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 [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 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 [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. 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 $\Omega(d^2m)$. Deng and Papadimitriou asked the question whether the exponential gap between

*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, Digital Equipment 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.

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, called *Balance*, 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 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 *Balance* algorithm 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 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. 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, $1 \leq i \leq d$, 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, 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 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.

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

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. A problem Q^δ, $\delta = 1, 2$, is the same as P^δ.

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

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

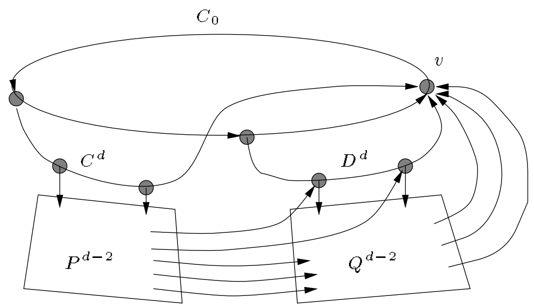

Figure 1: The graph for *Greedy*

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^δ or Q^δ, gets stuck at some node and cannot relocate to a suitable node by using only edges in P^δ resp. Q^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 $P^{\delta-2}$ and $Q^{\delta-2}$. Note (a) that, because of chain D^δ, no node in $Q^{\delta-2}$ can reach a node of $P^{\delta-2}$ without leaving P^δ. 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^δ 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^δ. 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^δ. 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}.

Generalized-Greedy: The graph used for the lower bound is outlined in Figure 2. The basic idea in the lower bound construction is as follows. *Generalized-Greedy* explores each subgraph Q_i^γ and its sibling R_i^γ "in parallel". Without loss of generality we can assume that the last chain traversed in the two subgraphs lies in Q_i^γ and the algorithm continues to explore Q_{i+1}^γ and R_{i+1}^γ. Let $N(\gamma)$ denote the number of times that the algorithm has to leave R_i^γ and traverse the root. We will show that $N(4\gamma) \geq N(\gamma)$, which implies that the root has to be traversed $N(d) \geq d^{\Omega(\log d)}$ times.

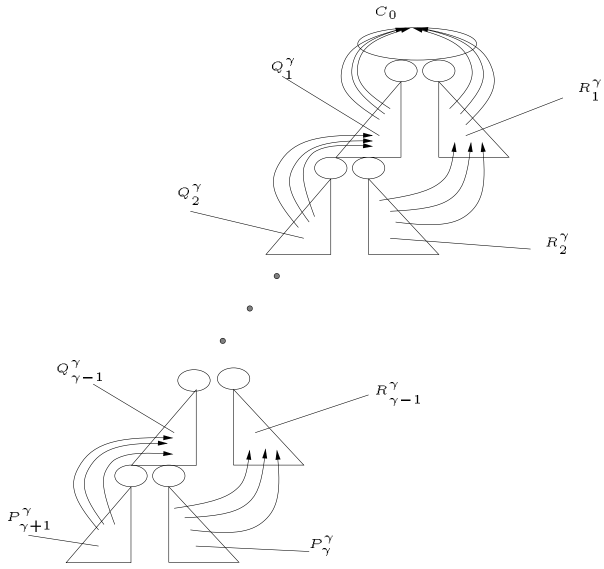

Figure 2: The graph for *Generalized-Greedy*

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^δ and the "sibling" graphs Q^δ and R^δ, for all $\delta \leq d$ that are a power of 4, and then show the lower bound on this graph.

A *problem* P^δ, $\delta > 1$, is a graph with two incoming edges and exactly δ outgoing edges. A *problem* R^δ, $\delta > 1$, consists of P^δ with $\delta - 2$ additional incoming edges. The *problem* Q^δ consists of R^δ 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_2^1, 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_2^1. The last edge of C_1^1 and of D_1^1 and the outgoing edges of P_1^1 and P_2^1 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_2^1. 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.

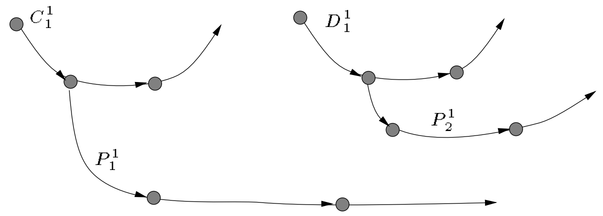

Figure 3: The subproblem P^4

$\delta = 4^l$, for some $l \geq 2$: Let $\gamma = \delta/4$. It is simpler to describe Q^δ first. The construction is depicted in Figure 4. Every node has the same indegree as outdegree, i.e., there are no sinks. Problem Q^δ consists of subproblems Q_i^γ and R_i^γ, for $1 \leq i \leq \gamma$, connected by chains C_i^γ and D_i^γ, 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_1^γ point into Q_1^γ. In general, the edges leaving the interior nodes of C_i^γ point into Q_i^γ. The same holds for the D-chains and R-subproblems. The first edge of C_i^γ and of D_i^γ are incoming edges of Q^δ, for $i = 1$, and start in Q_{i-1}^γ, for $1 < i \leq \gamma$, on a node of the leftmost subproblem Q^1 contained in Q_{i-1}^γ. 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_i^γ and the second outgoing edge is the first edge of D_i^γ.

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

There are $4\gamma + 2 = \delta + 2$ outgoing edges in Q^δ: The last edge of C_i^γ and the last edge of D_i^γ, for $1 \leq i \leq \gamma$, all edges leaving R_1^γ, all but two edges leaving Q_1^γ (the other two are the incoming edges of D_2^γ and C_2^γ), and two edges leaving Q_γ^γ. There are also $\delta + 2$ incoming edges: the first edge of C_1^γ and of D_1^γ, the edges pointing to the two interior nodes of C_γ^γ and D_γ^γ, the

γ incoming edges of Q^γ_γ, the $\gamma - 2$ incoming edges of R^γ_γ, and $2\gamma - 2$ incoming edges ending at the startnodes of C^γ_i and D^γ_i, for $2 \le i \le \gamma$.

A problem P^δ consists of 2γ chains C^γ_i and D^γ_i, $1 \le i \le \gamma$, as well as two subproblems P^γ_i, $\gamma \le i \le \gamma + 1$, and $2(\gamma - 1)$ subproblems Q^γ_i and R^γ_i, $1 \le i \le \gamma - 1$. These components are assembled in the same way as in Q^δ, except that Q^γ_γ is replaced by $P^\gamma_{\gamma+1}$, and R^γ_γ is replaced by P^γ_γ. Problems P^γ_γ and $P^\gamma_{\gamma+1}$ each have only two incoming edges from C^γ_γ and D^γ_γ, respectively.

There are $4\gamma = \delta$ outgoing edges in P^δ: The last edge of C^γ_i and the last edge of D^γ_i, for $1 \le i \le \gamma$, all but two edges leaving Q^γ_1 (the other two are the incoming edges of D^γ_2 and C^γ_2), all all edges leaving R^γ_1. There are two incoming edges in P^δ. The first edge of C^γ_1 and of D^γ_1 are incoming edges in every problem P^δ. The following $\delta - 2$ nodes are sinks for P^δ: the two interior nodes of C^γ_γ and of D^γ_γ, the $2\gamma - 2$ startnodes of C^γ_i and D^γ_i, for $2 \le i \le \gamma$, the $\gamma - 2$ sinks of P^γ_γ and the $\gamma - 2$ sinks if $P^\gamma_{\gamma+1}$.

A problem R^δ is a problem P^δ with an incoming edge into all sinks of P^δ. Thus there are δ incoming and δ outgoing edges.

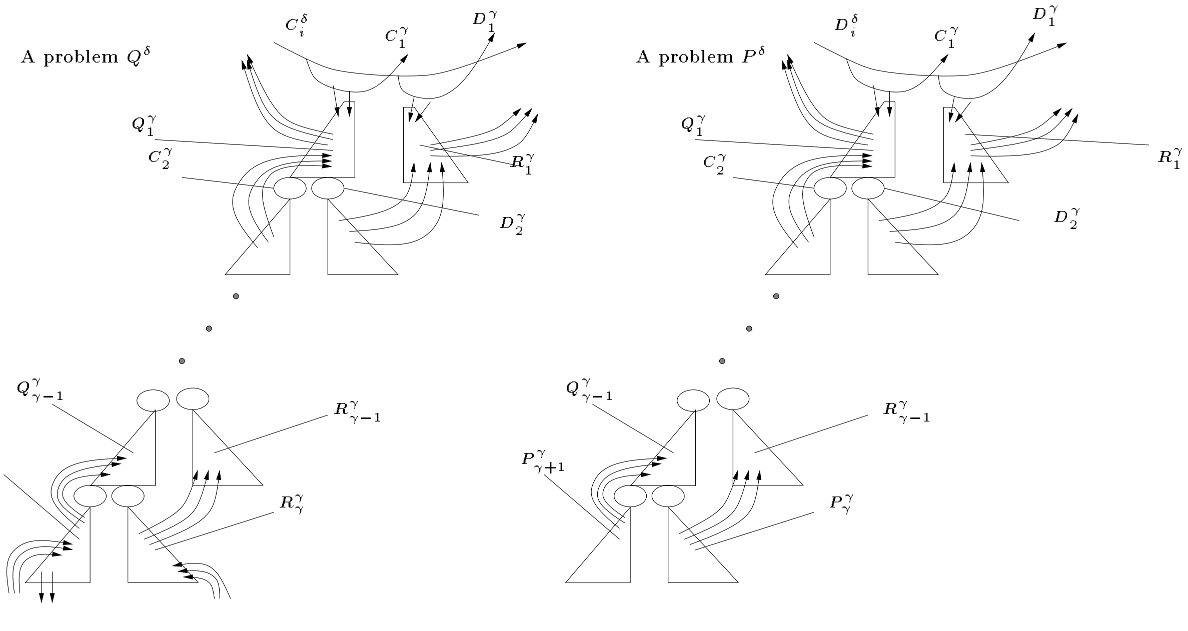

Figure 4: The subproblems Q^δ and P^δ

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

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

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

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 Appendix.

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 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 5. The robot returns to x and the next phase has the same current node and current chain.

   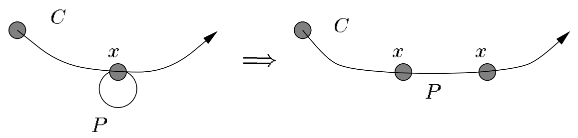

   Figure 5: Case 1

   Case 2: $y \neq x$, y has a token τ_i and is the startnode of a fresh chain D (see Figure 6)
   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.

   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

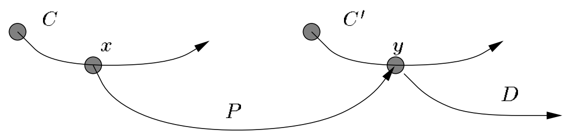

Figure 6: Case 2

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 7. 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.

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 7: The *Balance* algorithm

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. 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.

We describe the *Relocation* procedure, see Figure 8. 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.

Procedure Relocate(C)

```
if all chains are finished then return(empty_chain).
else Move robot to the startnode of C along closure(C);
    while C ≠ C_0 and T_v(C) is finished do
        Move robot to the startnode of parent(C) along closure(parent(C));
        C := parent(C);
    while C is finished do
        Let C_1, C_2, ..., C_l be the chains with parent(C_k) = C, 1 ≤ k ≤ 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), ..., T_v(C_l) having unfinished chains;
        C := C_k; x := startnode of C;
        Move robot to x;
    if C is not in progress then
        Compute closure(C);
    return(C)
```

Figure 8: The *Relocation* procedure

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, $d = -\sum_{v, with\ bal(v)<0} bal(v) \geq \sum_v t(v)$.

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 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. $\Box$

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

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

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 1. 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 *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 $k = 1, \ldots, l(j) - 1$, $C_{j,k+1} = \mathit{parent}(C_{j,k})$ 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.

Note that in line 18, edge e could only be traversed while nodes on C_1 are unfinished if the robot gets stuck at a node y on C^1, y having a token, and has to move to an unfinished node z on C^1 that lies before y. 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. However, the node selection rule in line 18 ensures that this is impossible because z is unfinished. This also implies that in line 18, the robot can always reach the first unfinished node on C by following C, without traversing $closure(C)$.

Thus, 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. $\square$

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, $l \leq d$. Suppose that node u_j, $1 \leq j \leq d$, 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), $2 \leq c(j) \leq d$.

For fixed j and $k \geq 2$, we have to show: Up to the time when $closure(C')$ is computed, whenever $active(T_{v_{j,k}}) = 1$, then $w(T_{v_{j,k}}) \leq \lfloor \delta/2 \rfloor$. 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 $k \geq 2$, $w(T_{v_{j,k}})$ 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

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

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. $\Box$

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. $\Box$

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.*

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 u-hard for an 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 1: 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 2: 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 1: 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 1, 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 5 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 $v(C)$. $\Box$

Proof of Claim 2: 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 6, 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 1, $l \leq d$. □

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

$$1 + 3d + 2 + d(d+1)(d^{2\log d+1} + (d+2)d^{2\log d+2}) + d(d+1) \leq (d+1)^5 d^{2\log d}$$

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

The total number of edge traversals used by *Balance* 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 $def(v) = \min\{0, in_u(v) - out_u(v)\}$. 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 $\sum_{v \in G} \mathit{def}(v) \leq d$ 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, $\mathit{def}(z)$ 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 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. We show the bound or 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 9. 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.

In the following let δ, $1 \leq \delta \leq d$, be a power of 5. A *problem* P^δ, for any integer $\delta \geq 5$, is a subgraph that has two *incoming* edges whose startnodes do not belong to P^δ but whose endnodes do, and $\delta + 1$ *outgoing* edges whose startnodes belong to P^δ 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^δ we also need problems Q^δ. These problems are identical to P^δ except that, for $\delta > 1$, Q^δ 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^δ consists of $3\gamma^2$ chains $C^\gamma_{i,k}$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma$, as well as γ chains D^γ_i and γ recursive subproblems Q^γ_i, $1 \leq i \leq \gamma - 1$, and P^γ_γ.

These components are assembled as follows. One of the incoming edges of P^δ is the first edge of $C^\gamma_{1,1}$. We assume that $v_1(C_0)$ is the startnode of $C^{d/5}_{1,1}$. Node $v^1(C^\gamma_{i,k})$ is the startnode of $C^\gamma_{i,k+1}$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma - 1$. Node $v^1(C^\gamma_{i,3\gamma})$ is the startnode of $C^\gamma_{i+1,1}$, $1 \leq i \leq \gamma - 1$. The last edge of $C^\gamma_{1,k}$, $1 \leq k \leq 3\gamma$, is an outgoing edge of P^δ. The endnode of $C^\gamma_{i,k}$ is equal to the startnode of $C^\gamma_{i-1,k}$, $2 \leq i \leq \gamma$ and $1 \leq k \leq 3\gamma$. Note that the last edge of $C^\gamma_{2,1}$ hence is an

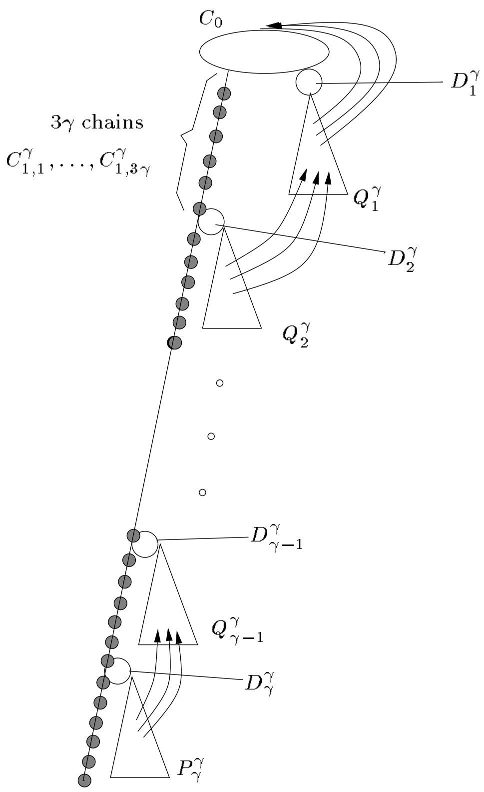

Figure 9: The graph G

outgoing edge of P^δ. Nodes $v^2(C^\gamma_{i,k})$, $1 \le i \le \gamma$, $1 \le k \le 3\gamma - 1$, have no additional outgoing edge but nodes $v^2(C^\gamma_{i,3\gamma})$, $1 \le i \le \gamma - 1$, do. Chain $C^\gamma_{\gamma,3\gamma}$ has no additional outgoing edges.

The second incoming edge of P^δ is the first edge of a chain D_1^γ and, for $2 \le i \le \gamma$, the edge leaving $v^2(C^\gamma_{i-1,3\gamma})$ is the first edge of D_i^γ. For $1 \le i \le \gamma$, the last edge of D_i^γ is an outgoing edge of P^δ. 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^γ, $1 \le i \le \gamma$, each have an additional outgoing edge. For $1 \le i \le \gamma - 1$, these two edges point into Q_i^γ and, for $i = \gamma$, they point into P_γ^γ.

If $\delta = 5$, then the outgoing edge of the only subproblem P^1 is an outgoing edges of $P^\delta = P^5$. If $\delta > 5$, the problems Q_i^γ, $1 \le i \le \gamma - 1$, and P_γ^γ each have $\gamma + 1$ outgoing edges. For Q_1^γ, γ these edges are also outgoing edges of P^δ and one edge points to the interior node of D_1^γ that is the startnode of $C^\gamma_{1,1}$. For $2 \le i \le \gamma - 1$, exactly $\gamma - 1$ edges leaving Q_i^γ point into Q^γ_{i-1} 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^δ_{i-1} that does not get an edge from Q^γ_{i-1} and the remaining edge points to the interior node of D_i^γ that is the startnode of $C^\gamma_{1,1}$. In the same way the edges leaving P_γ^γ are connected with $Q^\gamma_{\gamma-1}$, $D^\gamma_{\gamma-1}$ and D^γ_γ.

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

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

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

Let $N(\delta)$ be the number of times the following event happens while *Balance* works on a problem P^δ or Q^δ: *Balance* generates a new chain, gets stuck and cannot reach a node with new outgoing edges by using only edges in P^δ resp. Q^δ. Problem P^δ contains γ subproblems $Q^\gamma_1, \ldots, Q^\gamma_{\gamma-1}$ and P^γ_γ. 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^δ. This is because of the following facts: (1) When *Balance* explores Q^γ_i, $1 \leq i \leq \gamma - 1$, or P^γ_γ, the subproblems $Q^\gamma_1, \ldots, Q^\gamma_{i-1}$ resp. $Q^\gamma_1, \ldots, Q^\gamma_{\gamma-1}$ are already finished. (2) The chains $D^\gamma_1, \ldots, D^\gamma_\gamma$ ensure that *Balance* cannot reach any chain $C^\gamma_{i,k}$, $1 \leq i \leq \gamma$, $1 \leq k \leq 3\gamma$, from where the unfinished chains in P^δ can be reached. Again the same holds for a problem Q^δ. 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. □

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 $d \geq 1$, 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 1. □

Acknowledgment

We thank Prabhakar Raghavan for bringing to our attention the literature on the s–t connectivity problem.

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] G. Barnes, J.F. Buss, W.L. Ruzzo and B. Schieber. A sublinear space, polynomial time algorithms for directed s–t connectivity. *Proc. 7th Annual Conf. on Structure in Compexity Theory*, 27–33, 1992.

[4] G. Barnes and J. Edmonds. Time-space lower bounds for directed s–t connectivity on JAG models. *Proc. 34th Symp. on Foundations of Computer Science*, 228–237, 1993.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] S.A. Cook and C.W. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9:636–652, 1980.

[11] 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.

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

[13] X. Deng and C. H. Papadimitriou. 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. *Proc. 27th Symp. on Theory of Computing*, 147–156, 1995.

[15] 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.

[16] E. Koutsoupias. Result reported in [13].

[17] S. Kwek. On a simple depth-first search strategy for exploring unknown graphs. To appear in *Proc. 5th Workshop on Algorithms and Data Structures (WADS)*, 1997.

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

[19] R. Rivest. Problem formulation cited in [12].

5 Appendix

Lemma 8 *After at most d restarts of the Balance algorithm, whenever a new sink s_i, $1 \leq i \leq d$, was discovered, we can ensure that the subgraph of explored edges is always strongly connected.*

Proof: Whenever the *Balance* algorithm gets stuck in line 14 and there is no path to s using edges that have been traversed before, then the algorithm stops and has to be restarted as follows. Let $s_1, \ldots, s_{k-1}$ be the already discovered sinks. Assume the robot gets stuck at y on a path P that does *not* belong to any other chain created since the last restart. Whenever this happens, y is a newly discovered sink s_k and must have occurred on P before (each node has degree ≥ 2). Take the cycle C between the two occurrences of s_k on P and restart the algorithm with current node y and current chain C with the following modification: All edges traversed before the restart are marked as $k-1$-visited. We show below that there exists a path of $k-1$-visited edges from all previously visited sinks to y. Whenever the algorithm started at s_k encounters an already visited sink s_i, $i < k$, then the algorithm traverses the $k-1$-visited edges on the path from s_i to s_k as required in lines 16 and 17 of the algorithm, i.e., the algorithm does not get stuck at s_i, $i < k$.

Thus, whenever the modified algorithm restarts, it has discovered a new sink s_k. Hence, after at most d restarts, all sinks have been discovered and there is a path from the every sink s_i with $i < k$ to s_k.

We show inductively that there exists a path from all previously explored sinks to y. Obviously the claim holds initially. Whenever $y = s_j$ for $j > 1$, then obviously there exists a path from s_{j-1} to s_j, since this is how the previous algorithm got stuck. Thus, by transitivity of the reachability relation, all previously visited sinks can reach s_j. $\square$