$ bound for fuel-constrained exploration with a fuel tank of size $2(1 + \beta)r$, for any $\beta > \alpha$. Our bounds hold regardless of whether r is known in advance. When the radius is not known in advance, it can be determined incrementally.

We also consider the weighted case, where edges can be of varying lengths and show that the cost is still $O(\sum_{e\in E} \ell(e))$, but with a rope of length roughly $(1 + \alpha)r + \ell^*$ where ℓ^* is the length of the longest edge. As demonstrated in Figure 6, this bound is necessary. For a more precise definition, see Section 5.

The case with exploration of an unknown radius also leads to a solution for the *treasure hunting* problem defined in Awerbuch et al. [1999a], which requires searching for a treasure node t in an unknown infinite graph. Our algorithm solves this problem with cost $O(|G'|)$, where G' is the subgraph induced by nodes within a distance of $(1 + \alpha)d(s, t)$ from the start node, $d(s, t)$ being the distance of the treasure from s.

We describe the notation and the relationship between the two exploration models in Section 2. In Section 3, we present and analyze the `CFX` algorithm that solves the tethered graph exploration problem for unknown, unweighted graphs of known radius. In Section 4, we present and analyze a simple modification for cases when the radius is not known in advance. We also extend this solution to solve the treasure hunting problem. In Section 5, we extend our results to unknown, weighted graphs of unknown radius. Section 6 concludes with a few open problems.

2. *Preliminaries*

Let $G = (V(G), E(G))$ denote the (unweighted) graph to be explored with start node s. When the graph is understood, we typically refer to the set of vertices and edges as simply V and E. Let r, the *radius* of the graph, be equal to the distance to the farthest node from s. For now, we assume that r is known, or at least is bounded above by a known value. At every stage in our algorithms, we are also concerned with the complete known subgraph $G^* \subseteq G$ consisting of all edges and nodes visited so far.

Unless otherwise specified, for any graph $H = (V(H), E(H))$, let $|H|$ represent the number of nodes, as opposed to edges, in the graph. We note the exception that, for any path P, $|P|$ represents the number of edges. Let $P_H(v, w)$ be the shortest path in H between nodes $v, w \in V$. Let $d_H(v, w) = |P_H(v, w)|$ represent the distance between nodes v and w in H, that is, the number of *edges* in the path $P_H(v, w)$. We also expand our definition to include the distance between sets of nodes $U, W \subseteq V$. Let $P_H(U, W)$ be the shortest path in H between any $u \in U$ and $w \in W$. Similarly, we define $d_H(U, W) = \min_{u\in U, w\in W} d_H(u, w)$ to be the *minimum* distance between the two sets in H. In the same way, let $\Delta_H(U, W) = \max_{u\in U, w\in W} d_H(u, w)$ be the *maximum* distance between U and W in H. For convenience, we often drop the set notation for singleton sets. Thus, for $u \in V$ and $W \subseteq V$, $d_H(u, W) = d_H(\{u\}, W)$ and $\Delta_H(u, W) = \Delta_H(\{u\}, W)$.

[1] The inverse correlation between the cost and the terms α and β is due to the fact that the smaller the rope length or fuel tank the more unwinding or refueling steps required to perform the exploration.

Throughout our proofs, we use the following simple observation and related variants. If $H \subseteq G$ and $u, v \in V(H)$, then $d_G(u, v) \leq d_H(u, v)$ as any path in H is also a path in G.

Our algorithm requires grouping subgraphs into rooted trees. When describing such trees, we use the notation T_w to represent the tree rooted at vertex w.

Our tethered and piecemeal search algorithms require, respectively, a rope of length $(1+\alpha)r$ and a fuel tank of size $2(1+\beta)r$ for any constants $\alpha, \beta > 0$. Notice that the fuel tank must be at least twice the radius to allow the robot to reach the farthest node and return to the start node. As we show below, the two models are related within constant factors, and so we only describe algorithms for the tethered case.

LEMMA 2.1. *The fuel-constrained exploration problem with a tank of size $2(1+\beta)r$ can be reduced to the tethered exploration problem with a rope of length $(1+\alpha)r$, for any $\beta > \alpha$; the cost increases by a factor of $\frac{1+\beta}{\beta-\alpha}$. Conversely, for any $\alpha > \beta$, the tethered exploration problem with a rope of length $(1+\alpha)r$ can be reduced to the fuel-constrained exploration problem with a tank of size $2(1+\beta)r$, with a $\frac{2(1+\alpha)}{\alpha-\beta}$ factor increase in cost.*

PROOF. We first describe the reduction from piecemeal search to a tethered search. Let A be an algorithm that performs a tethered robot exploration, with a rope of length $(1+\alpha)r$, with cost T. An algorithm B for fuel-constrained exploration with a tank of size $2(1+\beta)r$, with $\beta > \alpha$ can be obtained in the following manner. Algorithm B emulates A with the addition that every $2(\beta-\alpha)r$ steps it sends the robot to s for refueling and back to the current node in A. Since the robot is always within distance $(1+\alpha)r$ to s, the amount of fuel needed to go to s and back is at most $2(1+\alpha)r$ and the total fuel used between refuelings is at most $2(\beta-\alpha)r + 2(1+\alpha)r = 2(1+\beta)r$. Since there is a refueling operation with cost at most $2(1+\alpha)r$ after at least $2(\beta-\alpha)r$ emulated steps of algorithm A, algorithm B has cost at most $T + T(1+\alpha)/(\beta-\alpha) = T(1+\beta)/(\beta-\alpha)$.

Similarly, let B be an algorithm that performs a fuel-constrained exploration with a tank of $2(1+\beta)r$ with cost T. To transform B into an an algorithm A for a tethered robot exploration with a rope of length $(1+\alpha)r$, we again emulate B. If at any time the rope length used by the emulation A reaches $(1+\alpha)r$, A unwinds the robot, by retreating to the root, and then takes the shortest known path back to the original position and continues emulating. The key observation here is that the robot in B must never go to a node where the known distance to the start node is more than $(1+\beta)r$. The reason is that if the robot extends beyond this limit it cannot guarantee a means to refuel in time. So after unwinding, when the robot in A reaches, via the shortest known path, its original position to resume emulating, the remaining length of rope is at least $(1+\alpha)r - (1+\beta)r = (\alpha-\beta)r$. This guarantees that another unwinding step will not occur until after at least $(\alpha-\beta)r$ steps. Consequently, the entire unwinding step has a cost of at most $(1+\alpha)r + (1+\beta)r$ and is executed at most every $(\alpha-\beta)r$ steps. Therefore, the total cost of the emulation is at most $T + T(2+\alpha+\beta)/(\alpha-\beta) = 2T(1+\alpha)/(\alpha-\beta)$. □

We note that the second emulation presented above does not take advantage of the typically wasted refueling process, blindly emulating these refueling steps. Ignoring these steps would only improve the constant factors. But, it also implies that the tethered search is equivalent within constant factors to a *distance-constrained*

```
bDFX($v, l$)
  if $v$ is *tagged*
    return
  if $l = 0$
    // Maximum length reached.  Must backtrack
    if $v$ has any unexplored edges
      Mark $v$ as *incomplete*
    else
      Mark $v$ as *explored*
    return
  Mark $v$ as *tagged*
  For each *unexplored* edge $(v, w) \in E$
    MOVE ROBOT to $w$ along edge $(v, w)$
    Mark $(v, w)$ as *explored*
    Call bDFX($w, l - 1$)
    MOVE ROBOT BACK from $w$ to $v$ along edge $(v, w)$
  Mark $v$ as *explored*
```

FIG. 2. The recursive bDFX algorithm from node v with a remaining rope length of l.

search, whereby the robot is constrained to lie only within a given known distance from the start node with no tether or fuel restriction.

3. *Efficient Exploration of Unweighted Graphs*

As mentioned above, we focus on the tethered robot case and assume that the robot has a rope of length $(1 + \alpha)r$ for some $\alpha > 0$ and with known radius r.

3.1. BOUNDED DEPTH FIRST EXPLORATION. Before presenting the optimal solution for tethered robot exploration we briefly examine some related non-optimal solutions, one of which is necessary in the final algorithm. Let us assume now that the robot has a rope length $l = |V|$. For any graph $G = (V, E)$ notice that $r < |V|$. It is easy to explore a graph using a tethered robot with rope length $l = |V|$ using a DFS traversal. In fact, for a tethered robot with such rope length, DFS is optimal. The algorithm takes exactly $2|E|$ steps as every edge is traversed twice, and, since a tethered robot must backtrack over every step, this is optimal. If we allow the rope to be of infinite length, the problem reverts to the unconstrained robot exploration. Here, one may explore the graph optimally with a cost of $|E| + \Theta(|V|)$ [Panaite and Pelc 1999]. However, if the length of the rope is $O(r)$, there are simple cases in which the standard DFS fails [Awerbuch and Kobourov 1998]. More specifically, the behavior of DFS is not well-defined when the robot reaches its maximum rope length. It certainly cannot continue further, but how does it proceed?

One obvious solution is to allow the robot to ignore any edges it cannot reach and proceed by backtracking, returning along the rope. Thus, the search has a bounded depth constraint associated with it. For clarity and because our main solution builds off of this simple notion, we present this bounded depth first exploration (bDFX) algorithm in Figure 2.

We classify the edges of G into two types, *explored* and *unexplored*. Similarly, the nodes are of four types, *explored*, *unexplored*, *incomplete*, and *tagged*. Initially, every edge and node is unexplored. Incomplete nodes are those that have been partially explored, but have some incident unexplored edges. Thus, a node becomes

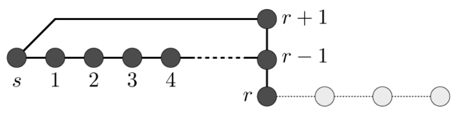

FIG. 3. An example of a bad graph traversal using bDFX. The nodes are labelled by their depth first traversal order. Since node r has depth r at the time visited, the white nodes cannot be explored. After exploring edge $(s, r+1)$, we discover a path between s and r with length only 3. However, in bDFX, these edges $(r+1, r-1)$ and $(r-1, r)$ cannot be traversed again. Thus, the white nodes must remain unexplored.

explored only when all of its incident edges are explored. Tagged nodes are used by the bDFX algorithm to mark nodes on the search stack temporarily.

The recursive bDFX algorithm simply starts at a node v with a given rope length l and, if $l > 0$, recursively visits all unexplored edges incident to v using a rope of length $l - 1$. At the end of the call, the robot returns to its original position v with the same initial rope length. From the algorithm, one can see that the amount of steps taken by the robot is exactly $2|E'|$ where E' is the set of edges marked *explored* by bDFX.

At first glance, this simple modification may appear to solve the problem of tethered exploration. Unfortunately, this is not the case, as shown in Figure 3. After termination, the algorithm may leave many edges *unexplored*.

3.2. OPTIMAL TETHERED EXPLORATION. In order to completely explore the graph, we enhance the bDFX algorithm to run in phases until all nodes and edges are explored. Recall that at any stage $G^* \subseteq G$ is the subgraph consisting of all visited edges and nodes, those labelled explored or incomplete. At each phase in our algorithm, we maintain a collection $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ of node disjoint subtrees of G^* whose union contains all currently *incomplete* nodes, possibly with other nodes. Initially, this set consists of one subtree containing only s. For each tree $T_i \in \mathcal{T}$, we are interested in (one of) the node(s) $s_i \in T_i$ closest to s in the graph G^*. That is, we want an $s_i \in T_i$ such that $d_{G^*}(s, s_i) = d_{G^*}(s, T_i)$. For each phase, we choose the tree $T_i \in \mathcal{T}$ closest to s, send the robot to s_i, and attempt to explore the incomplete nodes in T_i, by doing a regular DFS on T_i and a bDFX on each incomplete node. This process in turn creates new trees of incomplete nodes that we then add to $\mathcal{T}$. For convenience, we call a tree *incomplete* if it contains any incomplete nodes and *complete* otherwise.

Ideally, the robot would explore the entire subtree T_i, visiting and locally exploring all of T_i's incomplete nodes. Unfortunately, such an approach would lead to an inefficient algorithm, particularly, when the robot attempts to reach an incomplete node of T_i that is deep in the search tree. Our approach works by pruning T_i so that it has bounded depth, adding the leftover subtrees into $\mathcal{T}$ as well. Conversely, we cannot prune the tree too small, as this would also negatively impact the performance. Using a small tree risks having the algorithm spend too much time going back and forth from the start to the root of the tree relative to exploring the tree itself. Therefore, we guarantee a lower bound on the size of the pruned trees.

We perform pruning by breaking T_i into smaller subtrees $T_{i_0}, T_{i_1}, \dots, T_{i_k}$, and replacing these subtrees for T_i in $\mathcal{T}$. The actual exploration is performed on the pruned tree containing the original closest node s_i. Pruning a tree at a node $w \in T_i$, works by removing the edge between w and the parent of w, creating two new trees, $T_i \setminus T_w$ and T_w. We determine whether or not to prune the tree at a particular node

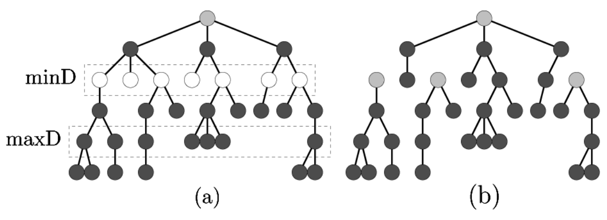

FIG. 4. A simple example of pruning a tree with minD = 2 and maxD = 4. (a) The initial tree with all nodes at depth minD highlighted. (b) The resulting trees formed after pruning.

based on two values minD and maxD with maxD > minD. We prune a node $w \in T_i$ if and only if w is at depth minD and T_w contains nodes at depth (with respect to T_i) greater than maxD; see Figure 4. As we prove later, this guarantees that the maximum depth on the original pruned T_i is maxD and that no pruned subtree has depth less than maxD − minD. The specific values of minD and maxD are chosen so that we are able to maintain a *minimum-size criterion* for every tree in $\mathcal{T}$. That is, after each phase, for all $T \in \mathcal{T}$, $|T| \geq \alpha r/4$. For convenience, we assume that $\alpha r/4$ is an integer.

Figure 5 describes the core elements of the CFX algorithm. The idea is to pick the incomplete tree T_i closest to the start node s, prune it, and explore it, all the while maintaining the set of all incomplete trees, with intersecting trees merged. The process is repeated until there are no remaining incomplete trees.

The explore function moves the robot to $s_i \in T_i$ via the shortest known path and proceeds to do a depth first traversal of T_i. The function then runs bDFX on all of the incomplete nodes in T_i as they are encountered. In the end a spanning forest of all *new* explored edges is created and added to $\mathcal{T}$. The robot is then moved back to the starting node.

The total cost of the exploration is equivalent to the cost of getting from s to $s_i \in T_i$ and back plus the cost to visit all incomplete nodes in T_i plus the cost needed to perform the various iterations of bDFX. The only mysterious step lies in the pruning operation, explained intuitively earlier, and whose importance becomes apparent in the analysis.

3.3. ANALYSIS. Intuitively in our analysis, what we do is group the total sequence of steps taken by the robot into three parts: roundtrip steps, tree-traversal steps, and exploration steps. For each incomplete tree T_i chosen, the roundtrip steps account for getting from s to $s_i \in T_i$, and the tree-traversal steps account for the cost to visit all incomplete nodes in T_i. The exploration steps account for the steps taken in the calls to bDFX, which essentially is the number of edges marked explored. We can therefore bound the total number of exploration steps to $\Theta(|E|)$. The tree-traversal steps, which are part of a depth-first traversal of each tree, can be bounded to the size of the trees. Next, we guarantee that each tree T_i visited has at least a certain minimum size that is roughly a factor of the distance from the start node to the tree. Now, we can also bound the roundtrip steps to a factor of the size of the trees. Since the trees chosen are more-or-less disjoint,[2] we can bound the size of the trees and consequently the number of roundtrip and tree-traversal steps

[2] In fact, each vertex appears in at most two chosen trees.

```
CFX($s, r, \alpha$)
  Initially $\mathcal{T} = \{s\}$
  Repeat
    Let $T_i \in \mathcal{T}$ be the closest tree to $s$ in $G^*$
    Let $s_i$ be the closest node in $T_i$ to $s$
    prune($T_i, s_i, \alpha r/4, \alpha r/2$)
      producing $\mathcal{T}_i = \{T_{i_0}, T_{i_1}, \ldots T_{i_k}\}$
    // Here $T_i = \cup_{T \in \mathcal{T}_i} T$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \{T_i\} \cup \mathcal{T}_i$
    Let $T \in \mathcal{T}_i$ such that $s_i \in T$
    explore($T, s_i, (1 + \alpha)r$)
    remove all complete trees from $\mathcal{T}$
    merge any trees in $\mathcal{T}$ with common nodes
  Until $\mathcal{T} = \emptyset$

prune($T, v$, minD, maxD)
  If $T = \{s\}$, return
  Root $T$ at $v$
  $\mathcal{T}_i \leftarrow \{T\}$
  for each node $w \in T$ such that $d_T(v, w) =$ minD
    Let $T_w \subseteq T$ be the subtree of $T$ rooted at $w$
    if $\Delta_T(v, T_w) >$ maxD
      // Separate subtree $T_w$ from $T$
      $T \leftarrow T \setminus T_w$
      $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{T_w\}$

explore($T, s_i, l$)
  MOVE ROBOT from $s$ to $s_i$ via shortest known path
  MOVE ROBOT using depth first traversal on $T$
    for each incomplete node $v$ visited
      Let $l'$ be the length of rope remaining
      Call bDFX($v, l'$)
      Let $E'$ be the set of new explored edges
      Let $V'$ be the set of nodes in $E'$
      Let $\mathcal{T}'$ be a spanning forest of $G' = (V', E')$
      $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
  MOVE ROBOT BACK from $s_i$ to $s$ retracing old path
```

FIG. 5. The Closest First Exploration (CFX) algorithm with start node s and rope length $(1 + \alpha)r$.

to $\Theta(|V|)$. We need to prove the minimum-size criterion for both the incomplete trees chosen and those incomplete trees that are newly created each iteration.

Our proofs begin with several smaller lemmas and build up to the main theorem of this section. Recall that $\Delta_T(s_i, T)$ is the maximum distance along T from s_i to all nodes in T and consequently is the radius of T. We begin with a lemma that bounds the size of the new incomplete trees formed during the `explore` process in terms of the rope length, the distance from the start node to the root of the explored tree, and the radius of the explored tree:

LEMMA 3.1. *Let $\mathcal{T}'$ be the collection of trees formed in any run of* `explore` *on a tree T rooted at node s_i with rope length l. For any tree $T' \in \mathcal{T}'$, if T' is incomplete, then $|T'| > l - d_{G^*}(s, s_i) - \Delta_T(s_i, T)$. Furthermore, if $l - d_{G^*}(s, s_i) - \Delta_T(s_i, T) > 0$, then the initial tree T becomes complete.*

PROOF. Following the procedure `explore`, let T' be an incomplete tree created after running `bDFX` from node v. We can bound the length of rope remaining when the robot is at node v by the current path taken from s as follows:

$$l' = l - d_{G^*}(s, s_i) - d_T(s_i, v) \geq l - d_{G^*}(s, s_i) - \Delta_T(s_i, T).$$

Let $w \in V(T')$ be an incomplete node. After the running of `bDFX` with a rope length l', in order for w to remain incomplete, the remaining length at node w must have been 0; see the code in Figure 2. Therefore, there must exist a node disjoint path of new explored edges from v to w whose length is l'. Since the number of nodes in T' is at least one more than the length of this path, we see that $|T'| > l'$.

For the second part of the lemma, assume that after termination T remains incomplete. Let v be any incomplete node in T. During the depth first traversal of T (lines 27–34), node v must have been visited. Since l' is the length of rope remaining, we know that $l' = l - d_{G^*}(s, s_1) - d_T(s_i, v) > 0$. After calling the function `bDFX` with a positive rope length, the initial node v can only be marked *explored*. This contradicts the assumption that v was incomplete after termination. Therefore, T cannot be incomplete. □

The following lemma further bounds the size of the trees created by showing that the distance from the start node to the root of the original tree is no more than the radius of the graph, r.

LEMMA 3.2. *If $G^* \neq G$, then the closest incomplete node v to s in G^* has $d_{G^*}(s, v) \leq r$.*

PROOF. Assume not. Let v be the closest incomplete node to s. Then, $d_{G^*}(s, v) > r$. Let $P = P_G(s, v)$ be the true shortest path from s to v. Notice that $|P| \leq r$ by the definition of the radius. Let $e = (u, w)$ be the first edge on P not in G^*. This must exist otherwise $d_{G^*}(s, v) = |P| \leq r$. This implies that node u is also incomplete in G^*. However, $d_{G^*}(s, u) = d_G(s, u) \leq r < d_{G^*}(s, v)$ contradicts the fact that v is the closest incomplete node. □

The final step in bounding the size of the trees created during the `explore` process requires bounding the radius of the tree T to be explored. However, as mentioned in the description of our algorithm, to guarantee this we must first prune T. In addition, we must be careful that the pruned subtrees do not themselves become too small. The following lemma shows that the pruning process preserves a minimum size of `minD` for T, a minimum size of `maxD` $-$ `minD` for all pruned subtrees, and a maximum radius of `maxD` for T.

LEMMA 3.3. *After pruning a tree T rooted at v, the maximum depth of T is less than or equal to* `maxD`. *In other words, $\Delta_T(v, T) \leq$* `maxD`. *If initially $|T| \geq$* `minD`, *then after pruning, $|T| \geq$* `minD`. *For any subtree T_w created after pruning, $|T_w| >$* `maxD` $-$ `minD`.

PROOF. The proof follows directly from the construction of the `prune` algorithm. □

In order to argue for the correctness of the `CFX` algorithm, we also need to guarantee that all incomplete nodes eventually be visited. The following lemma shows that in each iteration all current incomplete nodes belong to some current incomplete tree.

LEMMA 3.4. *After every iteration of the main loop in* CFX (*lines* 3–14)*, any incomplete node in G^* belongs to some tree $T \in \mathcal{T}$.*

PROOF. Nodes are marked *incomplete* only in the bDFX function. After each call, these nodes, by definition, belong to the graph G' and the corresponding spanning forest $\mathcal{T}'$ which is added to $\mathcal{T}$ during explore (line 34). Let us look at the other possible changes to $\mathcal{T}$. The prune function simply breaks trees up, not removing any nodes. After each iteration in the main loop of CFX, only complete trees are removed. The final step in the main loop (line 13) simply merges trees, also not eliminating any nodes. Since initially the only incomplete node, s, belongs in $\mathcal{T}$, after each iteration, every incomplete node in G^* must still lie in some tree in $\mathcal{T}$. □

We now have the tools necessary to prove that the minimum-size criterion is always maintained.

LEMMA 3.5. *After every iteration of the main loop in* CFX (*lines* 3–14)*, for every $T \in \mathcal{T}$, $|T| \geq \alpha r/4$. In addition, after each call of* explore *in line* 11*, the tree T that is explored becomes complete and is removed from $\mathcal{T}$.*

PROOF. After the initial iteration of the loop, if the graph is complete, then there are no incomplete trees and $\mathcal{T}$ is empty. If instead the graph is incomplete, $\mathcal{T}$ contains only one tree T (since no pruning is done) and, by Lemma 3.1, $|T| \geq (1+\alpha)r > \alpha r/4$.

During every successive iteration, we choose the tree $T_i \in \mathcal{T}$ closest to s, with s_i being the node in T_i closest to s. Let T be the resulting pruned tree of T_i containing s_i. Since $\texttt{minD} = \alpha r/4$, Lemma 3.3 shows that T still maintains the minimum-size criterion. Furthermore, Lemma 3.3 also shows that for each pruned subtree T_w, $|T_w| > \texttt{maxD} - \texttt{minD} = \alpha r/4$. Since all explored trees are removed in line 12 and the merging step (line 13) only increases the size of trees, we only need to show that, for every incomplete tree T' created in explore (line 11), $|T'| \geq \alpha r/4$.

From Lemma 3.2, let v^* be the closest incomplete node. From Lemma 3.4, we know that v^* must belong to some tree in $\mathcal{T}$. Since T_i is the tree in $\mathcal{T}$ closest to s and s_i is the closest node, we know that s_i is as close to s as *any* other node in any tree of $\mathcal{T}$. Therefore, we see that $d_{G^*}(s, s_i) \leq d_{G^*}(s, v^*) \leq r$. Lemma 3.3 states that after pruning $\Delta_T(s_i, T) \leq \texttt{maxD} = \alpha r/2$. Let v be any node in (the pruned) T that was incomplete prior to calling explore on T, and let T' be the newly explored tree discovered by calling the bDFX function (line 30) on v. From lines 26–29 of the explore procedure, we know that $l' = l - d_{G^*}(s, s_i) - d_T(s_i, v) \geq (1+\alpha)r - r - \texttt{maxD} = \alpha r/2 > 0$. If T' is incomplete, from Lemma 3.1, we see that $|T'| > l' \geq \alpha r/2$.

Therefore, after each iteration of the main loop of CFX we see that for any tree $T \in \mathcal{T}$, $|T| \geq \alpha r/4$. In addition from Lemma 3.1, we know that, after the explore function on a tree T terminates, T becomes complete and is removed from $\mathcal{T}$ (line 12). □

THEOREM 3.6. *Any unknown graph $G = (V, E)$ with source node s and (given) radius r can be explored by a tethered robot using a maximum rope length of $(1+\alpha)r$ with cost $\Theta(|E| + |V|/\alpha)$, for $0 < \alpha < 1$.*

PROOF. Since all steps taken by the robot are performed in the explore and bDFX functions, we can analyze the cost of the algorithm by analyzing these two procedures. Let us again examine the pruned tree T given to the `explore` function. For each pruned tree explored, we group the number of exploration steps performed by the robot into three parts, $S_1(T)$, $S_2(T)$, $S_3(T)$. Let $S_1(T) = 2d_{G^*}(s, s_i)$ be the number of steps taken by lines 26 and 35 in the algorithm, which performs a roundtrip from s to s_i. Let $S_2(T) = 2|T|$ be the number of steps taken in line 27 to perform a depth first traversal of T. Let $S_3(T)$ be the number of steps taken by all calls to `bDFX` (line 30) for one call of `explore`, which is equal to twice the number of unexplored edges traversed, those marked explored.

In the initial phase, $T = \{s\}$, so $S_1(T) = S_2(T) = 0$. Otherwise, from Lemma 3.5, we know that $|T| \geq \alpha r/4$. In addition, as in the proof of Lemma 3.5, we know that $d_{G^*}(s, s_i) \leq r$. Therefore, we can bound $S_1(T) = 2d_{G^*}(s, s_i) \leq 2r \leq 8|T|/\alpha$. Consequently, the cost of the `explore` function on a given pruned tree T is $S_1(T) + S_2(T) + S_3(T) \leq (2 + 8/\alpha)|T| + S_3(T)$. Let $\mathcal{T}_e$ be the set of all trees explored in the entire algorithm. Since `bDFX` only traverses unexplored edges, we immediately see that $\sum_{T\in\mathcal{T}_e} S_3(T) = 2|E|$.

We now need to bound the size of all trees explored. After any iteration of the main loop (lines 3–14), all trees in $\mathcal{T}$ are node disjoint because of the merge step taken in line 13. One would then assume that the size of all trees is bounded by $|V|$. However, there is a little overlap between successive iterations. Let T be the current tree explored and $v \in T$ be a node in the tree. From Lemma 3.5, we know that after the `explore` function call of line 11, T is complete. As a result, if v was incomplete prior to the `explore` function call, then it becomes complete after but may be part of a new tree in $\mathcal{T}$. If v was complete prior to the call, then after the call it can no longer be present in any other tree in $\mathcal{T}$; all trees are node disjoint and v could not be added in subsequent explorations since the trees are formed by newly explored edges. Consequently, any node v can be part of at most *two* trees, once while incomplete and once while complete. We can then bound the size of the explored trees by $\sum_{T\in\mathcal{T}_e} |T| \leq 2|V|$. Therefore, the total number of exploration steps $\mathcal{S}$ taken by the `CFX` algorithm is

$$\begin{aligned}\mathcal{S} &= \textstyle\sum_{T\in\mathcal{T}_e}(S_1(T) + S_2(T) + S_3(T))\\ &\leq 2|E| + \textstyle\sum_{T\in\mathcal{T}_e}(2 + 8/\alpha)|T|\\ &\leq 2|E| + (2 + 8/\alpha)2|V|.\end{aligned}$$

By Lemma 3.4, any incomplete node in G^* must belong to some tree $T \in \mathcal{T}$. Since the algorithm terminates when $\mathcal{T} = \emptyset$, we see that upon termination there can be no more incomplete nodes in G^*. Therefore, $G^* = G$ and the graph is fully explored with a cost of $\Theta(|E| + |V|/\alpha)$. □

4. *Simple Extension for an Unknown Radius*

We now assume that the radius is not known but is discovered by the robot. Let us first look at a simple solution by starting with an initial constant value guess, r', of the radius. We search the graph as much as possible using a rope of length $(1+\alpha)r'$ and if this length is insufficient enough to explore the whole graph, we repeat with a rope double the length. Under this situation, since any radius $r' > r$ suffices, we obtain a cost of $O(|E| \log r)$. Observing that we can use prior information to avoid re-exploring edges yields a cost of $O(|E| + |V| \log r)$.

We can, however, do much better and surprisingly we only need to modify the CFX algorithm (Figure 5) slightly, in two places. The changes involve the two function calls in the main loop. First, we prune the current tree to be explored to a size relative to its distance to the start node rather than to the (unknown) radius. Second, we pass the explore function a rope length relative to this distance rather than the (unknown) radius. Essentially, we make the following two alterations:

```
6       prune($T_i, s_i, \alpha d_{G^*}(s, s_i)/4, 9\alpha d_{G^*}(s, s_i)/16$)
11      explore($T, s_i, (1+\alpha) d_{G^*}(s, s_i)$)
```

As an additional caveat, notice in line 11 if $d_{G^*}(s, s_i) = 0$, the function is given a rope length of size 0 and is, therefore, unable to explore any edges. So, we actually replace $d_{G^*}(s, s_i)$ with $\max(d_{G^*}(s, s_i), c)$ for some constant $c > 0$. For the remainder of this section, we let algorithm CFX refer to this new modified version. The essentials behind the analysis of this algorithm are obviously nearly identical to the original algorithm. Again, the key is to maintain the *minimum-size criterion*. However, in this case, we desire a more relaxed constraint that, for any tree $T \in \mathcal{T}$, $|T| \geq \alpha d_{G^*}(s, T)/4$.

The resulting algorithm essentially uses a radius, $d_{G^*}(s, s_i)$, that is always guaranteed to be no larger than the true radius. Since our modification is so slight, all previous lemmas still hold for this algorithm except for Lemma 3.5 proving the minimum-size criterion.

LEMMA 4.1. *After every iteration of the main loop in* (*the modified version of*) CFX (*lines* 3–14)*, for every* $T \in \mathcal{T}$, $|T| \geq \max(d_{G^*}(s, T), c)\alpha/4$. *In addition, after each call of* explore *in line* 11, *the tree* T *that is explored becomes complete and is removed from* $\mathcal{T}$.

PROOF. We follow the same lines as the proof of Lemma 3.5. Again, we can see that if the initial tree constructed is incomplete, then the bound holds.

Let us now examine each successive iteration, where we choose the closest tree $T_i \in \mathcal{T}$, with closest node s_i. For simplicity, we shall assume that $d_{G^*}(s, T_i) > c$. Let T be the resulting pruned tree of T_i containing s_i, and let T_w be any other pruned subtree of T_i. By Lemma 3.3, we know that $|T| \geq \texttt{minD} = \alpha d_{G^*}(s, s_i)/4 = \alpha d_{G^*}(s, T)/4$ and that $|T_w| > \texttt{maxD} - \texttt{minD} = 5\alpha d_{G^*}(s, s_i)/16$. To see that the criterion still holds for T_w notice that

$$\begin{aligned} d_{G^*}(s, T_w) &\leq d_{G^*}(s, s_i) + d_{G^*}(s_i, w) \\ &= d_{G^*}(s, s_i)(1 + \alpha/4) && (\text{Since } d_{G^*}(s_i, w) = \alpha d_{G^*}(s, s_i)/4) \\ &< 5 d_{G^*}(s, s_i)/4. && (\text{Since } \alpha < 1) \end{aligned}$$

Substituting the above equation, we see that $|T_w| > \alpha d_{G^*}(s, T_w)/4$.

Again, we now only need to show that every incomplete tree created in the modified call to explore (line 11) also maintains the minimum-size criterion. From Lemma 3.3, we know that, after pruning T, $\Delta_T(s_i, T) \leq \texttt{maxD} = 9\alpha d_{G^*}(s, s_i)/16$. Let v be any node in (the pruned) T that was incomplete prior to calling explore on T and let T' be the newly explored tree discovered by the bDFX function (line 30) on v. Since $v \in T$ and $v \in T'$, we can see that

$$\begin{aligned} d_{G^*}(s, T') &\leq d_{G^*}(s, s_i) + d_{G^*}(s_i, v) \leq d_{G^*}(s, s_i) + \Delta_T(s_i, T) \\ &\leq d_{G^*}(s, s_i) + 9\alpha d_{G^*}(s, s_i)/16 \end{aligned}$$

$$\begin{aligned} &< 25d_{G^*}(s, s_i)/16, \text{ or} \\ d_{G^*}(s, s_i) &> 16d_{G^*}(s, T')/25. \end{aligned}$$

If T' is incomplete, from Lemma 3.1, we see that

$$\begin{aligned} |T'| &> l - d_{G^*}(s, s_i) - \Delta_T(s_i, T) \\ &\geq (1+\alpha)d_{G^*}(s, s_i) - d_{G^*}(s, s_i) - 9\alpha d_{G^*}(s, s_i)/16 \\ &\geq 7\alpha d_{G^*}(s, s_i)/16 \\ &> 7\alpha d_{G^*}(s, T')/25 \\ &> \alpha d_{G^*}(s, T')/4. \end{aligned}$$

Therefore, after each iteration of the main loop of CFX we see that for any tree $T \in \mathcal{T}$, $|T| \geq \alpha d_{G^*}(s, T)/4$. In addition, as with Lemma 3.5, we know that, after the `explore` function on a tree T terminates, T becomes complete and is removed from $\mathcal{T}$ (line 12). $\square$

THEOREM 4.2. *Any unknown graph $G = (V, E)$ with source node s and unknown radius r can be explored by a tethered robot using a maximum rope length of* $(1+\alpha)r$ *with cost* $\Theta(|E| + |V|/\alpha)$, *for* $0 < \alpha < 1$.

PROOF. As with Theorem 3.6, we break the robot traversal steps on a tree T into three groups, $S_1(T)$, $S_2(T)$, $S_3(T)$, defined as before. Using Lemma 4.1, we can bound $S_1(T) = 2d_{G^*}(s, s_i) \leq 8|T|/\alpha$. Recall that $S_2(T) = 2|T|$ and that $\sum_{T\in\mathcal{T}_e} S_3(T) = 2|E|$, where $\mathcal{T}_e$ is the set of all trees explored.

We can bound the total exploration steps $\mathcal{S}$ taken by the modified CFX algorithm as

$$\begin{aligned} \mathcal{S} &= \textstyle\sum_{T\in\mathcal{T}_e}(S_1(T) + S_2(T) + S_3(T)) \\ &\leq 2|E| + (2 + 8/\alpha)\textstyle\sum_{T\in\mathcal{T}_e}|T| \\ &\leq 2|E| + (2+8/\alpha)2|V| \text{ which is } \Theta(|E| + |V|/\alpha). \end{aligned}$$

By Lemma 3.4, we know that once the algorithm terminates it has fully explored the graph. We now only need to show that the length of the rope used is no more than $(1+\alpha)r$. For every iteration on a tree T_i with closest node s_i, we use a rope of length $(1+\alpha)d_{G^*}(s, s_i)$. By Lemma 3.2, we know that since $G^* \neq G$ there exists an incomplete node v such that $d_{G^*}(s, v) \leq r$. From Lemma 3.4, v belongs to some tree in $\mathcal{T}$. Recall that s_i is chosen to be the closest node among all nodes in all trees in $\mathcal{T}$. Therefore, $(1+\alpha)d_{G^*}(s, s_i) \leq (1+\alpha)d_{G^*}(s, v) \leq (1+\alpha)r$. $\square$

Using the simple reduction as described in Lemma 2.1, we can also achieve similar bounds for the piecemeal search when the radius is unknown.

COROLLARY 4.3. *Any unknown graph $G = (V, E)$ with source node s and unknown radius r can be explored by a fuel-constrained robot using a tank of size at most* $2(1+\beta)r$ *with cost* $\Theta(|E|/\beta)$, *for* $0 < \beta < 1$.

4.1. TREASURE HUNTING. In the treasure hunting problem or shortest path problem, recall that we are searching for a node t at unknown distance from s in an infinite graph G^∞. We define the graph $G^k \subset G^\infty$ to be the subgraph induced by all nodes v such that $d_{G^\infty}(s, v) \leq k$. If we let $r = d_{G^\infty}(s, t)$, in general graphs

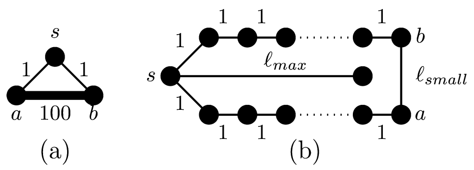

FIG. 6. (a) A simple example illustrating the need to extend the minimum rope length needed. Here the rope length needed is 101 and not just 1 or 2. (b) A slightly more complex example where the upper and lower chains have $\ell_{\max}$ nodes. Here the rope length needed $d(s, a) + \ell_{small} = \ell_{\max} + \ell_{small}$ and not simply $d(s, a) = \ell_{\max}$ or $d(s, a) + \ell_{\max} = 2\ell_{\max}$.

the treasure hunting problem must explore all nodes and edges in G^r. To see this, we note that the adversary only needs to make t be the last node in G^r visited by any exploration. If we modify algorithm CFX to stop once the node t is discovered, we know that the robot can never go beyond distance $(1 + \alpha)d_{G^*}(s, t) \leq (1 + \alpha)r$ from s. This observation yields the following corollary.

COROLLARY 4.4. *For any unknown infinite graph G^∞ with source node s, we can find a destination node t by exploring the subgraph $G^{(1+\alpha)d(s,t)} = (V, E)$ with a cost of* $\Theta(|E| + |V|/\alpha)$, *for* $0 < \alpha < 1$.

5. *Extension to weighted graphs*

Let us now consider a variation of the problem with graphs containing weighted edges, or edges with different lengths. This variation requires a slight modification to our definitions.

5.1. SOME MODIFIED DEFINITIONS. For our graph $G = (V, E)$, we associate a length $\ell(e) = \ell(u, v) \geq 0$ for every $e = (u, v) \in E$. The cost for a robot to traverse an edge is $\ell(e)$. We also assume that a robot can detect the length of an edge, once the edge is first encountered but before actually traversing it.

We must now redefine our distance and size calculations in terms of these weighted edge lengths. For a set of edges E, the size of the set is the sum of edge lengths, $|E| = \sum_{e \in E} \ell(e)$. For any (sub)graph $H = (V(H), E(H))$, the size of the graph is $|H| = |E(H)|$. For the remainder of this section, when we refer to size and distance terms we are referring to their weighted values. For example, the weighted distance between two nodes is $d_G(v, w) = |P_G(v, w)| = \sum_{e \in P_G(v,w)} \ell(e)$. To help in our analysis later, let us define the *extended edge set* for a subgraph $H \subseteq G$ as $EE(H) = \{e = (u, v) \in E(G) | u \in V(H)\}$. That is, the extended edge set of H is the set of all edges in G that have at least one endpoint in $V(H)$. Similarly, we define the *extended size* of H as $\|H\| = \sum_{e \in EE(H)} \ell(e)$.

In the previous unweighted solution, it was sufficient to have a rope of length $r + 1$ to visit the entire graph. We note that every edge as well as every vertex *must* be explored in a graph of unknown radius, otherwise it would be impossible to determine if two unexplored edges were in fact the same edge or led to a new unexplored node. As is illustrated in Figure 6(a), a length of $r + 1$ would be insufficient to explore the graph. It would certainly be sufficient to include an additional rope length equal to the length of the longest edge $\ell_{\max}$. However, as Figure 6(b) illustrates, this would also not be as tight as we could use. Instead, we prefer to use a length proportional to the smallest rope required to explore the graph completely, including all edges, if the graph were known in advance. This is equal to the distance to the deepest edge. That is, let the distance from a node w to an edge $e = (u, v)$ be $d(w, e) = \min(d(w, u), d(w, v)) + \ell(e)$, and let the *deepest edge* be the edge $e_\Delta \in E$ such that $d_G(s, e_\Delta) = \max_{e \in E(G)} d_G(s, e)$. Therefore, the minimum rope length needed to explore a graph is $r_\Delta = d_G(s, e_\Delta)$. Observe that $r_\Delta \geq r$. We prove that our algorithm can explore the graph using a rope of length $r_\Delta + \alpha r$. Of course, for the fuel-constrained, piecemeal, search the same reasoning applies regarding the size of the fuel tank needed, and again the two problems can be shown to be equivalent within constant factors. For completeness, we give a proof below.

LEMMA 5.1. *Assume we are given an unknown, weighted graph G. For any $\beta > 2$, the fuel-constrained exploration on G with a tank of size $\beta\ell$ can be reduced to the tethered exploration problem with a rope of length ℓ; the cost increases by a factor of $\frac{\beta}{\beta-2}$. Conversely, for any $\alpha > 1$, the tethered exploration on G with a rope of length $\alpha t/2$ can be reduced to the fuel-constrained exploration problem with a tank of size t, with a $\frac{2\alpha}{\alpha-1}$ factor increase in cost.*

PROOF. We first describe the reduction from the fuel-constrained search to a tethered search. As with Lemma 2.1, let A be an algorithm that performs a tethered robot exploration with a rope of length ℓ, with cost T. We shall develop an algorithm B for fuel-constrained exploration with a tank of size $\beta\ell$, with $\beta > 2$ that emulates A. Unlike the previous lemma, a fueled robot cannot arbitrarily interrupt the emulation for refueling because the robot may be in the midst of traversing an edge, although technically we could define the problem to allow this. However, let us initially allow this interruptibility. Therefore, let algorithm B emulate A with the addition that, every $(\beta - 2)\ell$ steps, it sends the robot to s for refueling and back to the current edge portion. For example, if the edge has length 3 and the robot is able to explore 1 unit of the edge before refueling, the robot would refuel and return to continue with the remaining 2 units of the edge. Since the robot is always within distance ℓ to s, the amount of fuel needed to go to s and back is at most 2ℓ and the total fuel used between refuelings is at most $2\ell + (\beta - 2)\ell = \beta\ell$. Since there is a refueling operation with cost at most 2ℓ after at least $(\beta - 2)\ell$ emulated steps of A, algorithm B has cost at most $T + T(2\ell)/(\beta - 2)\ell = \beta T/(\beta - 2)$. For the case when the robot cannot temporarily interrupt an edge traversal, we just terminate each phase before starting to explore an edge partially. This only shortens the cost of the algorithm.

The converse reduction also follows the same lines as the proof in Lemma 2.1. Again, let B be an algorithm that performs a fuel-constrained exploration with a tank of size t with cost T, and let A emulate B with a tethered robot having a rope of length $\alpha t/2$. When A uses $\alpha t/2$ units of rope, we unwind and return. Since the fuel-constrained robot in B must never go beyond a known distance $t/2$ from the start node, the unwinding step can take at most $\alpha t/2 + t/2$ steps and is executed at most every $\alpha t/2 - t/2$ steps. Therefore, the total cost of the emulation is at most $T + T(\alpha + 1)/(\alpha - 1) = T(2\alpha)/(\alpha - 1)$. $\square$

5.2. MODIFIED ALGORITHM. In this section, we assume that the radius is unknown but we guarantee that the rope length used is never larger than $r_\Delta + \alpha r$, with r_Δ being the distance to the deepest edge.

```
$\ell$-CFX$(s, r, \alpha)$
  Initially $\mathcal{T} = \{s\}$
  Repeat
    Let $T_i \in \mathcal{T}$ be the closest tree to $s$ in $G^*$
    Let $s_i$ be the closest node in $T_i$ to $s$
    $\ell$-prune$(T_i, s_i, \alpha d_{G^*}(s, s_i)/5, 3\alpha d_{G^*}(s, s_i)/5)$
      producing $\mathcal{T}_i = \{T_{i_0}, T_{i_1}, \ldots T_{i_k}\}$
    // Here $T_i = \cup_{T \in \mathcal{T}_i} T$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \{T_i\} \cup \mathcal{T}_i$
    Let $T \in \mathcal{T}_i$ such that $s_i \in T$
    // Need to add the length of longest unexplored extended edge in $T$.
    Let $\ell' = \max_{e \in EE(T) | e \text{is unexplored}} \ell(e)$
    $\ell$-explore$(T, s_i, (1 + \alpha) d_{G^*}(s, s_i) + \ell')$
    remove all complete trees from $\mathcal{T}$
    merge any trees in $\mathcal{T}$ with common nodes
  Until $\mathcal{T} = \emptyset$

$\ell$-bDFX$(v, l)$
  if $v$ is tagged
    return
  Mark $v$ as tagged
  For each unexplored edge $e = (v, w) \in E$ with $\ell(e) \leq l$
    Move robot to $w$ along edge $(v, w)$
    Mark edge $(v, w)$ as explored
    Call $\ell$-bDFX$(w, l - \ell(e))$
    Move robot back from $w$ to $v$ along edge $(v, w)$
  if $v$ has any unexplored edges remaining
    Mark $v$ as incomplete
  else
    Mark $v$ as explored

$\ell$-prune$(T, v, \mathtt{minD}, \mathtt{maxD})$
  If $T = \{s\}$, return
  Root $T$ at $v$
  $\mathcal{T}_i \leftarrow \{T\}$
  for each edge $e = (u, w) \in T$ such that $d_T(v, u) < \mathtt{minD}$ and $d_T(v, w) \geq \mathtt{minD}$
    Let $T_w \subseteq T$ be the subtree of $T$ rooted at $w$
    if $\Delta_T(v, T_w) > \mathtt{maxD}$
      // Separate subtree $T_w$ from $T$
      $T \leftarrow T \setminus T_w$
      $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{T_w\}$
```

FIG. 7. The ℓ-`CFX`, ℓ-`bDFX`, and ℓ-`prune` algorithms. Note that ℓ-`explore` is nearly identical to the original `explore` and so is not shown.

Our approach is to modify the various algorithms to properly handle edge lengths in the following manner; see Figure 7:

(1) The ℓ-`bDFX` algorithm incorporates the following natural extensions to the `bDFX` algorithm:
   (a) On traversing an edge e, the length of the remaining rope decreases by $\ell(e)$, instead of 1.

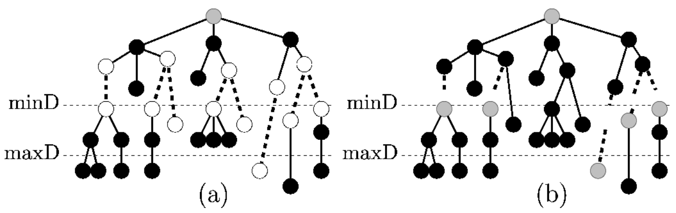

FIG. 8. A simple example of pruning a weighted tree with the indicated `minD` and `maxD` values. Here we illustrate the weight of an edge by its vertical displacement. (a) The initial tree with all edges crossing distance `minD` dashed. The two nodes involved per edge are highlighted. (b) The resulting trees formed after pruning. The edges pruned are left "dangling" from each node. For analysis purposes, these edge portions contribute to the (extended) size of their respective trees.

(b) The robot can only traverse an edge e if the remaining rope length l is not smaller than the edge length $\ell(e)$. This implies that we may backtrack even if there is rope remaining but no edge short enough to explore.

(2) We modify the ℓ-`prune` algorithm to prune only at nodes whose parent edge overlaps the required depth `minD`; see Figure 8.

(3) The ℓ-`CFX` algorithm is changed to make the proper modified algorithm calls. In addition, it sends slightly different values for `minD` and `maxD` to the ℓ-`prune` method and a slightly different rope length to ℓ-`explore`. This length is the original length plus a value ℓ', which is the length of the longest unexplored (extended) edge in $EE(T)$. As we show later, this length is still within the stated rope length bounds.

(4) The ℓ-`explore` algorithm is simply a modification of `explore` that calls ℓ-`bDFX` and so is not shown.

5.3. MODIFIED ANALYSIS. The analysis of the weighted case closely mirrors the unweighted case presented in Sections 3 and 4. However, as mentioned in Section 5.1, much of our analysis must now focus on the sum of edge lengths rather than the number of vertices. In addition, to ease our analysis, we consider the extended size of a tree when accounting for traversals on these trees. Our algorithm requires modifications to Lemmas 3.1, 3.3, and 4.1 and to Theorem 4.2. However, we note that Lemmas 3.2 and 3.4 and their proofs still hold.

LEMMA 5.2. *Let $\mathcal{T}'$ be the collection of trees formed in any run of ℓ-`explore` on a tree T rooted at node s_i with rope length l. For any tree $T' \in \mathcal{T}'$, if T' is incomplete, then $\|T'\| > l - d_{G^*}(s, s_i) - \Delta_T(s_i, T)$. Furthermore, if $l - d_{G^*}(s, s_i) - \Delta_T(s_i, T) \geq \ell'$, then the initial tree T becomes complete, with ℓ' being the length of the longest unexplored edge in $EE(T)$.*

PROOF. As in the proof of Lemma 3.1, we consider an incomplete tree T' created after running ℓ-`bDFX` from some node v during a run of ℓ-`explore`. As before, the length of the rope remaining when ℓ-`explore` starts from v is $l' = l - d_{G^*}(s, s_i) - d_T(s_i, v) \geq l - d_{G^*}(s, s_i) - \Delta_T(s_i, T)$. Let $w \in V(T')$ be an incomplete node. Let l'' be the length of rope remaining when ℓ-`bDFX` started at w. Since w is not explored there must have been an unexplored edge e incident on w such that $\ell(e) > l''$. This implies that there must exist a node disjoint path of new explored edges from v to w whose edge length is more than $l' - \ell(e)$. Since the extended size, $\|T'\|$, includes both these edges and the extended edge e, we see that $\|T'\| > l'$.

For the second part of the lemma, assume that after termination T remains incomplete. Let v be any node in T that remains incomplete after termination of ℓ-`explore`. During the depth first traversal of T, node v must have been visited. As in the first part and from the condition on the second part, we know that the length of rope remaining satisfies $l' \geq l - d_{G^*}(s, s_i) - \Delta_T(s_i, T) \geq \ell'$. Since v is incomplete after termination of ℓ-`explore`, it must also be incomplete after termination of the ℓ-`bDFX` function. This implies that there is an unexplored edge e incident to v such that $\ell(e) > l' \geq \ell'$. However, e is an unexplored edge in $EE(T)$. This contradicts the definition of ℓ', that $\ell(e) \leq \ell'$, and therefore T must be complete. □

LEMMA 5.3. *After pruning a tree T rooted at v, the maximum depth of T is less than or equal to* `maxD`. *In other words, $\Delta_T(v, T) \leq$* `maxD`. *If initially $\|T\| \geq$* `minD`, *then after pruning, $\|T\| \geq$* `minD`. *For any pruned subtree T_w created after pruning, $\|T_w\| > \max\{\mathtt{maxD}, d_T(v, w)\} - \mathtt{minD}$.*

PROOF. The proof for the first part follows directly from the construction of the ℓ-`prune` algorithm. For the second part, suppose the subtree T_w rooted at node w is pruned off. Then, $d_T(v, w) \geq \mathtt{minD}$. Let u be the parent of w. Since u remains in T, from our definition of the extended size, we know that after pruning $\|T\| \geq d_T(v, u) + \ell(u, w) = d_T(v, w) \geq \mathtt{minD}$.

For the final part, let T_w be any pruned subtree of T and let u be the parent of w. Recall that from the selection condition in ℓ-`prune`, $d_T(v, u) < \mathtt{minD}$ and $d_T(v, w) \geq \mathtt{minD}$. We look at two possible cases.

Case 1. Let $d_T(v, w) \leq \mathtt{maxD}$. From the condition in ℓ-`prune` we know that $\Delta_T(v, T_w) > \mathtt{maxD}$. Since the edge $(u, w) \notin E(T_w)$ and $w \in T_w$, we know that

$$\begin{aligned} \|T_w\| &\geq \ell(u, w) + \Delta_{T_w}(w, T_w) \\ &= \Delta_T(u, T_w) \\ &= \Delta_T(v, T_w) - d_T(v, u) \\ &> \mathtt{maxD} - \mathtt{minD}. \end{aligned}$$

Case 2. Let $d_T(v, w) > \mathtt{maxD}$. Since $d_T(v, u) < \mathtt{minD}$ and $(u, w) \in EE(T_w)$, we know that $\|T_w\| \geq \ell(u, w) = d_T(v, w) - d_T(v, u) > d_T(v, w) - \mathtt{minD}$.

Therefore, we know that $\|T_w\| > \max\{\mathtt{maxD}, d_T(v, w)\} - \mathtt{minD}$. □

As with the previous two `CFX` algorithms, the key to the analysis is to maintain a *minimum-size criterion* for each tree that is explored. In this case, we desire that, for any tree $T \in \mathcal{T}$, $\|T\| \geq \alpha d_{G^*}(s, T)/5$. That is, we consider the *extended* size of the tree and use a factor of one-fifth instead of the previous one-quarter.

LEMMA 5.4. *After every iteration of the main loop in ℓ-*`CFX` *(lines 3–16), for every $T \in \mathcal{T}$, $\|T\| \geq \alpha d_{G^*}(s, T)/5$. In addition, after each call of ℓ-*`explore` *in line 13, the tree T that is explored becomes complete and is removed from $\mathcal{T}$.*

PROOF. Clearly, this holds for the initial tree created as $d_{G^*}(s, T) = 0$. For each successive iteration, we must again examine the operations on the tree $T_i \in \mathcal{T}$ closest to s, with closester node s_i. Let T be the resulting pruned tree of T_i and let T_w be any other pruned subtree of T_i. By Lemma 5.3, we know that $\|T\| \geq \mathtt{minD} =$

$\alpha d_{G^*}(s, s_i)/5 = \alpha d_{G^*}(s, T)/5$ and that $\|T_w\| > \max\{d_T(s_i, w), \mathtt{maxD}\} - \mathtt{minD}$. To see that the criterion still holds for T_w, we look at the two cases. Recall that $\mathtt{minD} = \alpha d_{G^*}(s, s_i)/5$ and $\mathtt{maxD} = 3\alpha d_{G^*}(s, s_i)/5$.

Case 1. If $d_T(s_i, w) \le \mathtt{maxD}$, then $\|T_w\| > \mathtt{maxD} - \mathtt{minD} = 2\alpha d_{G^*}(s, s_i)/5$.

$$\begin{aligned} d_{G^*}(s, T_w) &\le d_{G^*}(s, s_i) + d_T(s_i, w) \\ &\le d_{G^*}(s, s_i) + \mathtt{maxD} \\ &= d_{G^*}(s, s_i)(1 + 3\alpha/5) \\ &< 8d_{G^*}(s, s_i)/5. \qquad \text{(Since } \alpha < 1\text{)} \end{aligned}$$

Substituting the above equation yields

$$\|T_w\| > 2\alpha d_{G^*}(s, s_i)/5 > 2\alpha d_{G^*}(s, T_w)/8 > \alpha d_{G^*}(s, T_w)/5.$$

Case 2. If $d_T(s_i, w) > \mathtt{maxD} = 3\alpha d_{G^*}(s, s_i)/5$, then

$$\begin{aligned} \|T_w\| &> d_T(s_i, w) - \mathtt{minD} \\ &= d_T(s_i, w) - \alpha d_{G^*}(s, s_i)/5 \\ &= d_T(s_i, w)/5 + 4d_T(s_i, w)/5 - \alpha d_{G^*}(s, s_i)/5 \\ &> \alpha d_T(s_i, w)/5 + 4d_T(s_i, w)/5 - \alpha d_{G^*}(s, s_i)/5 \qquad \text{(Since } \alpha < 1\text{)} \\ &> \alpha d_T(s_i, w)/5 + 12\alpha d_{G^*}(s, s_i)/25 - \alpha d_{G^*}(s, s_i)/5 \\ &> \alpha d_T(s_i, w)/5 + \alpha d_{G^*}(s, s_i)/5 \\ &\ge \alpha d_{G^*}(s, T_w)/5. \qquad \text{(Since } d_{G^*}(s, T_w) \le d_{G^*}(s, s_i) + d_T(s_i, w)\text{)} \end{aligned}$$

Again, we now only need to show that every incomplete tree created in ℓ-`explore` (line 13) also maintains the minimum-size criterion. From Lemma 5.3, we know that, after pruning T, $\Delta_T(s_i, T) \le \mathtt{maxD} = 3\alpha d_{G^*}(s, s_i)/5$. Let v be any node in (the pruned) T that was incomplete prior to calling ℓ-`explore` on T and let T' be the newly explored tree discovered by the ℓ-`bDFX` function on v. Since $v \in T$ and $v \in T'$, we can see that

$$\begin{aligned} d_{G^*}(s, T') &\le d_{G^*}(s, s_i) + d_{G^*}(s_i, v) \le d_{G^*}(s, s_i) + \Delta_T(s_i, T) \\ &\le d_{G^*}(s, s_i) + 3\alpha d_{G^*}(s, s_i)/5 \\ &< 8d_{G^*}(s, s_i)/5, \text{ or} \\ d_{G^*}(s, s_i) &> 5d_{G^*}(s, T')/8. \end{aligned}$$

If T' is incomplete, from Lemma 5.2, we see that

$$\begin{aligned} \|T'\| &\ge l - d_{G^*}(s, s_i) - \Delta_T(s_i, T) \\ &> (1 + \alpha)d_{G^*}(s, s_i) - d_{G^*}(s, s_i) - 3\alpha d_{G^*}(s, s_i)/5 \\ &= 2\alpha d_{G^*}(s, s_i)/5 \\ &> 2\alpha d_{G^*}(s, T')/8 \\ &> \alpha d_{G^*}(s, T')/5. \end{aligned}$$

Therefore, after each iteration of the main loop of ℓ-`CFX` we see that for any tree $T \in \mathcal{T}$, $\|T\| \ge \alpha d_{G^*}(s, T)/5$. In addition, as with Lemma 3.5, we know that, after the ℓ-`explore` function on a tree T terminates, T becomes complete and is removed from $\mathcal{T}$ (line 14). □

THEOREM 5.5. *Any unknown weighted graph $G = (V, E)$ with source node s, unknown radius r, and unknown deepest edge distance $r_\triangle$ can be explored by a tethered robot using a maximum rope length of $r_\triangle + \alpha r$ with cost $\Theta(|E|/\alpha)$, for $0 < \alpha < 1$.*

PROOF. As with Theorem 3.6, we break the robot traversal steps on a tree T into three groups, $S_1(T)$, $S_2(T)$, $S_3(T)$, defined as before. Using Lemma 5.4, we can bound $S_1(T) = 2d_{G^*}(s, s_i) \leq 10\|T\|/\alpha$. Recall that $S_2(T) = 2|T| \leq 2\|T\|$ and that $\sum_{T \in \mathcal{T}_e} S_3(T) = 2|E(G)|$, where $\mathcal{T}_e$ is the set of all trees explored.

We now need to bound the *extended* size of all trees explored. After any iteration of the algorithm, all trees in $\mathcal{T}$ are node disjoint. However, as discussed in the proof of Theorem 3.6, the number of nodes in all trees is not bounded simply by $|V|$. If we let T be the current tree explored, then from Lemma 5.4, we know that after the ℓ-`explore` function call of line 13, T is complete. As with Theorem 3.6, this implies that any node can be in at most *two* trees, once as an incomplete node and once as a complete node. Unlike the previous theorems, though, we are concerned with the extended size of the trees. Recall from the definition that every edge is included in the extended size only if one of its corresponding nodes is in the tree. This immediately implies that any edge can be included in the extended edge set of at most four trees, two per node. Hence, $\sum_{T \in \mathcal{T}_e} \|T\| \leq 4|E(G)|$.

We can bound the total exploration steps $\mathcal{S}$ taken by the ℓ-`CFX` algorithm as

$$\begin{aligned}
\mathcal{S} &= \textstyle\sum_{T \in \mathcal{T}_e}(S_1(T) + S_2(T) + S_3(T)) \\
&\leq 2|E| + (2 + 10/\alpha) \textstyle\sum_{T \in \mathcal{T}_e} \|T\| \\
&\leq 2|E| + (2 + 10/\alpha)4|E| \text{ which is } \Theta(|E|/\alpha).
\end{aligned}$$

We note that our constant factors are intentionally overestimated to simplify the analysis. By Lemma 3.4, we know that once the algorithm terminates it has fully explored the graph. We now only need to show that the length of the rope used is no more than $r_\triangle + \alpha r$. For every iteration on a tree T_i with closest node s_i, we use a rope of length $(1 + \alpha)d_{G^*}(s, s_i) + \ell'$. Let $e = (u, v) \in EE(T)$ be the longest unexplored extended edge in T; that is, $\ell' = \ell(e)$. From the definition of the distance to an edge and without loss of generality, let $d_G(s, e) = d_G(s, v) + \ell(e)$. Notice that v may not necessarily be in T. Let v' be the closest incomplete node to s on $P_G(s, v)$, which may be v itself. Such a node must exist since v is either incomplete or unexplored. Since there are no other incomplete nodes from s to v' we know that $P_G(s, v') = P_{G^*}(s, v')$. From Lemma 3.4 and the fact that s_i is the closest node among all nodes in $\mathcal{T}$, we know that $d_G(s, v') = d_{G^*}(s, v') \geq d_{G^*}(s, s_i)$. Using the definition of the deepest edge, $e_\triangle$, we see that

$$\begin{aligned}
r_\triangle &= d_G(s, e_\triangle) \\
&\geq d_G(s, e) \\
&= d_G(s, v) + \ell' \\
&\geq d_G(s, v') + \ell' \\
&\geq d_{G^*}(s, s_i) + \ell'
\end{aligned}$$

As in the proof of Theorem 4.2, we also know that $d_{G^*}(s, s_i) \leq r$. Therefore, we see that $(1 + \alpha)d_{G^*}(s, s_i) + \ell' \leq r_\triangle + \alpha r$. □

From Lemma 5.1 and the fact that $r_\Delta > r$, we achieve similar bounds for the piecemeal search.

COROLLARY 5.6. *Any unknown weighted graph $G = (V, E)$ with source node s, unknown radius r, and unknown deepest edge distance r_Δ can be explored by a fuel-constrained robot using a tank of size at most $2(1 + \beta)r_\Delta$ with cost $\Theta(|E|/\beta)$, for $0 < \beta < 1$.*

6. *Conclusion and Open Problems*

We have presented a linear-cost algorithm for the exploration of general unknown, weighted graphs with unknown radius and thus answered the open problem of Awerbuch et al. [1999a]. We conclude with some interesting open questions.

— It is critical for our algorithm to use a rope of length $(1+\alpha)r$. It is interesting to see what is the minimum rope length that still allows linear exploration cost. We conjecture that $r + o(r)$ is not sufficient. On the other hand, it is easy to see that piecemeal search with a fuel capacity of $2r + o(r)$ is not possible in linear steps on the "popsicle" graph consisting of a path of length $n/2$ connected to a clique on $n/2$ nodes. Let the start node be the first node on the path, so $r = n/2$. If the fuel limit is $n + f(n)$, it necessarily takes at least $\Theta(\frac{n^3}{f(n)})$ exploration steps.
— Another related question is whether a linear-cost algorithm exists for the treasure hunting problem of Awerbuch et al. [1999a]. In other words, does Corollary 4.4 hold with $\alpha = 0$ and only $\Theta(|E| + |V|)$ steps?
— In the unweighted case, the exploration cost of our algorithm is $2|E| + O(|V|)$. In light of Panaite and Pelc [1999], it is worth exploring whether an $|E| + O(|V|)$ algorithm exists for the fuel-constrained or the distance-constrained search.
— In the weighted case, our algorithm's dependency on α became $\Theta(|E|/\alpha)$. It is an interesting challenge to reduce this dependency and develop a $\Theta(|E| + f(G, \alpha))$ algorithm where f is some function that depends on α and specific properties of the graph. For example, it may be possible to state the performance in terms of the size of a (minimum) spanning tree in G.

ACKNOWLEDGMENT. The authors would like to thank the anonymous referees for their helpful comments and suggestions.

REFERENCES

ALBERS, S., AND HENZINGER, M. R. 1999. Exploring unknown environments. *SIAM J. Comput. 29*, 4, 1164–1188.

AVERBAKH, I., AND BERMAN, O. 1996. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. *Disc. Appl. Math. 68*, 1–2, 17–32.

AVERBAKH, I., AND BERMAN, O. 1997. $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. *Disc. Appl. Math. 75*, 3, 201–216.

AWERBUCH, B., BERGER, B., COWEN, L., AND PELEG, D. 1999a. Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput. 28*, 1 (Feb.), 263–277.

AWERBUCH, B., BETKE, M., RIVEST, R. L., AND SINGH, M. 1999b. Piecemeal graph exploration by a mobile robot. *Inf. Comput. 152*, 2, 155–172.

AWERBUCH, B., AND KOBOUROV, S. 1998. Polylogarithmic-overhead piecemeal graph exploration. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT)*. 280–286.

BAR-ELI, E., BERMAN, P., FIAT, A., AND YAN, P. 1994. On-line navigation in a room. *J. Algorithms 17*, 3, 319–341.

BENDER, M. A., FERNÁNDEZ, A., RON, D., SAHAI, A., AND VADHAN, S. P. 2002. The power of a pebble: Exploring and mapping directed graphs. *Inf. Comput. 176*, 1, 1–21.

BENDER, M. A., AND SLONIM, D. K. 1994. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, CA, 75–87.

BERMAN, P., BLUM, A., FIAT, A., KARLOFF, H., ROSÉN, A., AND SAKS, M. 1996. Randomized robot navigation algorithms. In *Proceedings of the 7th Annual Symposium on Discrete Algorithms (SODA)*. ACM, New York, 75–84.

BETKE, M., RIVEST, R. L., AND SINGH, M. 1995. Piecemeal learning of an unknown environment. *Mach. Learn. 18*, 2/3, 231–254.

BLUM, A., AND CHALASANI, P. 2000. An online algorithm for improving performance in navigation. *SIAM J. Comput. 29*, 6, 1907–1938.

BLUM, A., RAGHAVAN, P., AND SCHIEBER, B. 1997. Navigating in unfamiliar geometric terrain. *SIAM J. Comput. 26*, 1 (Feb.), 110–137.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill, New York.

DENG, X., KAMEDA, T., AND PAPADIMITRIOU, C. 1998. How to learn an unknown environment. I: the rectilinear case. *J. ACM 45*, 2 (Mar.), 215–245.

DENG, X., AND PAPADIMITRIOU, C. 1999. Exploring an unknown graph. *J. Graph Theory 32*, 265–297.

DUNCAN, C. A., KOBOUROV, S. G., AND KUMAR, V. S. A. 2001. Optimal constrained graph exploration. In *Proceedings of the 12*th *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, New York, 807–814.

FRAIGNIAUD, P., GASIENIEC, L., KOWALSKI, D. R., AND PELC, A. 2004. Collective tree exploration. In *LATIN*. 141–151.

HOFFMANN, F., ICKING, C., KLEIN, R., AND KRIEGEL, K. 2002. The polygon exploration problem. *SIAM J. Comput. 31*, 2 (Apr.), 577–600.

PANAITE, P., AND PELC, A. 1999. Exploring unknown undirected graphs. *J. Algorithms 33*, 2, 281–295.

PAPADIMITRIOU, C., AND YANAKAKIS, M. 1991. Shortest paths without a map. *Theoret. Comput. Sci. 84*, 127–150.

RAO, N. S. V., KARETI, S., SHI, W., AND IYENGAR, S. S. 1993. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Tech. Rep., Oak Ridge National Laboratory. July. ORNL/TM-12410.

RIVEST, R. L., AND SCHAPIRE, R. E. 1993. Inference of finite automata using homing sequences. *Inf. Comput. 103*, 2, 299–347.

RECEIVED APRIL 2005; REVISED DECEMBER 2005; ACCEPTED DECEMBER 2005