# EXPLORING UNKNOWN ENVIRONMENTS[*]

SUSANNE ALBERS[†] AND 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 [16] gave a lower bound for $R$ of $\Omega(d^2m)$, and Deng and Papadimitriou [12] showed an upper bound of $d^{O(d)}m$, where $m$ is the number of 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.

**Key words.** directed graph, exploration algorithm

**AMS subject classifications.** 05C20, 68Q20, 68Q25, 68R10

## 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 [12] 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 [12], where $m$ is the number of edges. This immediately implies that undirected graphs can be explored with at most $4m$ traversals. In fact, using depth-first-search they can be explored using $2m$ edge 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 [12] 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 [16] improved the lower bound to $\Omega(d^2m)$.

[*]A preliminary version of this paper was presented at the 29th Annual ACM Symposium on Theory of Computing (STOC), 1997.

[†]Max-Planck-Institut für Informatik, Im Stadtwald, D-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, Compaq Computer Corporation, 130 Lytton Ave, Palo Alto, CA 94301. Email: monika@pa.dec.com. This work was supported by the NSF CAREER Award, Grant No. CCR-9501712.

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 problem proposed by Rivest [19]. Betke *et al.* [8] 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 [9] 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 [12], Deng *et al.* [11] investigated a geometric exploration problem, whose goal is to explore a room with or without polygonal obstacles. Hoffmann *et al.* [15] 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 [18]. 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.* [7] investigated the problem of finding a shortest path in an unfamiliar terrain with convex obstacles. More work on this problem includes [2, 5, 6].

**Our Results.** Our main result is a new robot strategy that explores an arbitrary graph with deficiency $d$ and traverses each edge at most $(d+1)^7 d^{2\log d}$ times, see Section 3. The algorithm does not need to know $d$ in advance. The total number of traversals needed by the algorithm is also $O(\min\{nm, dn^2+m\})$, where $n$ is the number of nodes. At the end of Section 3 we show that any exploration algorithm that fulfills two intuitive conditions achieves an upper bound of $O(\min\{nm, dn^2+m\})$. A depth-first search strategy obtaining this bound was independently developed by Kwek [17].

In Section 4 we demonstrate that our analysis of the new robot strategy 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 present 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 prove a lower bound of $2^{\Omega(d)}m$ for a locally greedy, a depth-first and a breadth-first algorithm. We also give a lower bound of $d^{\Omega(\log d)}m$ for a generalized greedy strategy.

A related problem for which lower bounds have been studied extensively, is the *s–t connectivity problem* in directed graphs, see [3, 4, 14] and references therein. Given a directed graph, the problem is to decide whether there exists a path from a distinguished node $s$ to a distinguished node $t$. Most of the results are developed in the JAG model by Cook and Rackoff [10]. The best time–space tradeoffs currently known [4, 14] only imply a polynomial lower bound on the computation time if no upper bounds are imposed in the space used by the computation. Given the current knowledge of the $s$–$t$ connectivity problem it seems unlikely that one can prove super-polynomial lower bounds for a *general* class of graph exploration algorithms.

In the following 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 2.1. *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.*

*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* $P^\delta$, for any integer $\delta \geq 2$, is a subgraph that has two *incoming* edges whose startnodes do not belong to $P^\delta$ but whose endnodes do, and $\delta$ *outgoing* edges whose startnode belongs to $P^\delta$ but whose endnodes do not. A *problem* $P^1$ is defined in the same way as a problem $P^\delta$, $\delta \geq 2$, 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 $v$.

For the description of $P^\delta$ we also need recursively defined problems $Q^\delta$. These problems are identical to $P^\delta$ except that, for $\delta > 2$, $Q^\delta$ has exactly $\delta$ incoming edges.

A problem $P^\delta$, $\delta = 1, 2$, consists of $\delta$ chains of three edges each. The first edge of each chain is an incoming edge into $P^\delta$; the last edge of each chain is an outgoing edge. A problem $Q^\delta$, $\delta = 1, 2$, is the same as $P^\delta$.

We proceed to define $P^\delta$, for $\delta > 2$. One of the incoming edges of $P^\delta$ is the first edge of a chain $D^\delta$ consisting of three edges, the other incoming edge is the first edge of a long chain $C^\delta$. For each of these chains $C^\delta$ and $D^\delta$, the last edge is an outgoing edge of $P^\delta$. If $\delta = 3$, the last interior node of each of the chains $C^\delta$ and $D^\delta$ has an additional outgoing edge pointing into a problem $P^1$. If $\delta \geq 4$, (a) the last two interior nodes of $C^\delta$ each have an additional outgoing edge pointing into a subproblem $P^{\delta-2}$, (b) the last two interior nodes of $D^\delta$ each have an additional outgoing edge pointing into a subproblem $Q^{\delta-2}$. There are $\delta - 2$ edges leaving $P^{\delta-2}$, exactly $\max\{0, \delta - 4\}$ of which point to nodes of $Q^{\delta-2}$ such that each node in $Q^{\delta-2}$ that has $k$ more outgoing than incoming edges, for some $0 \leq k \leq \max\{0, \delta - 4\}$, receives $k$ incoming edges from $P^{\delta-2}$. The remaining outgoing edges of $P^{\delta-2}$ point to the interior nodes of $D^\delta$ that have additional outgoing edges. The problem $Q^{\delta-2}$ has $\delta - 2$ outgoing edges all of which are outgoing edges of $P^\delta$. The total number of edges in $C^\delta$ is 2 plus the number of edges of $D^\delta$ plus the total number of edges contained in the subproblem $Q^{\delta-2}$ below $D^\delta$.

A problem $Q^\delta$, $\delta > 2$, is the same as $P^\delta$ except that the subproblem $P^{\delta-2}$ is replaced by another $Q^{\delta-2}$ problem. That is, $Q^\delta$ is composed of chains $C^\delta$, $D^\delta$ and problems $Q_i^{\delta-2}$, $i = 1, 2$. As mentioned before, $Q^\delta$ has exactly $\delta$ incoming edges.

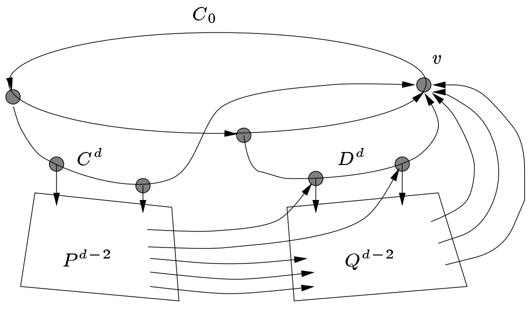

FIG. 2.1. *The graph for Greedy*

*Greedy* is started at node $v$ and traverses first chain $C_0$. Then it either explores $C^d$ or $D^d$. In either case, afterwards *Greedy* explores all edges of $Q^{d-2}$ since $C^d$ is prohibitively long. Thus, $P^{d-2}$ 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 $P^\delta$ or $Q^\delta$, gets stuck at some node and cannot relocate to a suitable node by using only edges in $P^\delta$ resp. $Q^\delta$. 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 $P^\delta$ contains two subproblems $P^{\delta-2}$ and $Q^{\delta-2}$. Note (a) that, because of chain $D^\delta$, no node in $Q^{\delta-2}$ can reach a node of $P^{\delta-2}$ without leaving $P^\delta$. Note (b) that $Q^{\delta-2}$ is completely explored when the exploration of $P^{\delta-2}$ starts and all paths starting in $P^{\delta-2}$ lead through $D^\delta$ or $Q^{\delta-2}$. Thus, every time *Greedy* gets stuck in a subproblem $P^{\delta-2}$ or $Q^{\delta-2}$ and has to leave $P^{\delta-2}$ resp. $Q^{\delta-2}$ in order to resume exploration, it also has to leave $P^\delta$. For $Q^{\delta-2}$ the statement follows from (a); for $P^{\delta-2}$ it follows from (a) and (b). In the same way we can argue for a problem $Q^\delta$. 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 $Q^{d-2}$ before it will start exploring $P^{d-2}$.

*Breadth-First:* Again we can use the same graph as in the lower bound for *Greedy.* The last two interior nodes of $C^d$ have a larger distance from the initial node $v$ than all nodes on $D^d$ and in $Q^{d-2}$. Thus $Q^{d-2}$ is finished before *Breadth-First* starts exploring $P^{d-2}$. □

THEOREM 2.2. *For Generalized-Greedy and for every $d$, there exists a graph of deficiency $d$ that requires $d^{\Omega(\log d)}m$ edge traversals.*

*Proof.* The graph used for the lower bound is outlined in Figure 2.2. The basic idea in the lower bound construction is as follows. *Generalized-Greedy* explores each subgraph $Q_i^\gamma$ and its sibling $R_i^\gamma$ "in parallel". Without loss of generality we can assume that the last chain traversed in the two subgraphs lies in $Q_i^\gamma$ and the algorithm continues to explore $Q_{i+1}^\gamma$ and $R_{i+1}^\gamma$. Let $N(\gamma)$ denote the number of times that the algorithm has to leave $R_i^\gamma$ and traverse the root. We will show that $N(4\gamma) \geq \gamma N(\gamma)$, which implies that the root has to be traversed $N(d) \geq d^{\Omega(\log d)}$ times.

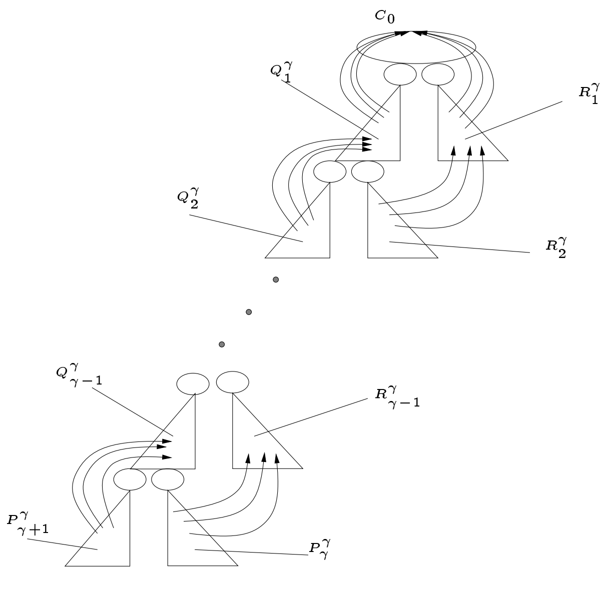

FIG. 2.2. *The graph for Generalized-Greedy*

To be precise we show the bound for $d$ being a power of 4. The bound for all values of $d$ follows by "rounding" down to the largest power of 4 smaller than $d$. The graph $G$ consists of two parts, (1) a cycle $C_0$ with nodes $v$, $v^1(C_0)$ and $v^2(C_0)$, and (2) a recursively defined subproblem $P^d$. Problem $P^d$ has two incoming edges, one starting at $v^1(C_0)$ and one starting at $v^2(C_0)$. It also has $d$ outgoing edges, all pointing to $v$. The subproblem $P^d$ 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 have at most one additional outgoing edge. We proceed to define $P^\delta$ and the "sibling" graphs $Q^\delta$ and $R^\delta$, for all $\delta \leq d$ that are a power of 4, and then show the lower bound on this graph.

A *problem* $P^\delta$, $\delta > 1$, is a graph with two incoming edges and exactly $\delta$ outgoing edges. A *problem* $R^\delta$, $\delta > 1$, consists of $P^\delta$ with $\delta - 2$ additional incoming edges. The *problem* $Q^\delta$ consists of $R^\delta$ with two additional incoming and two additional outgoing edges.

$\delta = 1$: A problem $P^1$ consists of one chain. The incoming edge of $P^1$ is the first edge of the chain, and the outgoing edge of $P^1$ is the last edge of the chain. In $P^1$, the interior nodes of the chain have no additional outgoing edges, in $Q^1$ each interior node has one additional incoming and one additional outgoing edge. Problem $R^1$ is equal to $P^1$.

$\delta = 4$: A problem $P^4$ consists of two subproblems $P^1_1$ and $P^1_2$, and chains $C^1_1$ and $D^1_1$, whose first interior nodes have one additional outgoing edge. The outgoing edge of $C^1_1$ is the incoming edge of $P^1_1$ and the corresponding edge of $D^1_1$ is the incoming edge of $P^1_2$. The last edge of $C^1_1$ and of $D^1_1$ and the outgoing edges of $P^1_1$ and $P^1_2$ are outgoing edges of $P^4$. A problem $R^4$ is $P^4$ with two additional incoming edges, one at the startnode of $P^1_1$ and one at the startnode of $P^1_2$. A problem $Q^4$ is $R^4$ with two additional incoming and outgoing edges; each interior node of $P^1_1$ has an additional incoming and outgoing edge.

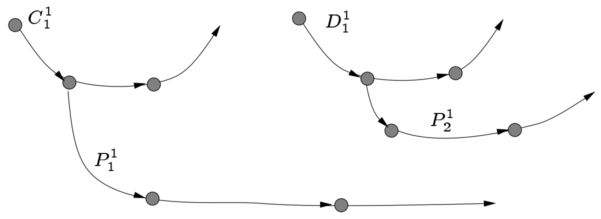

FIG. 2.3. *The subproblem* $P^4$

$\delta = 4^l$, for some $l \geq 2$: Let $\gamma = \delta/4$. It is simpler to describe $Q^\delta$ first. The construction is depicted in Figure 2.4. Every node has the same indegree as outdegree, i.e., there are no sinks. Problem $Q^\delta$ consists of subproblems $Q^\gamma_i$ and $R^\gamma_i$, for $1 \leq i \leq \gamma$, connected by chains $C^\gamma_i$ and $D^\gamma_i$, for $1 \leq i \leq \gamma$, whose interior nodes each have an additional outgoing edge.

The $C$-chains and $Q$-subproblems are interleaved as follows. The two edges leaving the interior nodes of $C^\gamma_1$ point into $Q^\gamma_1$. In general, the edges leaving the interior nodes of $C^\gamma_i$ point into $Q^\gamma_i$. The same holds for the $D$-chains and $R$-subproblems. The first edge of $C^\gamma_i$ and of $D^\gamma_i$ are incoming edges of $Q^\delta$, for $i = 1$, and start in $Q^\gamma_{i-1}$, for $1 < i \leq \gamma$, on a node of the leftmost subproblem $Q^1$ contained in $Q^\gamma_{i-1}$. Recall that this problem consists of one chain with two additional incoming and outgoing edges. One of these outgoing edges is the first edge of $C^\gamma_i$ and the second outgoing edge is the first edge of $D^\gamma_i$.

Additionally, the subproblems are connected as follows. Recall that $\gamma$ edges leave $R_i^\gamma$. For $i = 1$, the edges leaving $R_i^\gamma$ are outgoing edges of $Q^\delta$. For $1 < i \leq \gamma$, two edges leaving $R_i^\gamma$ point to the interior edges of $D_{i-1}^\gamma$. Additionally, there are $\gamma - 2$ edges leaving $R_i^\gamma$ and pointing into $R_{i-1}^\gamma$ such that every node in $R_{i-1}^\gamma$ that has $k$ more outgoing than incoming edges, for $k > 0$, receives $k$ edges from $R_i^\gamma$. The same holds for $Q_i^\gamma$ with $C_{i-1}^\gamma$. The problem $Q_\gamma^\gamma$ has $\gamma$ incoming edges which are incoming edges for $Q^\delta$, the problem $R_\gamma^\gamma$ has $\gamma - 2$ incoming edges which are incoming edges for $Q^\delta$.

There are $4\gamma + 2 = \delta + 2$ outgoing edges in $Q^\delta$: The last edge of $C_i^\gamma$ and the last edge of $D_i^\gamma$, for $1 \leq i \leq \gamma$, all edges leaving $R_1^\gamma$, all but two edges leaving $Q_1^\gamma$ (the other two are the incoming edges of $D_2^\gamma$ and $C_2^\gamma$), and two edges leaving $Q_\gamma^\gamma$. There are also $\delta + 2$ incoming edges: the first edge of $C_1^\gamma$ and of $D_1^\gamma$, the edges pointing to the two interior nodes of $C_\gamma^\gamma$ and $D_\gamma^\gamma$, the $\gamma$ incoming edges of $Q_\gamma^\gamma$, the $\gamma - 2$ incoming edges of $R_\gamma^\gamma$, and $2\gamma - 2$ incoming edges ending at the startnodes of $C_i^\gamma$ and $D_i^\gamma$, for $2 \leq i \leq \gamma$.

A problem $P^\delta$ consists of $2\gamma$ chains $C_i^\gamma$ and $D_i^\gamma$, $1 \leq i \leq \gamma$, as well as two subproblems $P_i^\gamma$, $\gamma \leq i \leq \gamma + 1$, and $2(\gamma - 1)$ subproblems $Q_i^\gamma$ and $R_i^\gamma$, $1 \leq i \leq \gamma - 1$. These components are assembled in the same way as in $Q^\delta$, except that $Q_\gamma^\gamma$ is replaced by $P_{\gamma+1}^\gamma$, and $R_\gamma^\gamma$ is replaced by $P_\gamma^\gamma$. Problems $P_\gamma^\gamma$ and $P_{\gamma+1}^\gamma$ each have only two incoming edges from $C_\gamma^\gamma$ and $D_\gamma^\gamma$, respectively.

There are $4\gamma = \delta$ outgoing edges in $P^\delta$: The last edge of $C_i^\gamma$ and the last edge of $D_i^\gamma$, for $1 \leq i \leq \gamma$, all but two edges leaving $Q_1^\gamma$ (the other two are the incoming edges of $D_2^\gamma$ and $C_2^\gamma$), all edges leaving $R_1^\gamma$. There are two incoming edges in $P^\delta$. The first edge of $C_1^\gamma$ and of $D_1^\gamma$ are incoming edges in every problem $P^\delta$. The following $\delta - 2$ nodes are sources for $P^\delta$: the two interior nodes of $C_\gamma^\gamma$ and of $D_\gamma^\gamma$, the $2\gamma - 2$ startnodes of $C_i^\gamma$ and $D_i^\gamma$, for $2 \leq i \leq \gamma$, the $\gamma - 2$ sources of $P_\gamma^\gamma$ and the $\gamma - 2$ sources of $P_{\gamma+1}^\gamma$.

A problem $R^\delta$ is a problem $P^\delta$ with an incoming edge into every source of $P^\delta$. Thus there are $\delta$ incoming and $\delta$ outgoing edges.

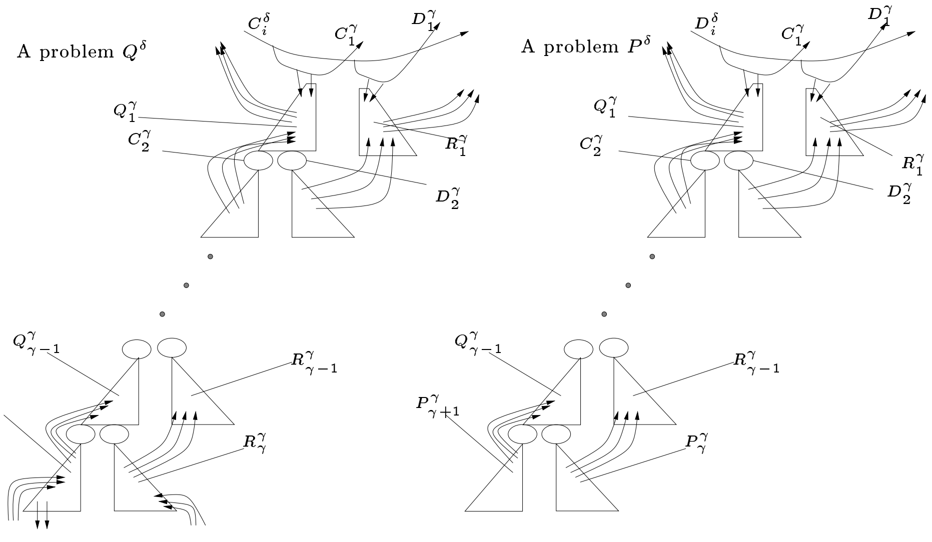

FIG. 2.4. *The subproblems $Q^\delta$ and $P^\delta$*

We analyze *Generalized-Greedy* on $G$. For simplicity we only discuss the exploration of a problem $Q^\delta$. The argument for $P^\delta$ and $R^\delta$ is analogous. As before, let $\gamma = \delta/4$. We show inductively that the symmetric construction of $Q_i^\gamma$ and $R_i^\gamma$ attached to $C_i^\gamma$ and $D_i^\gamma$ as well as the definition of *Generalized-Greedy* imply that $Q_i^\gamma$ and $R_i^\gamma$ are explored symmetrically. That is, during two consecutive traversals of $C$ (in order to resume exploration in $Q_i^\gamma$ or $R_i^\gamma$), *Generalized-Greedy* proceeds once into $Q_i^\gamma$ and once into $R_i^\gamma$, where $C$ is the chain at which chains $C_i^\gamma$ and $D_i^\gamma$ start. This obviously holds for $i = 1$. Assume it holds for $i$ and we want to show it for $i + 1$. Note that $Q_i^\gamma$ and $R_i^\gamma$ differ only in the last chain that *Generalized-Greedy* explores in $Q_i^\gamma$, rep. $R_i^\gamma$. Thus, until the traversal of the earlier of the last chain of $Q_i^\gamma$ and the last chain of $R_i^\gamma$, *Generalized-Greedy* does not distinguish $Q_i^\gamma$ from $R_i^\gamma$. Hence we can assume without loss of generality that *Generalized-Greedy* traverses first the last chain of $R_i^\gamma$ and afterwards the last chain of $Q_i^\gamma$. (Think of an adversary "giving" to *Generalized-Greedy* first the last chain of $R_i^\gamma$ and then the last chain of $Q_i^\gamma$.) Then *Generalized-Greedy* explores $C_{i+1}^\gamma$ and $D_{i+1}^\gamma$ and afterwards $Q_{i+1}^\gamma$ and $R_{i+1}^\gamma$ symmetrically. Thus, when *Generalized-Greedy* explores a subproblem $R_i^\gamma$, $1 \leq i \leq \gamma$, subproblems $R_j^\gamma$ with $1 \leq j < i$ are already finished.

Whenever *Generalized-Greedy* gets stuck in $R_i^\gamma$, $1 \leq i \leq \gamma$, and has to leave $R_i^\gamma$ in order to resume exploration, it also has to leave the "parent problem" $Q^\delta$ (or $P^\delta$, $R^\delta$). This is because the chains $D_i^\gamma$, $1 \leq i \leq \gamma$, prevent the algorithm from reaching a chain in $Q_j^\gamma$, $1 \leq j \leq i$, from where unfinished chains in $Q^\delta$, ($P^\delta$, $R^\delta$) can be reached. On the way from $R_i^\gamma$ to an outgoing edge of the parent problem, *Generalized-Greedy* can traverse problems $R_j^\gamma$, $j \leq i$. As shown above, the subproblems are finished, no further exploration of $R_j^\gamma$ is possible. The same arguments hold when the algorithm gets stuck in a problem $P_\gamma^\gamma$.

For any $\delta$, $4 \leq \delta \leq d$, let $N(\delta)$ be the number of times *Generalized-Greedy* generates a chain in $P^\delta$ or $R^\delta$, gets stuck and has to leave $P^\delta$ or $R^\delta$ in order to continue exploration. Then $N(\delta) \geq \gamma N(\gamma) = \delta/4 N(\delta/4)$. Since $N(1) \geq 1$, we have $N(d) \geq d^{\Omega(\log d)}$ and hence the edge leaving node $v$ is traversed $d^{\Omega(\log d)}$ times. □

## 3. An algorithm for graphs with deficiency $d$.

**3.1. The *Balance* 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 the problem can be reduced to the case described here. Details are given in Section 3.2.

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 [12] 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 $s$ [12]. 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 it is not finished and 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 is crucial. Every startnode of a fresh chain has more visited outgoing than 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* $\tau_i$, 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 $\tau_i$. 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. More specifically, the owner of $\tau_i$ is the chain that was current chain when the path from the current guess of $t_i$ to $s_i$ was extended last. Note that the owner is not the chain from the current guess of $t_i$ to $s_i$. 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 3.1. The robot returns to $x$ and the next phase has the same current node and current chain.
   **Case 2:** $y \neq x$, $y$ has a token $\tau_i$ and is the startnode of a fresh chain $D$ (see

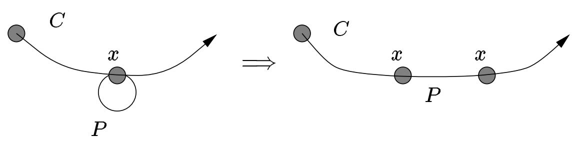

FIG. 3.1. *Case 1*

Figure 3.2)
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.

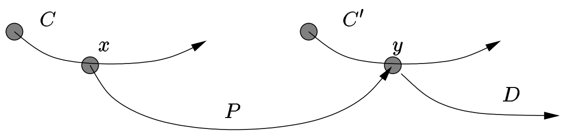

FIG. 3.2. *Case 2*

**Case 3:** $y \neq x$, $y$ has a token $\tau_i$ 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 $\tau_{i+1}$ 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 3.3. 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 1 and item 2, respectively. In line 13, $C'$ is the chain that was the previous owner of $\tau_i$ and becomes the new current chain.

Additionally, the algorithm maintains a tree $T$ such that each chain $C$ corresponds to a node $v(C)$ of $T$ and $v(C')$ 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 $v$. For, each chain, there is exactly one node in the tree. Note that the tree changes dynamically. If in line 10 of the algorithm, a path $P$ is appended at a chain $D$, then the node representing the resulting chain becomes a child of $v(C)$, i.e. a child of the node representing the current chain $C$. The node $v(D)$ is removed. Since only fresh chains are reassigned, each added or removed node is a leaf. This process ensures that the structure of nodes is indeed a tree.

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 $\tau$ 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

**Algorithm Balance**

```
 1.  $j := 0$, $k := 0$, $x := s$, $C := C_0$.
 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 $\tau_i$ then
 9.                      if $\exists$ chain $D$ of color $i$ starting in $y$ and $D$ is fresh then
10.                          Concatenate $P$ with $D$;
11.                      else
12.                          $j := j + 1$; $C_j :=$ chain that consists of $P$;
13.                      $C' := owner(\tau_i)$; Place $\tau_i$ 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 $\tau_k$ 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.
```

FIG. 3.3. *The Balance algorithm*

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 $i$. Intuitively, if a maximal subpath $R$ of $Q$ that does not belong to $T_u$ is preceded by an edge of color $i$, then $R$ is just the path of color $i$ that leads to the previous chain of color $i$ in $T_u$. In Figure 3.4 solid, dashed and dotted lines denote different colors. In the corresponding tree, the root $v(C_0)$ has two children, namely $v(C_1)$ and $v(C_5)$. Consider the path $Q$ that starts at $x$, follows the solid chains to $v$ and $w$ and then follows the dashed edges to $u$. (Path $Q$ is shown in bold.) Path $Q$ is a $C_2$-completion. It is also $T_{v(C_1)}$-homogenous because the two chains $C_3$ and $C_4$ not belonging to $T_{v(C_1)}$ have the same color as $C_1$ and $C_2$.

We try to choose $closure(C)$ to be "as local to $C$" as possible: Let $S(C)$ be the set

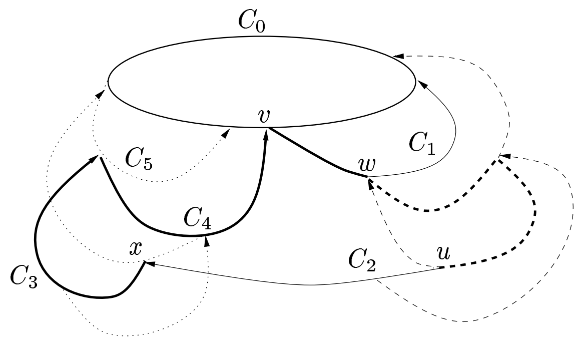

FIG. 3.4. *The path from $x$ to $u$ via $v$ and $w$ is $T_{v(C_1)}$-homogenous*

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.

We describe the *Relocation* procedure, see Figure 3.5. 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.

```
Procedure Relocate(C)
 1.  if all chains are finished then return(empty_chain).
 2.  else Move robot to the startnode of $C$ along $closure(C)$;
 3.      while $C \neq C_0$ and $T_{v(C)}$ is finished do
 4.          Move robot to the 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 $T_{v(C_1)}, \ldots, T_{v(C_l)}$ having 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($C$)
```

FIG. 3.5. *The Relocation procedure*

## 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 3.2 shows that if a chain of color $i$ is fresh, then $\tau_i$ lies at the startnode of the chain. Thus, in line 10, token $\tau_i$ 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 Propositon 3.2, part 4, shows later that *closure(C)* is not needed.

**3.2.2. Fundamental properties of the algorithm.**

LEMMA 3.1. *At most d tokens are introduced during the execution of the Balance algorithm.*

*Proof.* We say that the algorithm first introduces the token $\tau_k$ 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, $d = -\sum_{v, with\ bal(v)<0} bal(v) \geq \sum_v t(v)$.

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 introduced on $v$, i.e. when the algorithm for the first time gets stuck at $v$ and there is no token at $v$. 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 the $i$-th new token is introduced 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 3.2.

1. *For every chain $C$ that is in progress or that was in progress and is 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, $\tau_i$ is located at the startnode of $C$, and parent($C$) = owner($\tau_i$). (b) If $C$ is in progress and not the current chain, then $C$ is the owner of some token $\tau$.*
3. *Every chain $C$ is the parent of at most d chains.*
4. *If the Balance algorithm gets stuck at a node $y$ of a chain $C$ and $y$ holds a token with $C$ being the owner, then the startnode of $C$ and all nodes of $C$ lying between the startnode and $y$ are finished.*

*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*, $\tau_i$ is placed on the startnode of $C$. Whenever the robot gets stuck at the current startnode of $C$ and removes $\tau_i$, chain $C$ is extended by a path $P$ because $C$ is not in progress. Token $\tau_i$ is placed on the new startnode of $C$. Lines 13 and 16 ensure that the parent of $C$ is always the owner of $\tau_i$.

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. This suffices to prove part 2b because the children of a chain, and thus the corresponding tokens, can only be taken over by the current chain, see lines 13 and 16 of the algorithm.

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 $\mathit{parent}(C') = C$ is 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.

Part 4. Since $y$ holds a token, with $C$ being the owner, $y$ must have been the current node in a subphase when $C$ was current chain. The node selection rule in line 18 of *Balance* ensures that the startnode of $C$ and every node on $C$ between the startnode and $y$ are finished since, otherwise, the robot would have moved to an unfinished node $z$ before $y$. □

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 $\mathit{active}(T_v) = 1$ if the current chain is in $T_v$; otherwise let $\mathit{active}(T_v) = 0$.

LEMMA 3.3. *Let $u, v \in T$ be siblings in $T$ such that $T_u$ and $T_v$ contain unfinished chains. Then $|w(T_u) + \mathit{active}(T_u) - w(T_v) - \mathit{active}(T_v)| \leq 1$.*

*Proof.* Let $\mathit{active}(C) = 1$ iff $C$ is the current chain, and let $\mathit{active}(C) = 0$ otherwise. Let $\mathit{token}(C)$ be the number of tokens owned by $C$, and let $g(C) = \mathit{token}(C) + \mathit{active}(C)$. Finally, let $g(v) = \sum_{C, v(C) \in T_v} g(C) = w(T_v) + \mathit{active}(T_v)$. 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 $\mathit{active}(T_v)$ 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 3.2. Thus, the claim holds for any pair of children of $v(C)$.

We consider next all changes to $w(T_v)$ and $\mathit{active}(T_v)$.

*Line 13:* Let $C$ be the current chain before the execution of line 13. Note that $\mathit{token}(C)$ increases by 1, $\mathit{active}(C)$ becomes 0, $\mathit{token}(C')$ decreases by 1, and $\mathit{active}(C')$ 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 $\mathit{token}(C')$ and $\mathit{active}(C')$ 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 $\mathit{active}(\bar{C})$ drops to 0 and $\mathit{active}(C)$ 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 $\mathit{active}(C') = 0$, $|w(T_{v(C)}) - w(T_{v(C')}) + \mathit{active}(C) - \mathit{active}(C')| \leq 1$. □

LEMMA 3.4. *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 3.5. *Each edge is traversed at most $d$ times during executions of line 17 and at most $d+1$ 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 $C$.

By Proposition 3.2, part 4, $e$ is traversed in line 18 when all nodes on $C^1$ are finished and the robot moves to the next unfinished node on $C^2$. Thus, $e$ is traversed (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 $C^1$. Whenever the robot interrupts the work on $C^2$, another token is placed on some node of $C^2$. Every time case (b) occurs, $\mathit{token}(C)+\mathit{active}(C)$ increases by 1, while no other step of the algorithm can decrease this value as long as $C$ is unfinished. Note that a token is placed on a node of $C^2$. Since there are only $d$ tokens, cases (a) and (b) occur a total of at most $d+1$ times. $\square$

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

LEMMA 3.6. *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 $\delta$, $1 \le \delta \le d$, let

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

We will show that for any $\delta$, $1 \le \delta \le d$, 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 $\delta$. Consider a subtree $T_{v(C)}$ that contains at most $\delta$ tokens when $\mathit{active}(T_{v(C)}) = 1$. Out of all chains of color $i$ dependent on $C$, let $C'$ be the chain whose closure is computed last. We show that when the algorithm computes $\mathit{closure}(C')$, 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 3.4, $l \le d$. Suppose that node $u_j$, $1 \le j \le l$, 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 $\mathit{closure}(C')$ is computed, whenever $\mathit{active}(T_{v_{j,k}}) = 1$, then $w(T_{v_{j,k}}) \le \lfloor \delta/2 \rfloor$. Consider the point in time when $\mathit{closure}(C')$ is computed. Since $T_{v_{j,1}}$ contains $C'$, $T_{v_{j,1}}$ is unfinished. By Lemma 3.3, *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 $\mathit{active}(T_{v_{j,k}}) = 1$. 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 3.3, $w(T_{v_{j,k}})$ is up to this point in time at most $\lfloor \delta/2 \rfloor$ whenever $\mathit{active}(T_{v_{j,k}}) = 1$. We conclude that up to the time when $\mathit{closure}(C')$ 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 3.2. Since $T_{v(C)}$ never branches, it can contain at most one chain of color $i$ that is dependent on $C$. □

LEMMA 3.7. *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 3.7 follows immediately from Lemma 3.6.

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 than $C$. 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 3.8. *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 $C$.*

*Proof.* A chain $C'$ *needs* a chain $C$ if *closure*($C'$) uses edges of $C$ and $C'$ is *$u$-hard* if *closure*($C'$) is $T_u$-homogeneous, but not $T_v$-homogeneous for any child $v$ of $u$. For each chain $C'$ there exists a unique node $u$ of $T$ such that $C'$ is $u$-hard. If $C'$ is dependent on chain $C$, then $C'$ is $v(C)$-hard of $u$-hard for a true ancestor $u$ of $v(C)$. If $C'$ is $u$-hard and $v$ is a descendant of $u$ and an ancestor of $v(C')$, then $C'$ is dependent on $C(v)$. To prove the lemma it suffices to show the following two claims:

CLAIM 3.9. *There are at most $d^{2\log d+2}$ chains $C' \notin T_{v(C)}$ such that $C'$ needs $C$ and $C'$ is $u$-hard for some ancestor $u$ of $v(C)$.*

CLAIM 3.10. *There are at most $(d+1)d^{2\log d+2}$ chains $C' \notin T_{v(C)}$ such that $C'$ needs $C$ and $C'$ is $u$-hard for some node $u$ that is not an ancestor of $v(C)$.*

*Proof of Claim 3.9.* If $C'$ needs $C$, then $C'$ either does not yet exist or is unfinished when $C$ is taken into progress. Consider the point in time when $C$ is taken into progress. Let $u_1, u_2, \ldots, u_l$ be the ancestors of $v(C)$ in $T$ that fulfill the following conditions: Each node $u_j$ has a child $v_j$ such that (a) $T_{v_j}$ contains unfinished chains, and (b) $v(C) \notin T_{v_j}$. Thus, every chain that needs $C$ lies in one of the subtrees $T_{v_j}$. Note that $l \le d$, since by Proposition 3.2, every subtree that contains an unfinished chain not equal to the current chain must own a token. Assume $C'$ belongs to $T_{v_j}$. Since $u_j$ is the least common ancestor of $v(C)$ and $v(C')$, and $C'$ is $u$-hard for an ancestor $u$ of $v(C)$, $C'$ is dependent on $C(u_j)$. Since by Lemma 3.6 there are at most $d^{2\log d+1}$ chains that are dependent on $C(u_j)$, there can be at most $l \cdot d^{2\log d+1} \le d^{2\log d+2}$ chains $C' \notin T_{v(C)}$ that need $C$ and are $u$-hard for an ancestor of $v(C)$. □

*Proof of Claim 3.10.* Let $i$ be the color of $C$. Let us denote the concatenation of all chains of color $i$ as the *path of color $i$*. Note that the path of color $i$ introduces a linear order on the chains of color $i$. We say a chain $C$ *lies between* two other chains on the path of color $i$ if $C$ is not equal to one of the chains and lies between them in the linear order. We define first the nearest predecessor of a chain. Then we show (1) that for each chain $C' \notin T_{v(C)}$ that needs $C$ and is $u$-hard for some node $u$ that is not an ancestor of $v(C)$, there exists a chain $C_1$ of color $i$ such that

- $C$ lies on the path of color $i$ between $C_1$ and its nearest predecessor, and
- $C_1$ fulfills the conditions of Claim 1, i.e., $C'$ needs $C_1$ and $u$ is an ancestor of $v(C_1)$.

We show next (2) that there exist at most $d$ chains $C_1$ of color $i$ for which $C$ lies on the path of color $i$ between $C_1$ and its nearest predecessor. By Claim 1 and Lemma 3.7, for each $C_1$ there exist at most $(d+1)d^{2\log d+1}$ closures that are hard for an ancestor of $v(C_1)$. It follows that there are at most $d(d+1)\cdot d^{2\log d+1}$ chains $C'$ that need $C$ and are $u$-hard for some node $u$ that is not an ancestor of $v(C)$.

Consider the point in time when $C$ is taken into progress. Let $a(C)$ be the closest ancestor of $v(C)$ such that $T_{a(C)}$ contains a node of color $i$ that is not equal to $v(C)$. The *nearest predecessor* of $C$ is the chain $C' \neq C$ of color $i$ that was taken into progress most recently in $T_{a(C)}$.

(1) The closure of $C'$ introduces an order on the chains belonging to it. Let $C_1$ be the last chain of $T_u$ before $C$ on *closure*$(C')$ and let $C_2$ be the first chain of $T_u$ after $C$ on *closure*$(C')$, i.e. $C$ lies on the path of color-$i$ edges between $C_1$ and $C_2$. We show below that the path of color-$i$ edges between $C_1$ and $C_2$ is contained in the path of color-$i$ edges between $C_1$ and its nearest predecessor. This implies that $C$ lies on the path of color-$i$ edges between $C_1$ and its nearest predecessor and completes the proof of (1).

Since $T_u$ is a subtree that contains $C_1$ and $C_2$, i.e. $C_1$ and another chain of color $i$ that was taken into progress before $C_1$, $T_u$ also must contain the nearest predecessor of $C_1$. Following the path of color-$i$ edges from $C_1$, $C_2$ is the first chain of $T_u$ that is encountered. Thus, the color-$i$ path between $C_1$ and $C_2$ is contained in the color-$i$ path between $C_1$ and its nearest predecessor.

(2) We want to bound the number of color-$i$ chains $C_1$ such that $C$ lies on the path of color $i$ between $C_1$ and its nearest predecessor. Obviously, $C_1$ was created, after $C$ was taken in progress (otherwise, $C_1$ would have been appended to $C$). Consider the point in time when $C$ is taken into progress. Let $\overline{C}_1, \ldots, \overline{C}_l$ be the chains that are parents of fresh chains. All chains created afterwards must belong to $T_{v(C)}$ or to $T_{v(\overline{C}_1)}, \ldots, T_{v(\overline{C}_l)}$. Note (a) that for no color-$i$ chain in $T_{v(C)}$, $C$ can lie on the color-$i$ path between the chain and its nearest predecessor. Note (b) that for $k = 1, \ldots, l$, only for the color-$i$ chain $C^{(k)}$ in $T_{v(\overline{C}_k)}$ created first after $C$ was taken into progress, $C$ can lie between $C^{(k)}$ and its nearest predecessor. The nearest predecessor of every color-$i$ chain $D$ created later belongs to $T_{v(\overline{C}_k)}$ and was created after $C$. Thus, $C$ does not lie on the color-$i$ path between $D$ and its predecessor. Thus, at most $l$ chains exists such that $C$ lies on the color-$i$ path between the chain and its predecessor. By Proposition 3.2, $l \leq d$. ☐

THEOREM 3.11. *Using the Balance algorithm and assuming that when a new sink is discovered the subgraph of explored edges is strongly connected, the robot explores an unknown graph with deficiency $d$ and traverses each edge at most $(d+1)^5 d^{2\log d}$ 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 3.5, it can be traversed $2d+1$ times during executions of lines 17 and 18. By Lemmas 3.7 and 3.8, $e$ 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 3.2, 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

$$(3.1)\ 1 + 2d + 1 + d(d+1)(d^{2\log d+1} + (d+2)d^{2\log d+2}) + d(d+1) \leq (d+1)^5 d^{2\log d}$$

times. □

**3.3. The *Complete* algorithm.** In Subsections 3.1 and 3.2 we assumed that the subgraph of *explored* edges is strongly connected. We used this assumption only in line 16 of algorithm *Balance*. However, all that is needed in line 16 is that the algorithm "knows" a path from $y$ to $s$, i.e., the robot can reach $s$ from $y$. To achieve this we define a parametrized algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) as follows: Additional to $s$ and $C_0$ it receives as input a set $\mathcal{P}$ of paths between various nodes in the graph. It executes algorithm *Balance* as before except when the robot gets stuck at $y$ in line 16 and there is no path of explored edges from $y$ to $s$. If there exists a path $X$ from $y$ to $s$ consisting of (i) a (possibly empty) subpath of explored edges, followed by (ii) a path in $\mathcal{P}$, followed by (iii) another (possibly empty) subpath of explored edges, then a *fake* edge from $y$ to $s$ is added to the graph and traversed to reach $s$. Since the fake edge does not exist in the orginial graph the robot "simulates" traversing the fake edge by traversing $X$. The fake edge continues to exist (and might be traversed) in the graph until the end of algorithm *P-Balance*. We show below that at most $d-1$ fake edges are added during algorithm *P-Balance*.

We execute algorithm *P-Balance* repeatedly to construct an algorithm *Complete* that assumes only that the original graph is strongly connected and makes *no* assumption about the subgraph of explored edges. We call the edges traversed during execution $i \leq k$ of algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) *$k$-visited.*

We describe algorithm *Complete* in detail: Initially $\mathcal{P}$ is empty and Phase 1 (see Subsection 3.1) is executed to determine $s$ and $C_0$. Algorithm *Complete* then repeatedly executes algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) on the graph until *P-Balance* terminates or until while traversing path $P$ the robot gets stuck at a node $y$ in line 16 and cannot reach $s$. In the former case algorithm *Complete* terminates, in the latter case it adds to $\mathcal{P}$ a path of $k$-visited edges to $y$ from each node in the subgraph traversed during the current or an earlier execution of algorithm *P-Balance.* Next all fake edges are discarded, all edges are marked as unvisited and unexplored, and all nodes are marked as unexplored and unfinished. Then $s$ is set to $y$, the cycle $C_0$ is set to be the path between the first and the last occurrence of $y$ on $P$, and algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) is called.

Consider execution $k$ of algorithm *P-Balance*. A *$k$-path* is a concatenation of three paths $A_1$, $A_2$, and $A_3$ such that $A_1$ and $A_3$ are possibly empty subpaths of edges explored during execution $k$ and $A_2$ is a path of $\mathcal{P}$. Note that the concatenation of a $k$-path with edges explored during execution $k$ (either at the beginning or at the end of the $k$-path) results again in a $k$-path. Note further that each $k$-path consists of $k$-visited edges.

Lemma 3.12 shows that if *P-Balance* gets stuck at a node $y$ in line 16 and cannot reach $s$, then there exists a path of $k$-visited edges to $y$ from each node in the subgraph traversed during the current or an earlier execution of algorithm *P-Balance* and that $y$ appears at least twice on $P$. This proves that algorithm *Complete* is well-defined.

LEMMA 3.12. *If while traversing path $P$ during an execution of P-Balance($\mathcal{P}$, $s$, $C_0$) the robot gets stuck in line 16 at a node $y$ and cannot reach $s$ then*

1. *each node in the subgraph traversed during an earlier execution of algorithm P-Balance($\mathcal{P}$, $s$, $C_0$) can reach $y$ on a path of $k$-visited edges;*
2. *each node in the subgraph traversed during the current execution of algorithm P-Balance($\mathcal{P}$, $s$, $C_0$) can reach $y$ on a $k$-path;*
3. *$y$ is a newly discovered sink;*
4. *$y$ appears at least twice on $P$.*

*Proof. Part 1, 2, and 3:* We use induction on the number $k$ of calls to algorithm *P-Balance* to show the claim. Obviously the claim holds for $k = 0$. Consider next $k > 0$. Let $s_k$ be the sink newly discovered by execution $k$ of algorithm *P-Balance*. We show first that each node in the subgraph traversed during an earlier execution of algorithm *P-Balance* can reach $y$ on a path of $k$-visited edges. There exists a path of $k$-visited edges from $s_{k-1}$ to $y$, since execution $k$ started at $s_{k-1}$. Inductively each node in the subgraph traversed during an earlier execution can reach $s_{k-1}$ on a path of $(k-1)$-visited edges. Thus, by transitivity of the reachability relation and since all $(k-1)$-visited edges are also $k$-visited, each node in the subgraph traversed during an earlier execution of algorithm *P-Balance* can reach $y$ on a path of $k$-visited edges.

We show next that each node in the subgraph traversed during the current execution of algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) can reach $y$ on a $k$-path. Since $y$ is the last node on chain $P$ every node on $P$ can reach $y$ following $P$. Each other node in the subgraph explored during algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) belongs to a chain $Q \neq P$. We show by induction on the number of such chains $Q$ created during the current execution that all nodes on such a chain $Q$ can reach $s$ by a $k$-path. Since execution $k$ started at $s$, $s$ can reach $y$ on edges explored during execution $k$. It follows that each node in the subgraph traversed during algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) can reach $y$ on a $k$-path.

It remains to be shown that all nodes on a chain $Q \neq P$ created during the current execution can reach $s$ by a $k$-path. This holds trivially before any chain is created. Consider a path $P'$ that is part of $Q$. Then the endpoint $y'$ of $P'$ either belongs to an already existing chain or not. If $y'$ belongs to a chain created earlier then inductively $y'$ and, thus, all nodes on $P'$ can reach $s$ by a $k$-path. If $y'$ does not belong to a chain created earlier, then there exists a path in $\mathcal{P}$ from $y'$ to $s$ since $P' \neq P$. Thus there is a $k$-path from $y'$ to $s$. It follows that every node on $P'$ can reach $s$ by a $k$-path.

We are left with showing that $y = s_k$, i.e., that $y$ is a newly discovered sink. By the above proof, (a) if $y$ was visited by an earlier execution of algorithm *P-Balance* then there would exist a path from $y$ to $s$ in $\mathcal{P}$, and (b) if $y$ belonged to a chain $Q \neq P$ in the current execution of algorithm *P-Balance* then there would exist a $k$-path from $y$ to $s$. Thus, algorithm *P-Balance*($\mathcal{P}$, $s$, $C_0$) would have been able to reach $s$ from $y$.

It follows that $y$ was not visited before, i.e., that $y$ is a newly discovered sink.

*Part 4:* Each node has outdegree at least 1. By the proof of part 1, $y$ does not belong to a chain $Q \neq P$. Thus all of $y$'s outedges must belong to $P$, i.e. $y$ appeared at least twice on $P$. $\square$

Since there are only $d$ sinks in the graph, part 3 of the above lemma shows that at most $d$ executions of *P-Balance*($\mathcal{P}$, $s$, $C_0$) are made. Thus it follows that algorithm *Complete* terminates.

Now let us analyze the number of edge traversals. Algorithm *P-Balance* traverses the same path that algorithm *Balance* would have traversed on the graph consisting of the original graph and all fake edges. Since each fake edge connects two sinks, it does not change the deficiency of the graph. Thus, the previous analysis shows that each edge, including each fake edge, is traversed at most $(d+1)^5 d^{2\log d}$ times. The traversal of a fake edge corresponds to at most one traversal of every non-fake edge. We show below that there are at most $d-1$ fake edges. Thus the total number of traversals per edge is at most $(d-1)(d+1)^5 d^{2\log d}$ for each execution of algorithm *P-Balance.* Since there are at most $d$ such executions, each edge is traversed at most $(d-1)d(d+1)^5 d^{2\log d}$ times during algorithm *Complete.*

It remains to show that there are at most $d-1$ fake edges. Each fake edge in execution $k$ increases the number $in_v(s_i)$ of visited incoming edges for a sink $s_i$ with $i < k$ without increasing the number $out_v(s_i)$ of visited outgoing edges. Since over all sinks $s_i$, $i < k$, there are at most $d-1$ more incoming than outgoing edges into these sinks, there are at most $d-1$ fake edges created during execution $k$.

We summarize our main result.

THEOREM 3.13. *Using the Complete algorithm, the robot explores an unknown graph with deficiency d and traverses each edge at most* $(d+1)^7 d^{2\log d}$ *times.*

The total number of edge traversals used by our algorithm is also $O(\min\{mn, dn^2 + m\})$, where $n$ is the number of nodes in the graph. It is not hard to show that an upper bound of $O(\min\{mn, dn^2 + m\})$ is achieved by any exploration algorithm satisfying the following two properties: (1) When the robot gets stuck, it moves on a cycle-free path to some, i.e. arbitrary, node with new outgoing edges. (2) When the robot is not relocating, it always traverses new edges whenever possible.

We show that any exploration algorithm satisfying (1) and (2) gets stuck at most $\min\{m, dn\}$ times. The bound follows because, by Property (1), at most $n$ edges are traversed during each relocation. Obviously, a robot gets stuck at most $m$ times. For the proof of the second bound, let $in_u(v)$ and $out_u(v)$ be the number of unvisited incoming and unvisited outgoing edges of $v$, respectively. Let $def(v) = \max\{0, in_u(v) - out_u(v)\}$. We show inductively that $\sum_{v\in G} def(v) \leq d$. This implies that, for every node $v$, whenever the robot explores the last unvisited edge out of $v$, there are at most $d$ unvisited incoming edges at $v$. Thus the robot gets stuck at most $d$ times at any node $v$. Summing over all nodes in $G$ gives the desired bound of $dn$.

The inequality $\sum_{v\in G} def(v) \leq d$ holds intitially. The invariant is maintained whenever the robot relocates from a node $y$, where it got stuck, to some node $z$ with new outgoing edges because only visited edges are traversed. Whenever the robot starts a new exploration at a node $z$, visits a sequence of new edges and gets stuck at a node $x$, $def(z)$ increases by at most 1, $def(x)$ decreases by 1 while at no other node, the *def*-value changes.

## 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 4.1. *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. We show the bound for $d$ being a power of 5. The bound for all values of $d$ follows by "rounding" down to the largest power of 5 smaller than $d$.

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 additional outgoing edge. We describe $G$, see also Figure 4.1. 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 $C_0$.

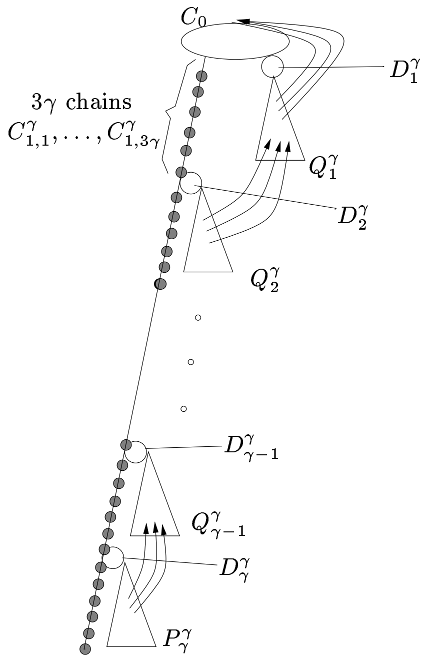

FIG. 4.1. *The graph G*

In the following let $\delta$, $1 \leq \delta \leq d$, be a power of 5. A *problem* $P^\delta$, for any integer $\delta \geq 5$, is a subgraph that has two *incoming* edges whose startnodes do not belong to $P^\delta$ but whose endnodes do, and $\delta + 1$ *outgoing* edges whose startnodes belong to $P^\delta$ but whose endnodes do not. 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; $d$ outgoing edges point to $v$ and one outgoing edge points to $v^1(C_0)$.

For the definition of $P^\delta$ we also need problems $Q^\delta$. These problems are identical to $P^\delta$ except that, for $\delta > 1$, $Q^\delta$ has exactly $\delta + 1$ incoming edges.

A problem $P^1$ consists of a single chain; the first edge of the chain represents an incoming edge and the last edge represents an outgoing edge. The interior nodes have no additional outgoing edges. A problem $Q^1$ is identical to $P^1$.

For $\delta \geq 5$, let $\gamma = \delta/5$. Problem $P^\delta$ consists of $3\gamma^2$ chains $C^\gamma_{i,k}$, $1 \leq i \leq \gamma$,

$1 \leq k \leq 3\gamma$, as well as $\gamma$ chains $D_i^\gamma$ and $\gamma$ recursive subproblems $Q_i^\gamma$, $1 \leq i \leq \gamma-1$, and $P_\gamma^\gamma$.

These components are assembled as follows. One of the incoming edges of $P^\delta$ is the first edge of $C_{1,1}^\gamma$. We assume that $v^1(C_0)$ is the startnode of $C_{1,1}^{d/5}$. Node $v^1(C_{i,k}^\gamma)$ is the startnode of $C_{i,k+1}^\gamma$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma-1$. Node $v^1(C_{i,3\gamma}^\gamma)$ is the startnode of $C_{i+1,1}^\gamma$, $1 \leq i \leq \gamma-1$. The last edge of $C_{1,k}^\gamma$, $1 \leq k \leq 3\gamma$, is an outgoing edge of $P^\delta$. The endnode of $C_{i,k}^\gamma$ is equal to the startnode of $C_{i-1,k}^\gamma$, $2 \leq i \leq \gamma$ and $1 \leq k \leq 3\gamma$. Note that the last edge of $C_{2,1}^\gamma$ hence is an outgoing edge of $P^\delta$. Nodes $v^2(C_{i,k}^\gamma)$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma-1$, have no additional outgoing edge but nodes $v^2(C_{i,3\gamma}^\gamma)$, $1 \leq i \leq \gamma-1$, do. Chain $C_{\gamma,3\gamma}^\gamma$ has no additional outgoing edges.

The second incoming edge of $P^\delta$ is the first edge of a chain $D_1^\gamma$ and, for $2 \leq i \leq \gamma$, the edge leaving $v^2(C_{i-1,3\gamma}^\gamma)$ is the first edge of $D_i^\gamma$. For $1 \leq i \leq \gamma$, the last edge of $D_i^\gamma$ is an outgoing edge of $P^\delta$. If $\delta = 5$, then the first interior node of the chain $D_i^\gamma = D_1^\gamma$ has an additional outgoing edge pointing into a problem $P^1$. If $\delta > 5$, then the two interior nodes of $D_i^\gamma$, $1 \leq i \leq \gamma$, each have an additional outgoing edge. For $1 \leq i \leq \gamma-1$, these two edges point into $Q_i^\gamma$ and, for $i = \gamma$, they point into $P_\gamma^\gamma$.

If $\delta = 5$, then the outgoing edge of the only subproblem $P^1$ is an outgoing edge of $P^\delta = P^5$. If $\delta > 5$, the problems $Q_i^\gamma$, $1 \leq i \leq \gamma-1$, and $P_\gamma^\gamma$ each have $\gamma+1$ outgoing edges. For $Q_1^\gamma$, $\gamma$ of these edges are also outgoing edges of $P^\delta$ and one edge points to the interior node of $D_1^\gamma$ that is the startnode of $C_{1,1}^\gamma$. For $2 \leq i \leq \gamma-1$, exactly $\gamma-1$ edges leaving $Q_i^\gamma$ point into $Q_{i-1}^\gamma$ such that every node that has $l$ more outgoing than incoming edges, for $l > 0$, receives $l$ edges. One outgoing edge points to the interior nodes of $D_{i-1}^\delta$ that does not get an edge from $Q_{i-1}^\gamma$ and the remaining edge points to the interior node of $D_i^\gamma$ that is the startnode of $C_{1,1}^\gamma$. In the same way the edges leaving $P_\gamma^\gamma$ are connected with $Q_{\gamma-1}^\gamma$, $D_{\gamma-1}^\gamma$ and $D_\gamma^\gamma$.

We identify the sources of $P^\delta$, i.e. the nodes having higher outdegree than indegree. At each source, outdegree and indegree differ by 1. The startnodes of the chains $D_i^\gamma$, $2 \leq i \leq \gamma$, and $C_{\gamma,k}^\gamma$, $1 \leq k \leq 3\gamma$, represent a total of $4\gamma-1$ sources. One interior node of $D_\gamma^\gamma$ represents a source. Finally, the subproblem $P_\gamma^\gamma$ contains $\gamma-1$ sources.

A problem $Q^\delta$, $\delta \geq 5$, is the same as $P^\delta$, except that the subproblem $P_\gamma^\gamma$ is replaced by a problem $Q_\gamma^\gamma$. As mentioned before, a problem $Q^\delta$ receives $\delta-1$ additional incoming edges. These edges point to the nodes that represent sources in $P^\delta$.

We analyze the number of edge traversals used by *Balance* on $G$. Consider a problem $P^\delta$, $\delta \geq 5$, and let $\gamma = \delta/5$. When *Balance* generates the strand of chains $C_{i,1}^\gamma, \ldots, C_{i,3\gamma}^\gamma$, for some $1 \leq i \leq \gamma$, this strand contains $3\gamma > \gamma+1$ tokens. Since $D_i^\gamma$ and the subproblem attached to it contain $\gamma$ tokens *Balance* does not explore the unvisited edges out of $C_{i,3\gamma}^\gamma$ before the subproblem attached to $D_i^\gamma$ is finished. In the same way we can argue for a problem $Q^\delta$.

Let $N(\delta)$ be the number of times the following event happens while *Balance* works on a problem $P^\delta$ or $Q^\delta$: *Balance* generates a new chain, gets stuck and cannot reach a node with new outgoing edges by using only edges in $P^\delta$ resp. $Q^\delta$. Problem $P^\delta$ contains $\gamma$ subproblems $Q_1^\gamma, \ldots, Q_{\gamma-1}^\gamma$ and $P_\gamma^\gamma$. Every time *Balance* gets stuck in one of these subproblems and has to leave it in order to resume exploration, it also has to leave $P^\delta$. This is because of the following facts: (1) When *Balance* explores $Q_i^\gamma$, $1 \leq i \leq \gamma-1$, or $P_\gamma^\gamma$, the subproblems $Q_1^\gamma, \ldots, Q_{i-1}^\gamma$ resp. $Q_1^\gamma, \ldots, Q_{\gamma-1}^\gamma$ are already finished. (2) The chains $D_1^\gamma, \ldots, D_\gamma^\gamma$ ensure that *Balance* cannot reach any chain $C_{i,k}^\gamma$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma$, from where the unfinished chains in $P^\delta$ can be reached. Again the same holds for a problem $Q^\delta$. Thus, for $\delta \geq 5$, $N(\delta) \geq \gamma N(\gamma) = (\delta/5)N(\delta/5)$. Since $N(\delta) = 1$, for $\delta = 1$, 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. $\square$

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

*Proof.* The proof is identical to that of *Generalized-Greedy* in Theorem 2.2. $\square$

**Acknowledgments.** We thank Prabhakar Raghavan for bringing to our attention the literature on the $s$–$t$ connectivity problem. We also thank an anonymous referee for may helpful comments improving the presentation of the paper.

## REFERENCES

[1] B. AWERBUCH, M. BETKE, R. RIVEST AND M. SINGH, *Piecemeal Graph Learning by a Mobile Robot*, in Proc. 8th Conf. on Comput. Learning Theory, 1995, pp. 321–328.

[2] E. BAR-ELI, P. BERMAN, A. FIAT AND R. YAN, *On-line navigation in a room*, J. Algorithms, 17 (1994), pp. 319–341.

[3] G. BARNES, J.F. BUSS, W.L. RUZZO AND B. SCHIEBER, *A sublinear space, polynomial time algorithms for directed s–t connectivity*, in Proc. 7th Annual Conf. on Structure in Compexity Theory, 1992, pp. 27–33.

[4] G. BARNES AND J. EDMONDS, *Time-space lower bounds for directed s–t connectivity on graph automata models*, SIAM J. Comput., 27 (1998), pp. 1190–1202.

[5] P. BERMAN, A. BLUM, A. FIAT, H. KARLOFF, A. ROSÉN AND M. SAKS, *Randomized robot navigation algorithms*, in Proc. 7th ACM-SIAM Symp. on Discrete Algorithms, 1996, pp. 74–84.

[6] A. BLUM AND P. CHALASANI, *An on-line algorithm for improving performance in navigation*, in Proc. 34th Symp. on Foundations of Computer Science, 1994, pp. 2–11.

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

[8] M. BETKE, R. RIVEST AND M. SINGH, *Piecemeal learning of an unknown environment*, Machine Learning, 18 (1995), pp. 231–254.

[9] M. BENDER AND D. SLONIM, *The power of team exploration: two robots can learn unlabeled directed graphs*, in Proc. 35th Symp. on Foundations of Computer Science, 1994, pp. 75–85.

[10] S.A. COOK AND C.W. RACKOFF, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[11] X. DENG, T. KAMEDA AND C. H. PAPADIMITRIOU, *How to learn an unknown environment*, J. Assoc. Comput. Mach., 45 (1998), pp. 215–245.

[12] X. DENG AND C. H. PAPADIMITRIOU, *Exploring an unknown graph*, in Proc. 31st Symp. on Foundations of Computer Science, 1990, pp. 356–361.

[13] ———, *Exploring an unknown graph*, Revised version of [12].

[14] J. EDMONDS AND C.K. POON, *A nearly optimal time-space lower bound for directed st-connectivity on the NNJAG model*, in Proc. 27th Symp. on Theory of Computing, 1995, pp. 147–156.

[15] F. HOFFMANN, C. ICKING, R. KLEIN AND K. KRIEGEL, *A competitive strategy for learning a polygon*, in Proc. 8th ACM-SIAM Symp. on Discrete Algorithms, 1997, pp. 166–174.

[16] E. KOUTSOUPIAS, Result reported in [13].

[17] S. KWEK, *On a simple depth-first search strategy for exploring unknown graphs*, in Proc. 5th Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 1272, Springer Verlag, New York, 1997, pp. 345–353.

[18] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Shortest paths without a map*, Theoretical Comput. Sci., 84 (1991), pp. 127–150.

[19] R. RIVEST, Problem formulation cited in [12].