Exploring Unknown Environments

Susanne Albers* Monika R. Henzinger†

Abstract

We consider exploration problems where a robot has to construct a complete map of an unknown environment. We assume that the environment is modeled by a directed, strongly connected graph. The robot's task is to visit all nodes and edges of the graph using the minimum number R of edge traversals. Koutsoupias [12] gave a lower bound for R of $\Omega(d^2m)$, and Deng and Papadimitriou [9] showed an upper bound of $d^{O(d)}m$, where m is the number edges in the graph and d is the minimum number of edges that have to be added to make the graph Eulerian. We give the first sub-exponential algorithm for this exploration problem, which achieves an upper bound of $d^{O(\log d)}m$. We also show a matching lower bound of $d^{\Omega(\log d)}m$ for our algorithm. Additionally, we give lower bounds of $2^{\Omega(d)}m$, resp. $d^{\Omega(\log d)}m$ for various other natural exploration algorithms.

1 Introduction

Suppose that a robot has to construct a complete map of an unknown environment using a path that is as short as possible. In many situations it is convenient to model the environment in which the robot operates by a graph. This allows to neglect geometric features of the environment and to concentrate on combinatorial aspects of the exploration problem. Deng and Papadimitriou [9] formulated thus the following exploration problem. A robot has to explore all nodes and edges of an unknown, strongly connected directed graph. The robot *visits* an edge when it traverses the edge. A node or edge is *explored* when it is visited for the first time. The goal is to determine a *map*, i.e. the adjacency matrix, of the graph using the minimum number R of edge traversals. At any point in time the robot knows (1) all visited nodes and edges and can recognize them when encountered again; and (2) the number of unvisited edges leaving any visited node. The robot does not know the head of unvisited edges leaving a visited node or the unvisited edges leading into a visited node. At each point in time, the robot visits a *current node* and has the choice of leaving the current node by traversing a specific known or an arbitrary (i.e. given by an adversary) unvisited outgoing edge. An edge can only be traversed from tail to head, not vice versa.

If the graph is Eulerian, $2m$ edge traversals suffice [9], where m is the number of edges. This immediately implies that undirected graphs can be explored with at most $4m$ traversals. For a non-Eulerian graph, let the *deficiency* d be the minimum number of edges that have to be added to make the graph Eulerian. Deng and Papadimitriou [9] suggested to study the dependence of R on m and d and showed the first upper and lower bounds: they gave a graph such that any algorithm needs $\Omega(d^2m/\log d)$ edge traversals, and they also presented an algorithm that achieves an upper bound of $d^{O(d)}m$. Koutsoupias [12] improved the lower bound to $\Omega(d^2m)$. Deng and Papadimitriou asked the question whether the exponential gap between the upper and lower bound can be closed. Our paper is a first step in this direction: we give an algorithm that is sub-exponential in d, namely it achieves an upper bound of $d^{O(\log d)}m$. We also show a matching lower bound for our algorithm and exponential lower bounds for various other exploration algorithms.

Note that d arises also in the complexity of the "offline" version of the problem: Consider a directed cycle with one edge replaced by $d+1$ parallel edges. On this graph any Eulerian traversal requires $\Omega(dm)$ edge traversals. A simple modification of the Eulerian online algorithm solves the offline problem on any directed graph with $O(dm)$ edge traversals.

Related Work. Exploration and navigation problems for robots have been studied extensively in the past. The exploration problem in this paper was formulated by Deng and Papadimitriou based on a learning

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: albers@mpi-sb.mpg.de. Work supported in part by the Deutsche Forschungsgemeinschaft, Project Al 464/1-1.

†Systems Research Center, Digital Equipment Corporation, 130 Lytton Ave, Palo Alto, CA 94301. Email: monika@pa.dec.com. This work was supported by an NSF CAREER Award, Grant No. CCR-9501712.

problem proposed by Rivest [14]. Betke *et al.* [6] and Awerbuch *et al.* [1] studied the problem of exploring an undirected graph and requiring additionally that the robot returns to its starting point every so often. Bender and Slonim [7] showed how two cooperating robots can learn a directed graph with indistinguishable nodes, where each node has the same number of outgoing edges. Subsequent to the work in [9], Deng *et al.* [8] investigated a geometric exploration problem, whose goal is to explore a room with or without polygonal obstacles. Hoffmann *et al.* [11] gave an improved exploration strategy for rooms without obstacles. More generally, theoretical studies of exploration and navigation problems in unknown environments were initiated by Papadimitriou and Yannakakis [13]. They considered the problem of finding a shortest path from a point s to a point t in an unknown environment and presented many geometric and graph based variants of this problem. Blum *et al.* [5] investigated the problem of finding a shortest path in an unfamiliar terrain with convex obstacles. More work on this problem includes [2, 3, 4].

Our Results. Our main result is a new robot strategy that explores an arbitrary graph with deficiency d and traverses each edge at most $(d+1)^6 d^{2\log d}$ times, see Section 3. (The total number of traversals is also $O(\min\{dn^2+m, nm\})$, where n is the number of nodes.) The algorithm does not need to know d in advance. In Section 4 we demonstrate that our analysis is tight: There exists a graph that is explored by our algorithm using $d^{\Omega(\log d)}m$ edge traversals. We also show that various variants of the algorithm have the same lower bound. In Section 2, we sketch lower bounds of $2^{\Omega(d)}m$, resp. $d^{\Omega(\log d)}m$ for various other natural exploration algorithms to give some intuition for the problem.

Our exploration algorithm tries to explore new edges that have not been visited so far. That is, starting at some visited node x with unvisited outgoing edges, the robot explores new edges until it gets *stuck* at a node y, i.e., it reaches y on an unvisited incoming edge and y has no unvisited outgoing edge. Since the robot is not allowed to traverse edges in the reverse direction, an adversary can always force the robot to visit unvisited nodes until it finally gets stuck at a visited node.

The robot then relocates, using visited edges, to some visited node z with unexplored outgoing edges and continues the exploration. The *choice* of z is the only difference between various algorithms and the *relocation* to z is the only step where the robot traverses visited edges. To minimize R we have to minimize the total number of edges traversed during all relocations. It turns out that a locally greedy algorithm that tries to minimize the number of traversed edges during each relocation is not optimal: it has a lower bound of $2^{\Omega(d)}m$ (see Section 2).

Instead, our algorithm uses a divide-and-conquer approach. The robot explores a graph with deficiency d by exploring d^2 subgraphs with deficiencies $d/2$ each and uses the same approach recursively on each of the subgraphs. To create subgraphs with small deficiencies, the robot keeps track of visited nodes that have more visited outgoing than visited incoming edges. Intuitively, these nodes are *expensive* because the robot, when exploring new edges, can get stuck there. The relocation strategy tries to keep portions of the explored subgraphs "balanced" with respect to their expensive nodes. If the robot gets stuck at some node, then it relocates to a node z such that "its" portion of the explored subgraph contains the minimum number of expensive nodes.

2 Lower bounds for various algorithms

In this section we give lower bounds of $2^{\Omega(d)}m$, resp. $d^{\Omega(\log d)}m$ for a locally greedy, a generalized greedy, a depth-first, and a breadth-first algorithm. Let G be a directed, strongly connected graph and let v be a node of G. Let $in(v)$ and $out(v)$ denote the number of incoming, resp. outgoing edges of v. Let the *balance* $bal(v) = out(v) - in(v)$. For a graph with deficiency d there exist at most d nodes s_i, $1 \le i \le d$, such that $bal(s_i) < 0$. Every node s_i with $bal(s_i) < 0$ is called a *sink*. Note that $-\sum_{s,bal(s)<0} bal(s) = d$. We use the term *chain* to denote a path. A chain is a sequence of nodes and edges $x_1, (x_1,x_2), x_2, (x_2,x_3), \ldots, (x_{k-1},x_k), x_k$ for $k > 1$.

Greedy: If stuck at a node y, move to the nearest node z that has new outgoing edges.

Generalized-Greedy: At any time, for each path in the subgraph explored so far, define a lexicographic vector as follows. For each edge on the path, determine its current *cost*, which is the number of times the edge was traversed so far. Sort these costs in non-increasing order and assign this vector to the path. Whenever stuck at a node y, out of all paths to nodes with new outgoing edges traverse the path whose vector is lexicographic minimum.

Depth-First: If stuck at a node y, move to the most recently discovered node z that can be reached and that has new outgoing edges.

Breadth-First: Let v be the node where the exploration starts initially. If stuck at a node y, move to the node z that has the smallest distance from v among all nodes with new outgoing edges that can be reached from y.

Theorem 1 *For Greedy, Depth-First, and Breadth-First and for every d, there exist graphs of deficiency d that require $2^{\Omega(d)}m$ edge traversals. For Generalized-Greedy and for every d, there exists a graph of deficiency d that requires $d^{\Omega(\log d)}m$ edge traversals.*

Proof: *Greedy:* Basically *Greedy* fails since it is easy to "hide" a subgraph. Whenever *Greedy* discovers this

subgraph, the adversary can force it to repeat all the work done so far.

The graph G consists of two parts, (1) a cycle C_0 of three edges and nodes v, $v^1(C_0)$, and $v^2(C_0)$, and (2) a recursively defined problem P^d. A *problem P^δ of deficiency δ*, for any integer $\delta \geq 2$, is a subgraph that has two *incoming* edges whose startnodes do not belong to P^δ but whose endnodes do, and δ *outgoing* edges whose startnode belongs to P^δ but whose endnodes do not. A *problem P^1 of deficiency 1* is defined in the same way as a problem P^δ, $\delta \geq 2$, except that P^1 has only one incoming edge. In the case of P^d, the two incoming edges start at $v^1(C_0)$ and $v^2(C_0)$, respectively; the d outgoing edges all point to v.

A problem P^δ, $\delta = 1, 2$, consists of δ chains of three edges each. The first edge of each chain is an incoming edge into P^δ; the last edge of each chain is an outgoing edge. For $\delta > 2$, one of the incoming edges of P^δ is the first edge of a chain D_1^δ consisting of three edges, the other incoming edge is the first edge of a long chain D_2^δ. For each of these chains D_j^δ, $j = 1, 2$, (a) the last edge is an outgoing edge of P^δ, and (b) each of the last two interior nodes has one unvisited outgoing edge pointing into a recursive subproblem $P_j^{\delta-2}$. There are $\delta-2$ edges leaving $P_1^{\delta-2}$, all of which also represent edges leaving P^δ. Also, there are $\delta-2$ edges leaving $P_2^{\delta-2}$ and pointing to nodes of $P_1^{\delta-2}$ such that each node in $P_1^{\delta-2}$ that has k more outgoing than incoming edges receives k incoming edges from $P_2^{\delta-2}$. The total number of edges in D_2^δ is 2 plus the number of edges of D_1^δ plus the total number of edges contained in the subproblem $P_1^{\delta-2}$ below D_1^δ.

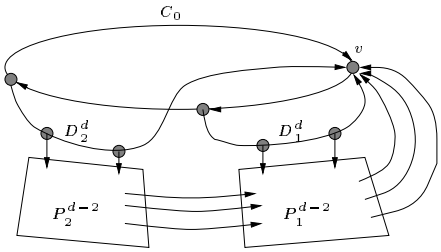

Figure 1: The graph for *Greedy*

Greedy is started at node v and traverses first chain C_0. Then it either explores D_1^d or D_2^d. In either case, afterwards *Greedy* explores all edges of P_1^{d-2} since D_2^d is prohibitively long. Thus, P_2^{d-2} is "hidden" from *Greedy*. We exploit this in the analysis: Let $N(\delta)$ be the number of times that *Greedy* explores edges of a problem P^δ, gets stuck at some node and cannot relocate to a suitable node by using only edges in P^δ. We show that $N(\delta) \geq 2^{\delta/2}$. Since the edge leaving v is traversed every time the algorithm cannot relocate by using only edges in P^d, the bound follows.

A problem P^δ contains two subproblems $P_1^{\delta-2}$ and $P_2^{\delta-2}$. Note (a) that, because of chain $D_1^{\delta-2}$, no node in $P_1^{\delta-2}$ can reach a node of $P_2^{\delta-2}$ without leaving P^δ. Note (b) that $P_1^{\delta-2}$ is completely explored when the exploration of $P_2^{\delta-2}$ starts and all paths starting in $P_2^{\delta-2}$ lead through $P_1^{\delta-2}$. Thus, every time *Greedy* gets stuck in a subproblem $P_j^{\delta-2}$, $j \in \{1, 2\}$, and has to leave $P_j^{\delta-2}$ in order to resume exploration, it also has to leave P^δ. For $P_1^{\delta-2}$ the statement follows from (a); for $P_2^{\delta-2}$ it follows from (a) and (b). Thus, $N(\delta) \geq 2N(\delta-2)$. Since, for $\delta = 1, 2$, $N(\delta) \geq 1$, we obtain $N(\delta) \geq 2^{\delta/2}$.

This implies that the edge e on C_0 leaving v is traversed $2^{\Omega(d)}$ times. The desired bound follows by replacing e by a path consisting of $\Theta(m)$ edges.

Depth-First: We can use the same graph as in the case of the *Greedy* algorithm. *Depth-First* will explore all edges in P_1^{d-2} before it will start exploring P_2^{d-2}.

Generalized-Greedy and Breadth-First: Proof omitted. □

3 The *Balance* algorithm

3.1 The algorithm

We present an algorithm that explores an unknown, strongly connected graph with deficiency d, without knowing d in advance. First we give some definitions. At the start of the algorithm, all edges are *unvisited* or *new*. An edge becomes *visited* whenever the robot traverses it. A node is *finished* whenever all its outgoing edges are visited. The robot is *stuck* at a node y if the robot enters a finished node y on an unvisited edge. A sink is *discovered* whenever the robot gets stuck at the sink for the first time. We assume that whenever the robot discovers a new sink, the subgraph of explored edges is strongly connected. This does not hold in general, but by properly restarting the algorithm at most d times the problem can be reduced to the case described here (details are given in the full version of the paper).

Assume the algorithm knew the d missing edges $(s_1, t_1), (s_2, t_2), \ldots, (s_d, t_d)$ and a path from each s_i to t_i. Then a modified version of the Eulerian algorithm could be executed: Whenever the original Eulerian algorithm traverses an edge (s_i, t_i), the modified Eulerian algorithm traverses the corresponding path from s_i to t_i. Obviously, the modified algorithm traverses each edge at most $2d + 2$ times. Thus, the problem is to find the missing edges and corresponding paths.

Our algorithm tries to find the missing edges by maintaining d edge-disjoint chains such that the endnode of chain i is s_i and the startnode of chain i is our current *guess* of t_i. As the algorithm progresses paths can be appended at the start of each chain. At termination, the startnode of chain i is indeed t_i. To mark chain i all edges on chain i are colored with color i.

The algorithm consists of two phases.

Phase 1: Run the algorithm of [9] for Eulerian graphs. Since G is not Eulerian, the robot will get stuck at a sink s. At this point stop the Eulerian graph algorithm and goto Phase 2. The part of the graph explored so far contains a cycle C_0 containing s [9]. We assume that at the end of Phase 1 all visited nodes and edges not belonging to C_0 are marked again as unvisited.

Phase 2: Phase 2 consists of *subphases*. During each subphase the robot visits a *current* node x of a *current* chain C and makes progress towards finishing the nodes of C. The current node of the first subphase is s, its current chain is C_0. The current node and current chain of subphase j depend on the outcome of subphase $j-1$.

A chain can be in one of three states: *fresh*, *in progress*, or *finished*. A chain C is *finished* when all its nodes are finished; C is *in progress* in subphase j if C was a current chain in a subphase $j' \leq j$ and C is not yet finished; C is *fresh* if its edges are explored, but C is not yet in progress.

At the same time up to $d+1$ chains in progress and up to d fresh chains can exist. The invariant that there are always at most $d+1$ chains in progress is convenient but not essential in the analysis of the algorithm. The invariant that there exist always at most d fresh chains in crucial. Every startnode of a fresh chain has more visited outgoing that visited incoming edges and, thus, the robot can get stuck there. In the analysis we require that there always exist at most d such nodes.

The algorithm marks the current guess for t_i with a *token* τ_i, for $1 \leq i \leq d$. In fact, every startnode of a fresh chain represents the current guess for some t_i, $1 \leq i \leq d$, and thus has a token τ_i. To simplify the description of the relocation process, each token is also assigned an *owner* which is a chain that contains the node on which the token is placed. Note that a node can be the current guess for more than one node t_i and, thus, have more than one token.

From a high-level point of view, at any time, the subgraph explored so far is partitioned into chains, namely C_0 and the chains generated in Phase 2. During the actual exploration in the subphases, the robot travels between chains. While doing so, it generates or extends fresh chains, which will be taken into progress later, and finishes the chains currently in progress.

We give the details of a subphase. First, the algorithm tests if x has an unvisited outgoing edge.

1. If x does not have an unvisited outgoing edge and x is not the endnode of C, then the next node of C becomes the current node and a new subphase is started.

2. If x has no unvisited outgoing edge and x is the endnode of C, procedure *Relocate* is called to decide which chain becomes the current chain and to move the robot to the startnode z of this chain. Node z becomes the current node.

3. If x has unvisited outgoing edges, the robot repeatedly explores unvisited edges until it gets stuck at a node y. Let P be the path traversed. We distinguish four cases:

   Case 1: $y = x$
   Cut C at x and add P to C. See Figure 2. The robot returns to x and the next phase has the same current node and current chain.

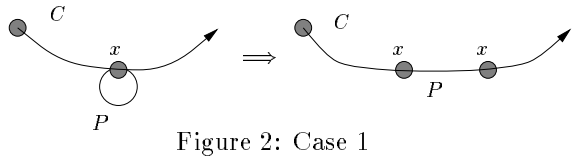

Figure 2: Case 1

   Case 2: $y \neq x$, y has a token τ_i and is the startnode of a fresh chain D (see Figure 3)
   Append P at D to create a longer fresh chain, and move the token from y to x. The current chain C becomes the *owner* of the token, the previous owner becomes the current chain, and y becomes the current node.

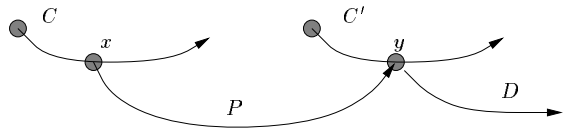

Figure 3: Case 2

   Case 3: $y \neq x$, y has a token τ_i but is not the startnode of a fresh chain.
   This is the same as Case 2 except that no fresh chain starts at y. The algorithm creates a new fresh chain of color i consisting of P. It moves the token from y to x and C becomes the owner of the token. The previous owner of the token becomes the current chain and y becomes the current node.

   Case 4: $y \neq x$ and y does not own a token.
   In this case $bal(y) < 0$. If $bal(y) = -k$, then this case occurs k times for y. Let i be the number of existing tokens. The algorithm puts a new token τ_{i+1} on x with owner C, creates a fresh chain of color $i+1$ consisting of P (the first chain with color $i+1$), and moves the robot back to s. The initial chain C_0 becomes the current chain, s becomes the current node.

This leads to the algorithm given in Figure 4. We use x to denote the current node, C to denote the current chain, k the number of tokens used, and j the highest index of a chain. Lines 4–17 of the code correspond to item 3 above. Line 6 and 7 correspond to Case 1, lines 8–13 correspond to Cases 2 and 3, and lines 14–16 to Case 4. Lines 18 and 19 implement item 2 and item 1, respectively.

Additionally, the algorithm maintains a tree T such that each chain C corresponds to a node $v(C)$ of T and

Algorithm Balance

1. $j := 0$, $k := 1$, $x := s$, $C := C_0$.
2. **repeat**
3. **while** C is unfinished **do**
4. **while** $\exists$ new outgoing edge at x **do**
5. Traverse new edges starting at x until stuck at a node y. Call this path P.
6. **if** $y = x$ **then**
7. Insert P into C;
8. **else if** y has a token **then**
9. **if** $\exists$ chain D of color i starting in y and D is fresh **then**
10. $C' := owner(\tau_i)$. Concatenate P with D;
11. **else**
12. $j := j + 1$; $C_j :=$ chain that consists of P;
13. Place τ_i on x; $owner(\tau_i) := C$; $x := y$; $C := C'$;
14. **else** (* $y \neq x$ and y has no token *)
15. $j := j + 1$; $C_j :=$ chain that consists of P;
16. $k := k + 1$; Place token τ_k on x; $owner(\tau_k) := C$; $x := s$; $C := C_0$;
17. Move robot to x;
18. Move robot to first unfinished node z that appears on C after its startnode; $x := z$;
19. $C := Relocate(C)$; $x =$ startnode of C;
20. **until** $C =$ empty_chain.

Figure 4: The *Balance* algorithm

$v(C')$ is a child of $v(C)$ if the last subpath appended to C' was explored while C was the current chain. Reversely, we use $C(v)$ to denote the chain represented by node v. We use T_v to denote the subtree of T rooted at v and say C is *contained* in T_v if $v(C)$ lies in T_v. We also say a token τ or an edge e is *contained* in T_v if $owner(\tau)$, respectively the chain of e is contained in T_v. If all chains in T_v are finished, we say that T_v is *finished*. To represent T, the algorithm assigns a *parent* to each chain.

To relocate the robot needs to be able to move on explored edges from the endpoint of a chain C to its startnode. This is always possible, since at the beginning of each subphase the explored edges form a strongly connected graph. To avoid that an edge is traversed often for this purpose, we define for each chain C a path $closure(C)$ connecting the endnode of C with the startnode of C such that an edge belongs to $closure(C)$ for at most $d^{O(\log d)}$ chains C. Finally, we will show that $closure(C)$ is traversed at most $O(d^2)$ times.

A path Q is called a *C-completion* if it connects the endnode of a chain C with the startnode of C. A path Q in the graph is called *i-uniform* if it is a concatenation of chains of color i. Let u be a node of T. A path Q in the graph is *T_u-homogeneous* if any maximal subpath R of Q that does not belong to T_u is (a) i-uniform for some color i; (b) the edge of Q preceding R is the last edge of a chain of color i; and (c) the edge of Q after R is the first edge of a chain of color i.

We try to choose $closure(C)$ to be "as local to C" as possible: Let $S(C)$ be the set of explored edges when C becomes the current chain for the first time. Given $S(C)$, $a(C)$ is the lowest ancestor of $v(C)$ in T such that a $T_{a(C)}$-homogeneous completion of C exists in $S(C)$. Note that $a(C)$ is well-defined since each chain has a $T_{v(C_0)}$-homogeneous completion. The path $closure(C)$ is an arbitrary $T_{a(C)}$-homogeneous completion of C using only edges of $S(C)$. The algorithm can compute $closure(C)$ whenever C becomes the current chain for the first time without moving the robot.

Procedure Relocate(C)

1. **if** all chains are finished **then**
   return(empty_chain).
2. **else** Move robot to startnode of C along $closure(C)$;
3. **while** $C \neq C_0$ and $T_{v(C)}$ is finished **do**
4. Move robot to startnode of $parent(C)$ along $closure(parent(C))$;
5. $C := parent(C)$;
6. **while** C is finished **do**
7. Let $C_1, C_2, \ldots, C_l$ be the chains with $parent(C_k) = C$, $1 \leq k \leq l$. Let C_k be the chain such that $T_{v(C_k)}$ contains the smallest number of tokens among all $T_{v(C_1)}, \ldots, T_{v(C_l)}$ with unfinished chains;
8. $C := C_k$; $x :=$ startnode of C;
9. Move robot to x;
10. **if** C is not in progress **then**
11. Compute $closure(C)$;
12. **return**(C).

We describe the *Relocation* procedure. In the relocation step, the robot repeatedly moves from the current chain to its parent until it reaches a chain C such that

$T_{v(C)}$ is unfinished. To move from a chain X to its parent X', the robot proceeds along X to the endnode of X and traverses *closure*(X) to the startnode of X, which belongs to X'. When reaching C, the robot repeatedly moves from the startnode of the current chain X to the startnode of one of its children until it reaches the startnode of an unfinished chain. It chooses the child X' of X such that among all subtrees rooted at children of X and containing unfinished chains, $T_{v(X')}$ has the minimum number of tokens.

3.2 The analysis of the algorithm

3.2.1 Correctness

Since the graph is strongly connected, all nodes of the graph must be visited during the execution of the algorithm. When the algorithm terminates, all visited nodes are finished. Thus, all edges must be explored. We show next that each operation and each move of the robot are well-defined. Proposition 1 shows that if a chain of color i is fresh, then τ_i lies at the startnode of the chain. Thus, in line 10, token τ_i lies on y. By assumption there exists a path from any finished node to s. Thus, the move in line 17 is well-defined. In line 18, the robot moves to the next unfinished node of the current chain C. It would be possible to walk along *closure(C)*, but the proof of Lemma 4 shows later that *closure(C)* is not needed.

3.2.2 Fundamental properties of the algorithm

Lemma 1 *At most d tokens are introduced during the execution of the Balance algorithm.*

Proof: We say that the algorithm first introduces the token τ_k at y in line 16.

Let $in_v(v)$ and $out_v(v)$ denoted the number of visited incoming and visited outgoing edges of v, respectively. Let $t(v)$ be the total number of tokens introduced on node v in line 16. We show inductively that $\max\{in_v(v) - out_v(v), 0\} = t(v)$. Since at termination $in_v(v) = in(v)$ and $out_v(v) = out(v)$, it follows that $-bal(v) \geq t(v)$ if $bal(v) < 0$ and $t(v) = 0$, otherwise. Thus, $d = -\sum_{v,withbal(v)<0} bal(v) \geq \sum_v t(v)$.

The claim $\max\{in_v(v) - out_v(v), 0\} = t(v)$ holds initially. Let P be the newly explored path when the first token is placed on v, i.e. when the algorithm gets stuck at v for the first time. Before P enters v, $in_v(v) = out_v(v)$. Traversing P increments $in_v(v)$ by 1 and sets $in_v(v) - out_v(v) = 1$. Thus, the claim holds. Let P be the newly explored path when token i is placed on v. It follows inductively that $in_v(v) - out_v(v) = i - 1$ before P enters v and traversing P increments the value by 1 as before. □

We prove next some invariants.

Proposition 1 *1. For every chain C that is in progress or finished, parent(C) is finished.*

2. Let C be a chain of color i, $1 \leq i \leq d$. (a) If C is fresh, C does not own a token, τ_i is located at the startnode of C, and parent(C) = owner(τ_i). (b) If C is in progress and not the current chain, then C is the owner of some token τ.

3. Every chain C is the parent of at most d chains.

Proof: Part 1. Procedure *Relocate* ensures that *parent(C)* is finished before C is taken into progress.

Part 2a. When C is first created in line 12 or 15 of *Balance*, τ_i is placed on the startnode of C. Whenever the robot gets stuck at the current startnode of C and removes τ_i, chain C is extended by a path P because C is not in progress. Token τ_i is placed on the new startnode of C. Lines 13 and 16 ensure that the parent of C is always the owner of τ_i.

Part 2b. We show that whenever C is the current chain and *Balance* leaves C to continue work on an other chain, C becomes the owner of a token. Chain C is unfinished. Thus, if C is the current chain, *Balance* can only leave C to continue work on an other chain during lines 5–17 of the algorithm. In this situation, *Balance* places a token on a node of C and C becomes the owner of that token.

Part 3. Chain C can become the parent of other chains while C is in progress and unfinished. During this time, every chain C' with *parent*$(C') = C$ is unfinished and not in progress, see Part 1. By Part 2a, the startnode of such a chain C' holds a token and C is the owner of that token. Since there are only d token, the proposition follows. □

The next lemma shows that our algorithm always balances the number of tokens contained in neighboring subtrees of T. For a subtree T_v of T, let the *weight* $w(T_v)$ be the number of tokens contained in T_v. Let $active(T_v) = 1$ if the current chain is in T_v; otherwise let $active(T_v) = 0$.

Lemma 2 *Let $u, v \in T$ be siblings in T such that T_u and T_v contain unfinished chains. Then $|w(T_u) + active(T_u) - w(T_v) - active(T_v)| \leq 1$.*

Proof: Let $active(C) = 1$ iff C is the current chain, and let $active(C) = 0$ otherwise. Let $token(C)$ be the number of tokens owned by C, and let $g(C) = token(C) + active(C)$. Finally, let $g(v) = \sum_{C,v(C)\in T_v} g(C) = w(T_v) + active(T_v)$. We show by induction on the steps of the algorithm that $|g(u) - g(v)| \leq 1$.

The claim holds initially. For a subtree T_v of T, the values $w(T_v)$ and $active(T_v)$ only change in lines 13, 16, and 19 of *Balance* and in lines 4 and 9 of procedure

Relocate. Additionally, T changes in lines 10, 12, and 15.

Note first that changes in T do not affect the invariant: Whenever T changes, $v(C)$ receives a new child and C is not yet finished (or the algorithm has not yet determined that C is finished). Thus, the children of C are not yet in progress, i.e. they do not own any tokens by Proposition 1. Thus, the claim holds for any pair of children of $v(C)$.

We consider next all changes to $w(T_v)$ and $active(T_v)$.

Line 13: Let C be the current chain before the execution of line 13. Note that $token(C)$ increases by 1, $active(C)$ becomes 0, $token(C')$ decreases by 1, and $active(C')$ becomes 1. Thus, $g(C)$ and $g(C')$, and, hence, $g(v)$ is unchanged for every node $v \in T$.

Line 16: Note that (i) $g(C)$ is unchanged by the same argument as for line 13, (ii) $g(C')$ is unchanged, since $token(C')$ and $active(C')$ are unchanged, and (iii) $g(C_0)$ is increased by 1. Since C_0 only contributes to $g(v(C_0))$ and $v(C_0)$ is the root of T, the claim holds.

Line 19 of Balance/Line 4 and 9 of Relocate: Let $\bar{C}$ be the current chain before the execution of line 3 or 7 and let C be the current chain afterwards. In line 3, the claim does not apply to $T_{v(C)}$, since $T_{v(C)}$ is finished. Thus, we are left with line 7. Note that $active(\bar{C})$ drops to 0 and $active(C)$ increases to 1. Thus, for every node v such that T_v contains either both the parent and its child or neither the parent nor its child, $g(v)$ is unchanged. The only remaining subtree is $T_{v(C)}$. Before the execution of line 7, for any sibling C' of C, $w(T_{v(C)}) \leq w(T_{v(C')}) \leq w(T_{v(C)}) + 1$. Since $active(C') = 0$, $|w(T_{v(C)}) - w(T_{v(C')}) + active(C) - active(C')| \leq 1$. □

Lemma 3 *Let C be a chain of color i, $1 \leq i \leq d$, and, at the time when C is taken in progress, let $u \in T$ be the closest ancestor of $v(C)$ that satisfies the following condition. The path from u to $v(C)$ in T contains d nodes $u_1, u_2, \ldots, u_d$ such that each u_j with $1 \leq j \leq d$ has a child v_j*

(a) T_{v_j} contains a node of color i; and (b) $v(C) \notin T_{v_j}$.

If there is no such ancestor u, then let u be $v(C_0)$. Then there exists a T_u-homogeneous C-completion.

Proof: By assumption, the graph of explored edges is strongly connected, which implies that there exists a $T_{v(C_0)}$-homogeneous C-completion. Suppose that there are d nodes $u_1, \ldots, u_d$ satisfying (a) and (b). For $j = 1, \ldots, d$, let C_{u_j} be the chain corresponding to u_j. If one of the nodes $u_1, \ldots, u_d$, say u_k, is of color i, then there is the following T_{u_k}-homogeneous C-completion: Follow edges of color i until you reach the startnode of C_{u_k}, then walk "down" in T_{u_k} along ancestors of C to the startnode of C.

Thus, we are left with the case that none of the nodes $u_1, \ldots, u_d$ has color i. For $j = 1, \ldots, d$, let $C_{j,1} \in T_{v_j}$ be a chain of color i such that no ancestor of $C_{j,1}$ contained in T_{v_j} has color i. Let $C_{j,2}, \ldots, C_{j,l(j)}$ be the ancestors of $C_{j,1}$ in T_{u_j}. More precisely, for $k = 1, \ldots, l(j) - 1$, $C_{j,k+1} = parent(C_{j,k})$ and $C_{j,l(j)} = C_{u_j}$ is the chain corresponding to u_j.

Following the edges of color i gives a T_u-homogeneous path from C to every chain $C_{j,1}$ for $1 \leq j \leq d$. We want to show that there exists a T_u-homogenous path to a chain $C_{j,l(j)}$. We consider the following game on a $d \times \max_j l(j)$ grid, where for $1 \leq j \leq d$, square (j,k) has the color of $C_{j,k}$ for $1 \leq k \leq l(j)$ and no color for $k > l(j)$. Thus, all squares $(j,1)$ have color i and no other squares have color i. Initially all squares $(j,1)$ are checked, all other squares are unchecked. A square is checked if the robot can move to the startnode of the corresponding chain on a T_u-homogeneous path. The rules of the game are: (Note that the startnode of $C_{j',k'-1}$ belongs to $C_{j',k'}$.)

- *A square (j,k) of color i' gets checked whenever there exists a square (j',k') of color i' such that square $(j',k'-1)$ is checked and there exists a path of color-i' edges from the endnode of $C_{j',k'}$ to the startnode of $C_{j,k}$.*

- *The game terminates when one of the squares $(j,l(j))$ is checked or when no more square can be checked.*

We will show that one of the squares $(j,l(j))$ can be checked. This shows that there is a T_u-homogeneous path from C to $C_{j,l(j)}$. Since u_j is an ancestor of $v(C)$, the same argument as above shows that there exists a T_u-homogeneous C-completion.

We employ the pigeon-hole principle: Initially, there are d checked squares $(j,1)$ for $1 \leq j \leq d$ and each square $(j,2)$ has a color $i' \neq i$. Since there are at most $d-1$ other colors, there must be two squares $(s,2)$ and $(t,2)$ with the same color i'. Since the edges of color i' form a chain, there is either a path from $C_{s,2}$ to $C_{t,2}$ or vice versa. Thus, one of the two squares can be checked. Inductively, there are d checked squares $(j,k(j))$ such that $(j,k(j)+1)$ is unchecked. None of the squares $(j,k(j)+1)$ has color i and thus, there must be two squares $(j,k(j)+1)$ with the same color, which leads to checking one of the two squares. The game continues until one of the squares $(j,l(j))$ has been checked. □

3.2.3 Counting the number of edge traversals

Lemma 4 *Each edge is traversed at most d times during executions of line 17 and at most $2d+2$ times during executions of line 18 of the Balance algorithm.*

Proof: Let e be an arbitrary edge and let C be the chain e belongs to. Every time e is traversed during an execution of line 17, a new token is placed on the graph. Since a total of d tokens are placed, the first statement of the lemma follows.

Next we analyze executions of line 18. Let x and y be the tail and the head of e, i.e. $e = (x, y)$. Let C^1 be the portion of C that consists of the path from the startnode of C to x. Similarly, let C^2 be the path from y to the endnode of C.

It is not hard to show that e is traversed for the first time in line 18 when all nodes on C^1 are finished and the robot moves to the next unfinished node on C^2. The edge e can be traversed again (a) if the robot gets stuck at a node on C^1 and moves to the next unfinished node of C, or (b) if the robot traverses C from its startnode, since procedure *Relocate* returned chain C. Every time case (a) occurs, a token is removed from C^1, and this token cannot be placed again on C^1. Since there are only d tokens, e can be traversed at most d more times in case (a) after it was traversed the first time in that line. Every time case (b) occurs, $token(C)+active(C)$ increases by 1, while no other step of the algorithm can decrease this value as long as C is unfinished. Thus, case (b) occurs at most $d+1$ times. □

Thus, it only remains to bound how often an edge is traversed in *Relocate*. A chain C' is *dependent* on a chain C if $C' \in T_{v(C)}$ and $closure(C')$ is not T_u-homogeneous for any true descendant u of $v(C)$.

Lemma 5 *For every chain C, there exist at most $d^{2\log d+1}$ chains $C' \in T_{v(C)}$ that are dependent on C.*

Proof: Let $n_i(C)$ be the total number of chains of color i dependent on C. For a color i, $1 \le i \le d$, and an integer δ, $1 \le \delta \le d$, let

$$N_i(\delta) = \max_C\{n_i(C); T_{v(C)} \text{ contains at most } \delta \text{ of the } d \text{ tokens whenever } active(T_{v(C)}) = 1\}.$$

We will show that for any δ, $1 \le \delta \le d$, and any color i, (1) $N_i(\delta) \le d^2 N_i(\lfloor \delta/2 \rfloor)$ and (2) $N_i(1) = 1$. This implies $N_i(d) \le d^{2\log d}$. Since $\sum_{i=1}^{d} N_i(d) \le d \cdot d^{2\log d}$, the lemma follows.

To prove (1), fix a color i and an integer δ. Consider a subtree $T_{v(C)}$ that contains at most δ tokens when $active(T_{v(C)}) = 1$. Out of all chains dependent on C, let C' be the chain whose closure is computed last. We show that when the algorithm computes $closure(C')$, then the number of chains of color i that are already dependent on C is at most $d(d-1)N_i(\lfloor \delta/2 \rfloor)$. Thus, $n_i(C) \le d(d-1)N_i(\lfloor \delta/2 \rfloor) + 1 \le d^2 N_i(\lfloor \delta/2 \rfloor)$.

Let $u_1, u_2, \ldots, u_l$ be the sequence of nodes (from lowest to highest) on the path from $v(C')$ to $v(C)$ such that every node u_j, $j = 1, 2, \ldots, l$, has a child v_j with (a) T_{v_j} contains a node of color i, and (b) $v(C) \notin T_{v_j}$. By Lemma 3, $l \le d$. Suppose that node u_j, $1 \le j \le d$, has $c(j)$ children, $v_{j,1}, v_{j,2}, \ldots, v_{j,c(j)}$ with $v \in T_{v_{j,1}}$. By condition (b), $2 \le c(j) \le d$.

For fixed j and $k \ge 2$, we have to show: Up to the time when $closure(C')$ is computed, whenever $active(T_{v_{j,k}}) = 1$, then $w(T_{v_{j,k}}) \le \lfloor \delta/2 \rfloor$. Consider the point in time when $closure(C')$ is computed. Since $T_{v_{j,1}}$ contains C', $T_{v_{j,1}}$ is unfinished. By Lemma 2, *Balance* distributes the tokens contained in T_{u_j} evenly among the subtrees $T_{v_{j,1}}, T_{v_{j,2}}, \ldots, T_{v_{j,c(j)}}$ that contain unfinished chains. Thus, for each *unfinished* $T_{v_{j,k}}$ with $k \ge 2$, $w(T_{v_{j,k}})$ was up to now at most $\lfloor \delta/2 \rfloor$ whenever $active(T_{v_{j,k}}) = 1$. For each *finished* $T_{v_{j,k}}$, consider the last point of time when an unfinished chain of $T_{v_{j,k}}$ becomes the current chain. Since $v_{j,1}$ exists, $T_{v_{j,1}}$ is unfinished and, by Lemma 2, $w(T_{v_{j,k}})$ is up to this point in time at most $\lfloor \delta/2 \rfloor$ whenever $active(T_{v_{j,k}}) = 1$. We conclude that up to the time when $closure(C)$ is computed, $T_{v_{j,k}}$ contains at most $N_i(\lfloor \delta/2 \rfloor)$ chains of color i that can be dependent on the chain corresponding to $v_{j,k}$, and, thus, can be dependent on C. Summing up, we obtain that $T_{v(C)}$ contains at most

$$\sum_{j=1}^{d} \sum_{k=2}^{c(j)} N_i(\lfloor \delta/2 \rfloor) \le d(d-1) N_i(\lfloor \delta/2 \rfloor)$$

chains of color i that can be dependent on C.

Finally we show that $N_i(1) = 1$. If a subtree $T_{v(C)}$ contains at most one token whenever $active(T_{v(C)}) = 1$, then each node in $T_{v(C)}$ has only one child, by Proposition 1. Since $T_{v(C)}$ never branches, it can contain at most one chain of color i that is dependent on C. □

Lemma 6 *For every chain C, there exist at most $d^{2\log d+1}$ chains $C' \in T_{v(C)}$ such that closure(C') uses edges of C.*

Proof: Let C be an arbitrary chain and let $v \in T$ be the node corresponding to C. We show that if a chain $C' \in T_{v(C)}$ is not dependent on C, then $closure(C')$ does not use edges of C. Lemma 6 follows immediately from Lemma 5.

If a chain $C' \in T_{v(C)}$ is not dependent on C, then the path $closure(C')$ is T_u-homogeneous for a descendant u of v. Suppose that a T_u-homogeneous path P would use edges of C. Let i be the color of C. Chain C does not belong to T_u. Thus, after P has visited C, it may only traverse chains of color i until it reaches again a chain of color i that belongs to T_u. Note that all chains of color i that are reachable from C via edges of color i must have been generated earlier that C. However, all chain in T_u were generated later than C. We conclude that a T_u-homogeneous path cannot use edges of C. □

Lemma 7 *For every chain C, there exist at most $(d+2)d^{2\log d+2}$ chains $C' \notin T_{v(C)}$ such that closure(C') uses edges of C.*

The proof is omitted.

Theorem 2 *Using the Balance algorithm, the robot explores an unknown graph with deficiency d and traverses each edge at most $(d+1)^6 d^{2\log d}$ times.*

Proof: Let e be an arbitrary edge of chain C. Edge e is traversed for the first time when it is explored during an execution of line 5 of the *Balance* algorithm. By Lemma 4, it can be traversed $3d+2$ times during executions of lines 17 and 18. By Lemmas 6 and 7, e belongs to at most $d^{2\log d+1} + (d+2)d^{2\log d+2}$ paths *closure*(C'). We show that each path *closure*(C') is traversed at most $d(d+1)$ times. The path *closure*(C') is used at most d times during an execution of line 2 of *Relocate*, since each time a token is removed from the finished chain C'. The path *closure*(C') can also be used at most d^2 times in line 4 of *Relocate*, since each time a token is removed from the finished subtree $T_{v(C'')}$ of a child C'' of C'.

Finally, the edge e might be traversed $d(d+1)$ times in line 9 of *Relocate*. When e is traversed in line 9, then (i) either the robot had moved to C_0 after the introduction of a new token (line 16) or (ii) there exists an ancestor u of $v(C)$ with a child x such that the robot was stuck at a node in T_x and T_x is finished. Thus, by going "up" the tree T in lines 3–5, the robot reached u. Case (i) occurs at most d times. When C becomes the current chain for the first time, let $u_1, \ldots, u_l$ be the ancestors of $v(C)$ such that each u_j has a child v_j with (a) T_{v_j} contains unfinished chains, and (b) $v \notin T_{v_j}$. By Proposition 1, the nodes $u_1, \ldots, u_l$ can have a total of d children satisfying (a) and (b). Since each subtree rooted at one of these children can contain at most d tokens, case (ii) occurs at most d^2 times.

Thus, edge e is traversed at most

$$\begin{aligned} &1 + 3d + 2 + d(d+1)(d^{2\log d+1} + (d+2)d^{2\log d+2}) \\ &+d(d+1) \quad \leq \quad (d+1)^5 d^{2\log d} \end{aligned}$$

times. Multiplying the bound by d to account for restarts shows the theorem. □

We note that the total number of edge traversals used by *Balance* is also $O(\min\{dn^2 + m, nm\})$, where n is the number of nodes in the graph. For the $O(dn^2 + m)$ bound observe that the robot gets stuck at every node at most d times, i.e., it gets stuck a total of $O(dn)$ times. In each relocation step it traverses at most n edges. A similar argument shows the $O(nm)$ bound.

4 A tight lower bound for the *Balance* algorithm and modifications

In this section we give first a lower bound for the *Balance* algorithm and afterwards we give lower bounds for modifications of *Balance*.

Theorem 3 *For every $d \geq 1$, there exists a graph G of deficiency d that is explored by Balance using $d^{\Omega(\log d)} m$ edge traversals.*

Proof: We show that there exists a graph $G = (V, E)$ and an edge $e \in E$ that is traversed $d^{\Omega(\log n)}$ times while *Balance* explores G. The theorem follows by replacing e by a path of $\Theta(m)$ edges.

The graph is a union of chains C, each of which consists of three edges, a startnode, an endnode and two *interior* nodes $v^1(C)$ and $v^2(C)$. The interior nodes belong to exactly one chain and have up to one outgoing edge. We describe G, see also Figure 5. Graph G contains (a) a cycle C_0 that starts and ends in a node v (*Balance* is started at v and finds C_0 during Phase 1) and (b) a recursively defined problem P^d attached to C_0.

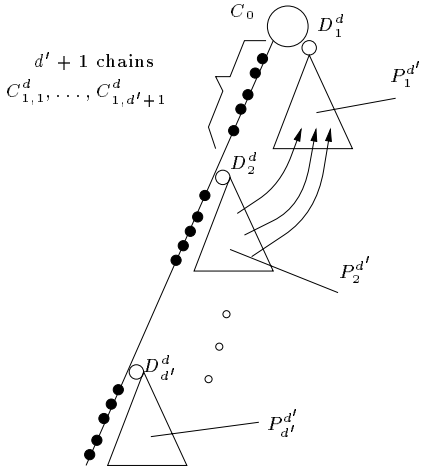

Figure 5: The graph G

A *problem P^δ of deficiency δ*, for any integer $\delta \geq 6$, is a subgraph that has two *incoming* edges whose startnodes do not belong to P^δ but whose endnodes do, and δ *outgoing* edges whose startnode belongs to P^δ but whose endnodes do not. A problem P^δ, with $\delta \in \{2, \ldots, 5\}$ has two incoming and two outgoing edges; a problem P^1 has one incoming and one outgoing edge. In the case of P^d, the two incoming edges start at $v^1(C_0)$ and $v^2(C_0)$, respectively; the d outgoing edges all point to v.

A problem P^1 consists of a single chain and, for $\delta \in \{2, \ldots, 5\}$, P^δ consists of two chains. For $\delta \in \{1, \ldots, 5\}$, the first edge of each chain in P^δ represents an incoming edge and the last edge represents an outgoing edge. The interior nodes of the chain have no outgoing edges. For $\delta \geq 6$, let $\delta' = \lfloor \delta/3 \rfloor - 1$. Problem P^δ consists of $\delta'(\delta'+1)$ chains $C^\delta_{j,k}$, $1 \leq j \leq \delta'$, $1 \leq k \leq \delta'+1$, as well as δ' chains D^δ_j and δ' recursive subproblems $P^{\delta'}_j$, $1 \leq j \leq \delta'$.

These components are assembled as follows. One of the incoming edges of P^δ is the first edge of $C^\delta_{1,1}$. Node

$v^1(C^\delta_{j,k})$ is the startnode of $C^\delta_{j,k+1}$, $1 \le j,k \le \delta'$. Node $v^1(C^\delta_{j,\delta'+1})$ is the startnode of $C^\delta_{j+1,1}$, $1 \le j \le \delta'-1$. The last edge of $C^\delta_{1,k}$, $1 \le k \le \delta'+1$, is an outgoing edge of P^δ. The endnode of $C^\delta_{j,k}$ is equal to the startnode of $C^\delta_{j-1,k}$, $2 \le j \le \delta'$ and $1 \le k \le \delta'+1$. Nodes $v^2(C^\delta_{j,k})$, $1 \le j,k \le \delta'$, have no outgoing edge but nodes $v^2(C^\delta_{j,\delta'+1})$, $1 \le j \le \delta'-1$, do. Chain $C^\delta_{\delta',\delta'+1}$ has no outgoing edges.

The second incoming edge of P^δ is the first edge of a chain D^δ_1 and, for $2 \le j \le \delta'$, the edge leaving $v^2(C^\delta_{j-1,\delta'+1})$ is the first edge of D^δ_j. For $1 \le j \le \delta'$, the last edge of D^δ_j is an outgoing edge of P^δ. The two edges leaving the interior nodes of the chain point into a subproblem $P^{\delta'}_j$. All outgoing edges of $P^{\delta'}_1$ are edges that also leave P^δ. *Balance* colors each $P^{\delta'}_j$, $2 \le j \le \delta'$, in the same way as $P^{\delta'}_1$, but the outgoing edges of $P^{\delta'}_j$ point into $P^{\delta'}_{j-1}$.

We identify the sources of G, i.e. the nodes having higher indegree than outdegree. At each source, indegree and outdegree differ by 1. The startnodes of the chains D^d_j, $1 \le j \le d'$, and $C^d_{d',k}$, $1 \le k \le d'+1$, represent a total of $2d'+1$ sources. Finally, the subproblem $P^{d'}_{d'}$ of deficiency d' attached to $D^d_{d'}$ contains d' sources.

We analyze the number of edge traversals used by *Balance* on G. Consider a problem P^δ, $6 \le \delta \le d$, and let $\delta' = \lfloor \delta/3 \rfloor - 1$. Suppose that *Balance* has just generated $C^\delta_{j,l'+1}$, for some $1 \le j \le \delta'$. Since the strand of chains $C^\delta_{j,1}, \ldots, C^\delta_{j,\delta'+1}$ contains $\delta'+1$ tokens, *Balance* does not explore the unvisited edges out of $C^\delta_{j,\delta'+1}$ before the subproblem $P^{\delta'}_j$ attached to D^δ_j is finished.

Let $N(\delta)$ be the number of times the following event happens while *Balance* works on a problem P^δ: *Balance* generates a new chain, gets stuck and cannot reach a node with new outgoing edges by using only edges in P^δ. Problem P^δ contains δ' subproblems $P^{\delta'}_1, \ldots, P^{\delta'}_{\delta'}$. Every time *Balance* gets stuck in a subproblem $P^{\delta'}_j$, $1 \le j \le \delta'$, and has to leave $P^{\delta'}_j$ in order to resume exploration, it also has to leave P^δ. This is because of the following facts: (1) When *Balance* explores $P^{\delta'}_j$, the subproblems $P^{\delta'}_1, \ldots, P^{\delta'}_{j-1}$ are already finished. (2) The chains $D^\delta_1, \ldots, D^\delta_j$ ensure that *Balance* cannot reach any chain $C^\delta_{j,k}$, $1 \le j \le \delta$, $1 \le k \le \delta'+1$, from where the unfinished chains in P^δ can be reached. Thus, for $d \ge \delta \ge 6$, $N(\delta) \ge \delta' N(\delta') = (\lfloor \delta/3 \rfloor - 1) N(\lfloor \delta/3 \rfloor - 1)$. Since $N(\delta) \ge 1$, for $1 \le \delta \le 5$, we obtain $N(d) = d^{\Omega(\log d)}$. Finally, consider the edge e on C_0 that leaves v. *Balance* must traverse e at least $N(d) = d^{\Omega(\log d)}$ times. □

We also modified the *Balance* algorithm by relocating to other nodes with new outgoing edges. Replace the choice of C_k in line 7 of by one of the following rules.

Round Robin: Let C_k be the chain among $C_1, \ldots, C_l$ that was selected least often in any execution of line 7.

Cheapest Subtree: Let C_k be the chain among $C_1, \ldots, C_l$, such that $T_{v(C_k)}$ contains the fewest number of dependent chains with respect to the current chain.

Theorem 4 *For Round Robin and for Cheapest Subtree and for all $d \ge 1$, there exist graphs of deficiency d that require $d^{\Omega(\log d)}m$ edge traversals.*

The proof is omitted.

References

[1] B. Awerbuch, M. Betke, R. Rivest and M. Singh. Piecemeal Graph Learning by a Mobile Robot. *Proc. 8th Conf. on Comput. Learning Theory*, 321–328, 1995.

[2] E. Bar-Eli, P. Berman, A. Fiat and R. Yan. On-line navigation in a room. *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, 237–249, 1992.

[3] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosén and M. Saks. Randomized robot navigation algorithms. *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, 74–84, 1996.

[4] A. Blum and P. Chalasani. An on-line algorithm for improving performance in navigation. *Proc. 34th Symp. on Foundations of Computer Science*, 2–11, 1994.

[5] A. Blum, P. Raghavan and B. Schieber. Navigating in unfamiliar geometric terrain. *Proc. 23rd Symp. on Theory of Computing*, 494–504, 1991.

[6] M. Betke, R. Rivest and M. Singh. Piecemeal learning of an unknown environment. *Proc. 5th Conf. on Comput. Learning Theory*, 277–286, 1993.

[7] M. Bender and D. Slonim. The power of teach exploration: two robots can learn unlabeled directed graphs. *Proc. 35th Symp. on Foundations of Computer Science*, 75–85, 1994.

[8] X. Deng, T. Kameda and C. H. Papadimitriou. How to learn an unknown environment. *Proc. 32nd Symp. on Foundations of Computer Science*, 298–303, 1991.

[9] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Proc. 31st Symp. on Foundations of Computer Science*, 356–361, 1990.

[10] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. Revised version of [9].

[11] F. Hoffmann, C. Icking, R. Klein and K. Kriegel. A competitive strategy for learning a polygon. *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, 166–174, 1997.

[12] E. Koutsoupias. Result reported in [10].

[13] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[14] R. Rivest. Problem formulation cited in [9].