Tree Exploration with Advice ⋆

Pierre Fraigniaud [a,1], David Ilcinkas [b,1,*], Andrzej Pelc [c,2]

[a] *CNRS and Univ. Denis Diderot (LIAFA), Paris, France*
[b] *CNRS and Univ. Bordeaux I (LaBRI), France*
[c] *Dép. d'informatique, Univ. du Québec en Outaouais, Canada*

Abstract

We study the amount of knowledge about the network that is required in order to efficiently solve a task concerning this network. The impact of available information on the efficiency of solving network problems, such as communication or exploration, has been investigated before but assumptions concerned availability of *particular* items of information about the network, such as the size, the diameter, or a map of the network. In contrast, our approach is *quantitative*: we investigate the minimum number of bits of information (bits of advice) that has to be given to an algorithm in order to perform a task with given efficiency.

We illustrate this quantitative approach to available knowledge by the task of tree exploration. A mobile entity (robot) has to traverse all edges of an unknown tree, using as few edge traversals as possible. The quality of an exploration algorithm $\mathcal{A}$ is measured by its *competitive ratio*, i.e., by comparing its cost (number of edge traversals) to the length of the shortest path containing all edges of the tree. Depth-First-Search has competitive ratio 2 and, in the absence of any information about the tree, no algorithm can beat this value.

We determine the minimum number of bits of advice that has to be given to an exploration algorithm in order to achieve competitive ratio strictly smaller than 2. Our main result establishes an exact threshold number of bits of advice that turns out to be roughly $\log \log D$, where D is the diameter of the tree. More precisely, for any constant c, we construct an exploration algorithm with competitive ratio smaller than 2, using at most $\log \log D - c$ bits of advice, and we show that every algorithm using $\log \log D - g(D)$ bits of advice, for any function g unbounded from above, has competitive ratio at least 2.

Preprint submitted to Elsevier *6 August 2008*

1 Introduction

For many network problems (such as leader election, minimum spanning tree, rendezvous, wakeup, broadcasting, etc.), the quality of the algorithmic solutions often depends on the amount of knowledge given to nodes of the network, or given to mobile entities moving in the network, about its topology. *Local* knowledge given to every node and/or to every mobile entity is its identity and, for a node, its degree (or the list of neighbor identities). Any other knowledge (e.g., the total number of nodes, network diameter, the total number of mobile entities, partial maps of the network, sense of direction in the network etc.) is *global* knowledge. Many results illustrate the impact of global knowledge on the ability and efficiency of solving network problems. For instance, it is proved in [4] that, if an upper bound $\hat{n}$ on the number n of nodes of a graph is known, then a robot can explore this graph in time polynomial in $\hat{n}$, using one pebble, while without this knowledge, $\Theta(\log \log n)$ pebbles are necessary and sufficient. The role of the type of global knowledge known as the sense of direction in a network has been studied, e.g., in [21], for various distributed tasks. Broadcasting in radio networks is another subject where global information significantly influences efficiency. In [25], it is shown that, if nodes have complete knowledge of the network, then deterministic broadcasting can be done in time $O(D+\log^3 n)$, for n-node radio networks with diameter D. (This result has been recently improved to $O(D + \log^2 n)$ in [27]). On the other hand, in [9], a lower bound of $\Omega(n \log D)$ is proved on deterministic broadcasting time in radio networks in which nodes know only their own identity. (An almost matching upper bound of $O(n \log^2 D)$ is proved in [10]). In fact, the impact of global knowledge is significant in many areas of distributed computing, as

---

⋆ A preliminary version of this paper appeared in the Proc. 31st International Symposium on Mathematical Foundations of Computer Science, (MFCS 2006), LNCS 4162, 24-37 (invited talk of A. Pelc).

* Corresponding author
LaBRI, bât A30, Université Bordeaux I
351 cours de la Libération
33405 Talence Cdex, France
Phone: (+33) 540 006 912. Fax: (+33) 540 006 669.
Email addresses: pierre.fraigniaud@liafa.jussieu.fr (Pierre Fraigniaud), david.ilcinkas@labri.fr (David Ilcinkas), pelc@uqo.ca (Andrzej Pelc).

[1] Pierre Fraigniaud and David Ilcinkas were both supported by the projects PairAPair of the ACI Masses de Données, and FRAGILE of the ACI Sécurité Informatique. Additional support from the INRIA project "Grand Large".
[2] Andrzej Pelc was supported in part by NSERC discovery grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais. This work was done during the visit of Andrzej Pelc at LRI. Additional supports from the Ministère des Relations Internationales du Québec, from University of Paris-Sud, and from the project PairAPair of the ACI Masses de Données.

witnessed by [19,28] where hundreds of impossibility results and lower bounds for distributed computing are surveyed, many of them depending on whether or not the nodes are given exact or approximate values of global parameters providing partial knowledge of the topology of the network. Finally, notice that the amount of global knowledge has also a strong impact on computing in anonymous networks. (See, e.g., [26], where the impact of knowing the total number of nodes is studied in depth.)

We interpret global knowledge, given to the nodes or to the mobile entities, as the advice obtained from an *oracle*. Given a problem $\mathcal{P}$ with the set of instances $\mathcal{I}$, an oracle is a function $\mathcal{O} : \mathcal{I} \mapsto \{0,1\}^*$ that maps any instance I to a binary string $\mathcal{O}(I)$, called the *advice* of oracle $\mathcal{O}$ on instance I. Solving problem $\mathcal{P}$ using oracle $\mathcal{O}$ consists in designing an algorithm that, given the advice $\mathcal{O}(I)$, but unaware of I, returns a $\mathcal{P}$-*scheme* for I, i.e., a sequence of instructions executed by the nodes or the mobiles entities, solving $\mathcal{P}$ for I. In this setting, the amount of global knowledge is measured by the number of bits of advice on every instance I, i.e., the length of the binary string $\mathcal{O}(I)$. Typical questions of interest are then: "What is the minimum number of bits of advice for solving problem $\mathcal{P}$?" or "What is the minimum number of bits of advice for solving $\mathcal{P}$ within some amount of time?". The novelty and significance of our modeling of global knowledge is that it enables asking such *quantitative* questions about the required knowledge, regardless of what *kind* of knowledge is supplied. This should be contrasted with the traditional approach that assumes availability of particular items of global information.

Modeling knowledge about the network by the advice obtained from an oracle has already proved useful in the context of communication problems. In a recent paper [23], we showed tight bounds on the number of bits of advice required for an efficient execution of two fundamental communication tasks: broadcast and wakeup. It turns out that the minimum number of bits of advice required for broadcast with a linear number of messages is strictly larger than that required for wakeup with a linear number of messages. In this paper, we address similar quantitative questions about knowledge required for one of the fundamental problems in mobile computing: the exploration problem. We prove a tight bound of roughly $\log \log D$ on the number of bits of advice enabling the design of an exploration algorithm with competitive ratio strictly less than 2, on trees of diameter D.

Added in proof. After the publication of the preliminary (conference) version of this paper, the advice paradigm has been used to investigate various other network problems: in [22] to study distributed graph coloring, in [24] to study the distributed minimum spanning tree construction, and in [32] to study graph searching.

1.1 The background of tree exploration

A robot has to traverse all edges of an undirected connected graph, using as few edge traversals as possible. Graph exploration is most often performed when the robot lacks some essential information on the explored graph. In such case, the quality of an exploration algorithm $\mathcal{A}$ is measured by comparing its cost (number of edge traversals) to the length of the shortest *covering walk* (i.e., the shortest path containing all edges of the graph). This ratio, maximized over all graphs and all starting nodes, is called the *competitive ratio* $\mathcal{R}(\mathcal{A})$ of algorithm $\mathcal{A}$. The situation here is similar to the context of online algorithms, where competitive ratio first appeared. In both cases, the performance of an algorithm lacking some essential knowledge about the environment is compared to that of an algorithm that has this knowledge: in the case of online algorithms, this knowledge concerns future events, and in the case of exploration, it concerns the topology of the graph and its labeling. (An algorithm provided with a fully labeled copy of the explored graph, showing which port at a visited node leads to which neighbor, can find the shortest covering walk off line.)

Depth-First-Search has competitive ratio 2 and it was shown in [14] that no exploration algorithm can beat this value for arbitrary graphs, even when provided with an unlabeled isomorphic copy of the explored graph with the starting node marked. It turns out that merely the absence of labels of ports and nodes in the map is sufficient to confuse any algorithm on some graphs, making it not better than DFS. On the other hand, in the absence of any global information whatsoever, beating competitive ratio 2 was shown impossible even for the family of trees. Hence the following question becomes natural. Is it possible to achieve competitive ratio smaller than 2, for tree exploration, if the algorithm is provided with some partial information concerning the explored environment? In [14] a positive answer to this question was given in the case of very large additional information: the robot was provided with an unlabeled map of the tree. However, this assumption is not very realistic. Indeed, exploration is often used as a tool to construct a map of an unknown network, and usually a priori information about the explored network is much more restricted.

1.2 The problem

We consider the problem of the *amount of information* needed to achieve tree exploration with competitive ratio smaller than 2. (Recall that the reason of restricting attention to trees is the above mentioned negative result for general graphs, showing that already relatively simple graphs force competitive

ratio at least 2 even with extensive additional information, namely an entire unlabeled copy of the explored graph.)

The problem is formalized as follows. In the framework of tree exploration, we define an *oracle* to be a function $\mathcal{O}$ from the class of all trees to the class of binary strings. Specifically, for every tree T, an exploration algorithm is provided with the advice string $\mathcal{O}(T)$ and returns an *exploration scheme* for T. Such a scheme, starting at any node u, traverses all edges of T. We ask what is the minimum number of bits of advice for which there exists an exploration algorithm achieving competitive ratio smaller than 2, for all trees.

1.3 Our results

We use the notion of advice to measure the minimum amount of information required for the design of an efficient exploration algorithm. Our main result establishes an exact threshold number of bits of advice to achieve competitive ratio smaller than 2 for tree exploration. This threshold turns out to be roughly $\log \log D$, where D is the diameter of the tree. More precisely, for any constant c we construct an exploration algorithm with competitive ratio smaller than 2, using at most $\log \log D - c$ bits of advice, and we show that every algorithm using $\log \log D - g(D)$ bits of advice, for any function g unbounded from above, has competitive ratio at least 2.

It is interesting to note the *structure* of the advice in our positive result. For any tree T, this is a string s of bits depending only on D, and giving an approximation of it, plus an additional bit b that allows the robot to choose between two types of exploration. This additional bit b (depending on D and on the size of the tree) is very important. Indeed, while the string s depends only on D and has length smaller than $\log \log D$, we show that even the full knowledge of D, but without b, is not sufficient to beat competitive ratio 2. More precisely, we show that every exploration algorithm knowing only the diameter of the tree must have competitive ratio at least 2.

1.4 Related work

Exploration of unknown environments has been extensively studied in the literature, both in the geometric and in the graph setting. In the first scenario the environment is modeled, e.g., as a terrain with obstacles that may be convex [7], polygonal [11] or rectangular [3]. Another way is to represent the unknown environment as a graph, assuming that the robot may only move along its edges. The graph model is further specified in two different ways. In [1,4,5,13,20], the robot explores strongly connected directed graphs and it

can move only in the direction from tail to head of an edge, not vice-versa. In [1,13], the authors study competitive ratio of algorithms exploring directed graphs. The constructed algorithms have competitive ratio exponential in the deficiency d of the graph [13], or competitive ratio $d^{O(\log d)}m$, where m is the number of edges [1]. Recently, the first exploration algorithm with competitive ratio polynomial in the deficiency of the graph has been given in [20].

In [2,8,14,18,29,30] the explored graph is undirected and the robot can traverse edges in both directions. In some papers, additional restrictions on the moves of the robot are imposed. It is assumed that the robot has either a restricted tank [2,8], forcing it to periodically return to the base for refueling, or that it is tethered, i.e., attached to the base by a rope or cable of restricted length [18].

Another direction of research concerns exploration of anonymous graphs (directed or undirected). In this case it is impossible to explore arbitrary graphs and stop, if no marking of nodes is allowed. Hence the scenario adopted in [4,5] is to allow pebbles which the robot can drop on nodes to recognize already visited ones, and then remove them and drop in other places. The authors concentrate attention on the minimum number of pebbles allowing efficient exploration of arbitrary directed graphs. Exploring anonymous trees without the possibility of marking nodes is investigated in [15]. The authors concentrate attention not on the cost of exploration but on the minimum amount of memory sufficient to carry out this task. Exploration of anonymous graphs was also considered in [12,16,17].

2 Terminology and preliminaries

For any tree T we denote by $|T|$ the number of nodes of T, and call it the *size* of this tree. For a given tree T and starting node u, we denote by $opt(T,u)$ the length of the shortest covering walk of T starting from u, i.e., the length of the shortest path in T starting from u and containing all edges of T. Clearly, $opt(T,u) = 2(n-1) - ecc(u)$, where n is the size of T and $ecc(u)$ is the eccentricity of the starting node u, i.e., the distance from u to the farthest leaf. Depth-First-Search ending in the leaf farthest from the starting node u uses fewest edge traversals.

We assume that all ports at a node v are numbered 1,...,deg(v). Hence the robot can recognize already visited nodes and traversed edges. However, it cannot tell the difference between yet unexplored edges incident to its current position. The robot executes a given *exploration scheme* that, at every node v, makes one of the following decisions: take a specific already explored edge, or take an unexplored edge. If the scheme decides to take an unexplored edge,

the actual choice of the edge belongs to an adversary, as we are interested in worst-case performance.

We want an oracle to provide information on the topology of the explored tree, independently of any labeling, hence we define it as a function $\mathcal{O}$ from the class of all *unlabeled* trees to the class of binary strings. For any string s, a tree T such that $\mathcal{O}(T) = s$ is called *compatible* with s. If a tree exploration algorithm $\mathcal{A}$ takes the advice $\mathcal{O}(T)$ as input for any tree T, we say that $\mathcal{A}$ *uses* oracle $\mathcal{O}$.

Consider an exploration algorithm $\mathcal{A}$ using oracle $\mathcal{O}$. For any string s in the range of $\mathcal{O}$, algorithm $\mathcal{A}$ produces an exploration scheme that explores all trees compatible with s. For any such tree T and starting node u, the *cost* $\mathcal{A}(T, u)$ of this scheme, run on tree T from the starting node u, is the worst-case number of edge traversals taken over all of the above mentioned choices of an adversary. The competitive ratio of $\mathcal{A}$ is defined as

$$\mathcal{R}(\mathcal{A}) = \sup_{T,u} \frac{\mathcal{A}(T, u)}{opt(T, u)} \ ,$$

where the supremum is taken over all trees T and all starting nodes u of T.

The fact that an oracle is defined on unlabeled rather than labeled trees is an important distinction. For example, for the class of lines, we will prove that (asymptotically) $\log \log n$ bits of advice are needed to achieve competitive ratio smaller than 2, where n is the length of the line. However, for a *given* labeling, a single bit of advice (indicating the port at the starting node leading to the closer endpoint of the line) is enough to achieve competitive ratio 1: DFS starting toward the closer endpoint achieves it.

The following remark will be useful for proving lower bounds on the competitive ratio of exploration algorithms. Suppose that the robot, at some point of the exploration, is at node v, then moves along an already explored edge e incident to v, and immediately returns to v. For any set of decisions of an adversary, an algorithm causing such a pair of moves, when run on a tree T from some starting node u, has cost strictly larger than the algorithm that skips these two moves. Hence, we restrict attention to exploration algorithms that never perform such returns. We call them *regular*.

In [14], the authors introduced the following classification of exploration algorithms for the class of lines. (They considered exploration algorithms that know the length n of the line.) Fix n and let *type* k be the set of algorithms that always do at most k returns before reaching an endpoint, and that do exactly this many returns for some combination of starting node and (adversary's) choice of the initial direction. They proved the following result that can be used to restrict attention to relatively simple algorithms exploring lines, when

looking for minimum competitive ratio.

Lemma 1 [14] *Fix $n \geq 11$. For every exploration algorithm $\mathcal{A}$ for the line L_n of length n, there exists an algorithm $\mathcal{A}'$ for L_n, such that $\mathcal{A}'$ is of type 1 and $\sup_{u \in L_n} \frac{\mathcal{A}'(L_n,u)}{opt(L_n,u)} \leq \sup_{u \in L_n} \frac{\mathcal{A}(L_n,u)}{opt(L_n,u)}$.*

In our setting, an algorithm does not know the length of the line but only the bits of advice. Hence we change the notion of type in the following way. Consider an algorithm $\mathcal{A}$ using oracle $\mathcal{O}$. Fix a string s in the range of $\mathcal{O}$ and consider the exploration scheme produced by $\mathcal{A}$ for this string. This scheme is of type k if it always does at most k returns before reaching an endpoint, for any line L_n of length n compatible with s, and any starting node u, and if it does exactly this many returns for some line compatible with s, some starting node and some adversary choice of the initial direction.

In the proof of Lemma 1, the algorithm $\mathcal{A}'$ is obtained from $\mathcal{A}$ independently of n. Hence this lemma implies that in our setting the best competitive ratio for the class of lines is achieved by an exploration algorithm that, for any string s, produces a scheme of type 1. This is a class of simple exploration schemes that go x steps in one direction (unless an endpoint is met), then return and go to an endpoint, then return and go to the other endpoint. For any scheme of type 1, this integer x will be called the *probing distance* of the scheme.

The next lemma describes the performance of schemes of type 1 as a function of the probing distance.

Lemma 2 *For any positive integer n and any $\alpha < 1$, consider an exploration scheme of type 1 for the line L_n of length n, with probing distance $\lfloor \alpha n \rfloor$, and let $t_{\alpha,n}(u)$ be the cost of this scheme, for starting node u. Let $F_n(\alpha) = \sup_{u \in L_n} \frac{t_{\alpha,n}(u)}{opt(L_n,u)}$. Then, there exists a positive integer N_0, such that for any $n \geq N_0$, the function F_n is strictly decreasing in the interval $(0, \frac{\sqrt{3}-1}{2}]$, and $\sup_{n>0} F_n(\alpha) < 2$, for any α in this interval.*

PROOF. Fix a starting node u in L_n. Let a be the distance from u to the endpoint of the line toward which the robot moves first, and let $b = n - a$. We may assume $a, b > 0$, otherwise $t_{\alpha,n}(u) = opt(L_n, u)$. Let $x = \lfloor \alpha n \rfloor$. If $a \leq x$ then, since $\alpha \leq \frac{\sqrt{3}-1}{2} < 1/2$, we have $t_{\alpha,n}(u) = opt(L_n, u)$. Hence assume $a > x$. If $a \leq b$ then $\frac{t_{\alpha,n}(u)}{opt(L_n,u)} = \frac{2x+b+n}{2n-b}$, which is maximized for $b = n - x - 1$ and in this case is equal to $\frac{2n+x-1}{n+x+1}$. If $a > b$ then $\frac{t_{\alpha,n}(u)}{opt(L_n,u)} = \frac{n+2x+b}{n+b}$, which is maximized for $b = 1$ and in this case is equal to $\frac{n+2x+1}{n+1}$. For $\alpha \leq \frac{\sqrt{3}-1}{2}$ and sufficiently large n, we have $\frac{2n+x-1}{n+x+1} \geq \frac{n+2x+1}{n+1}$, and hence $F_n(\alpha) = \frac{2n+x-1}{n+x+1}$. This fraction is a decreasing function of x, hence F_n is a decreasing function of α

in the interval $(0, \frac{\sqrt{3}-1}{2}]$. For the second part notice that, for sufficiently large n, we have $F_n(\alpha) = \frac{2n+x-1}{n+x+1} \leq \frac{2+\alpha}{1+\alpha}$, which is smaller than 2 for $\alpha > 0$. $\square$

3 The upper bound

In this and the next section, we prove our main result, establishing the exact threshold on the number of bits of advice for which an exploration algorithm can have competitive ratio smaller than 2. This result is presented in two theorems, one of which establishes an upper bound on the required number of bits of advice, by constructing an appropriate exploration algorithm, and the other, in section 4, proves a matching lower bound. In this section, we establish the upper bound, by constructing exploration algorithm SKE(c) (for SMALL-KNOWLEDGE-EXPLORATION(c)), for an arbitrary positive integer constant c. This algorithm has competitive ratio smaller than 2, and uses an oracle $\mathcal{O}_c$ of size at most $\max(1, \log\log D - c)$, for any tree of diameter D.

We first describe the oracle $\mathcal{O}_c$. Fix $c > 0$. Given a tree T of diameter D, the oracle $\mathcal{O}_c$ outputs the following advice: a bit called `choice` and, if `choice` $= 1$, an integer k using $\lceil \log \lceil \log D \rceil \rceil - (c+3)$ bits. The bit `choice` is used by the algorithm to make a decision concerning two alternative ways of exploration, and the integer k is used to obtain an approximation D_0 of the diameter.

Let N_0 be an integer (whose existence is guaranteed by Lemma 2) such that, for all $n \geq N_0$, the function F_n is strictly decreasing in the interval $(0, \frac{\sqrt{3}-1}{2}]$, and $\sup_{n>0} F_n(\alpha) < 2$, for any α in this interval. For $\alpha \in (0, \frac{\sqrt{3}-1}{2}]$, let $\beta(\alpha) = \sup_{n>0} F_n(\alpha)$. Let T be any tree and let n and D be, respectively, its number of nodes and its diameter. Take ϵ such that $D = (1-\epsilon)n$. We will use the following abbreviations: $\lambda = \frac{\sqrt{3}-1}{2}$, and $\gamma = 2^{2^{c+3}+1}$. We now define a threshold ϵ^* on the value of ϵ that will serve to define the bit `choice`. Let $\epsilon_1 = \frac{\lambda}{16\gamma}$, $\beta_1 = \beta(\epsilon_1)$, $\epsilon_2 = \frac{2-\beta_1}{624}$, and $\epsilon^* = \min(\epsilon_1, \epsilon_2)$. The oracle sets `choice` to 1 if

$$(\epsilon < \epsilon^*) \wedge (D \geq 2^{2^{c+3}}) \wedge (n \geq N_0) \ ,$$

and sets `choice` to 0 otherwise. If `choice` $= 1$, the oracle computes $k = \lfloor \frac{\lceil \log D \rceil}{2^{c+3}} \rfloor$.

Given `choice` and k, Algorithm SKE(c) returns an exploration scheme. If `choice` $= 0$, then this scheme is an arbitrary DFS. To fix attention, we take the DFS that always chooses the smallest yet unused port number at every node. Note that `choice` is set to 0 when the diameter of the tree is significantly smaller than its size, or when the diameter is bounded, or when the tree itself is small.

We now describe the much more subtle scheme $\mathcal{X}_c$ produced by the algorithm when `choice` $= 1$. The scheme $\mathcal{X}_c$ uses Procedure `DPDFS`(v) (for DOUBLING-PARTIAL-DEPTH-FIRST-SEARCH(v)) that is called at a node v of the explored tree, outputs the two edges connecting v to the two largest subtrees rooted at neighbors of v, completely explores all other subtrees, and eventually returns to v. In the sequel, we will use the notion of a subtree pending from v as an equivalent to the notion of a subtree rooted at a neighbor of v. Procedure `DPDFS`(v) is described in Figure 1.

Procedure `DPDFS`(v)
  $i \leftarrow 1$;
  $S \leftarrow$ set of edges incident to v, connecting v to subtrees not yet completely explored;
  while $|S| \geq 3$ **do**
    $S' \leftarrow S$;
    while $S' \neq \emptyset$ **do**
      let $e \in S'$ and let $T(e)$ be the subtree connected to v by edge e;
      explore $T(e)$ by DFS until $\min(|T(e)|, 2^i - 1)$ nodes are visited;
      return to v;
      $S' \leftarrow S' \setminus \{e\}$;
      if $T(e)$ is completely explored **then** $S \leftarrow S \setminus \{e\}$;
    $i \leftarrow i + 1$;
  if $|S| = 2$ **then return** S;
  if $|S| = 1$ **then** let e' be the edge connecting v to the largest explored subtree and **return** $S \cup \{e'\}$;
  if $S = \emptyset$ **then** let e' and e'' be the edges connecting v to the two largest explored subtrees and **return** $\{e', e''\}$;

Fig. 1. Procedure `DPDFS`

Lemma 3 *Let v be any node of degree at least 3. Let $T_1, \ldots, T_p$ be the enumeration of the subtrees pending from v in decreasing order of their sizes. Procedure* `DPDFS`(v) *returns two edges corresponding to two largest subtrees (up to size equality), and completely explores all other subtrees pending from v. Moreover, the cost of Procedure* `DPDFS`(v) *is at most $22 \sum_{i \geq 3}^{p} |T_i|$.*

PROOF. The first part is straightforward. For the second part, let $x_i = \lfloor \log |T_i| \rfloor$, for $i = 1, \ldots, p$. Let phase j denote the jth turn of the external while loop. Since $2^{x_i} - 1 < |T_i| \leq 2^{x_i+1} - 1$, subtree T_i is completely explored at the end of phase $x_i + 1$, and not previously. Hence the last executed phase is the phase $x_3 + 1$.

We first bound the cost of exploring T_1. During phase j, DFS visits at most 2^j

nodes of this subtree. This DFS costs at most $2 \cdot 2^j$, including the return to v. For the complete procedure, the cost of exploring T_1 is at most $\sum_{j=1}^{x_3+1} 2 \cdot 2^j \leq 2 \cdot 2^{x_3+2} = 8 \cdot 2^{x_3} \leq 8|T_3|$. The estimate holds for exploration of T_2 as well. Hence the cost of exploring both T_1 and T_2 is at most $16|T_3|$.

Let $3 \leq i \leq p$. The cost of exploring T_i in phase x_i+1 is exactly $2|T_i|$. The total cost of exploring T_i is thus at most $2|T_i| + \sum_{j=1}^{x_i} 2 \cdot 2^j \leq 2|T_i| + 4 \cdot 2^{x_i} \leq 6|T_i|$. Therefore the total cost of exploring all subtrees T_i, for $i \geq 3$, is at most $6 \sum_{i=3}^{p} |T_i|$.

Since $|T_3| \leq \sum_{i \geq 3}^{p} |T_i|$, the total cost of Procedure DPDFS(v) is bounded by $22 \sum_{i \geq 3}^{p} |T_i|$. $\square$

The intuitive idea of the exploration scheme $\mathcal{X}_c$ (returned by Algorithm SKE(c) when choice $= 1$) is the following. Let $D_0 = 2^{k \cdot 2^{c+3}-1}$. We will prove that D_0 approximates the diameter D as follows: $D_0 \leq D < \gamma D_0$. The robot uses Procedure DPDFS(v) to identify the two edges connecting the current node v to the largest subtrees pending from it. Then the robot moves along one of the edges and applies the procedure again. These consecutive applications define a path of length approximately equal to the diameter of the tree. On this path the robot applies a scheme of type 1 for lines: go at probing distance $\lfloor \lambda D_0/2 \rfloor$, return and go to the endpoint of the path, return and go to the other endpoint of the path. The approximation D_0 of the diameter is tight enough to guarantee good performance of the scheme on this path. On the other hand, the part of the tree disjoint from this path is negligible (this is implied by the conditions of setting choice to 1). These two facts (shown in detail in the proof of Theorem 5) imply that the competitive ratio of scheme $\mathcal{X}_c$ is smaller than 2.

The description of the exploration scheme $\mathcal{X}_c$ is provided in Figure 2. In the description, moves performed during the calls to Procedure DPDFS are called *internal*, and all other moves are called *external*. During the entire exploration, the robot stores the results of all previous actions, and constructs a map of the portion of the tree that has been explored so far.

Lemma 4 *Algorithm* SKE(c) *is correct.*

PROOF. If the oracle sets choice to 0, then Algorithm SKE(c) returns DFS as the exploration scheme, which clearly visits all nodes of the tree. Assume that the oracle sets choice to 1. We will prove that the returned scheme $\mathcal{X}_c$ visits all nodes of the tree. Assume this is not the case. Since at each execution of the while loop in $\mathcal{X}_c$ the robot checks (before terminating) whether the tree is completely explored, our assumption implies that the robot does not stop. Therefore, there exists a non-empty set S of nodes that are visited infinitely

```
Exploration scheme X_c
    while the exploration is not completed do
        let v be the current node;
        {Internal moves:}
        if deg(v) ≥ 3 then
            unless Procedure DPDFS(v) has already been applied in
a previous
            step, apply it to get edges e, e' connecting v to the two
largest
            subtrees pending from v;
        {External move:}
        if there is only one edge connecting v to a subtree not com-
pletely
        explored then
            leave v by this edge;
        else
            {e, e' are the two edges connecting v to the two subtrees
            not yet completely explored}
            if during the last external move (if any), the robot did
not
            come to v by e or e' then
                leave v by edge e;
            else
                assume w.l.o.g. that the robot came to v by edge e;
                if the robot is for the first time at distance ⌊λD_0/2⌋
from the
                starting node then
                    leave v by edge e;
                else leave v by edge e';
    endwhile
```

Fig. 2. Exploration scheme $\mathcal{X}_c$

often. S is a subtree of T. Let v be a leaf of S, and let v' be its unique neighbor in S. Consider a stage t of the exploration, at which every node in S has been visited at least twice, and no node in $T \setminus S$ will be visited anymore.

After stage t, the robot goes (infinitely often) from v' to v, by external moves. The robot never performs an external move toward a subtree that is completely explored. Therefore, the subtree U of T, pending from v' and containing v, is not completely explored. Once the robot reaches v, it is not for the first time at distance $\lfloor \frac{\lambda D_0}{2} \rfloor$ from the starting node because v has been visited at least twice at stage t. Moreover, node v cannot be a leaf of T because subtree U is not completely explored. Therefore, the only reason why the robot goes back to v' from v is that the subtree pending from v and containing v' is the unique subtree pending from v that is not completely explored. This implies that subtree U is completely explored, a contradiction. Therefore, all nodes of

the tree are explored by the scheme $\mathcal{X}_c$, which proves the correctness of the exploration scheme. □

Theorem 5 *Let c be an arbitrary positive integer constant. Algorithm* `SKE`(c) *uses at most* $\max(1, \log\log D - c)$ *bits of advice, for any tree of diameter D, and has competitive ratio smaller than 2.*

PROOF. The result is true for $n = 1$ or $n = 2$, as any algorithm is optimal in this case. In the following, we assume that $n \geq 3$.

Recall that $k = \lfloor \frac{\lceil \log D \rceil}{2^{c+3}} \rfloor$. The oracle $\mathcal{O}_c$ gives at most $\lceil \log k \rceil + 1$ bits, hence at most $\max(1, \log\log D - c)$ bits. The definition of k implies the inequality $k \leq \frac{\lceil \log D \rceil}{2^{c+3}} < k + 1$, hence $k \cdot 2^{c+3} \leq \lceil \log D \rceil < k \cdot 2^{c+3} + 2^{c+3}$, and finally $2^{k \cdot 2^{c+3} - 1} \leq D < 2^{k \cdot 2^{c+3}} \cdot 2^{2^{c+3}}$. From the definition of D_0 and γ we get $D_0 \leq D < \gamma D_0$.

First assume that the oracle sets `choice` to 0. Since the exploration scheme in this case is a DFS, the cost of the scheme is at most $2(n-1) - 1 = 2n - 3$. The cost $opt(T, u)$, where u is the starting node, is $2(n-1) - ecc(u)$. We have $ecc(u) \leq D = (1 - \epsilon)n$. We obtain

$$opt(T, u) \geq 2(n-1) - (1-\epsilon)n = (1+\epsilon)n - 2 \ .$$

Since $D \leq n - 1$, we have $\epsilon \geq 1/n$, or equivalently $\epsilon n \geq 1$. Hence the ratio in this case is at most

$$\frac{2n-3}{(1+\epsilon)n-2} = \frac{2n-3}{(1+\epsilon/2)n - 2 + \epsilon n/2} \leq \frac{2n-3}{(1+\epsilon/2)n - 1.5} \leq \frac{2}{1+\epsilon/2} \ .$$

If $\epsilon \geq \epsilon^*$, then $\frac{2}{1+\epsilon/2} \leq \frac{2}{1+\epsilon^*/2} < 2$. Assume that $\epsilon < \epsilon^*$. Let $D^* = 2^{2^{c+3}}$. Therefore, `choice` is set to 0 because either $D < D^*$ or $n < N_0$. If $D < D^*$, then $n < \frac{D^*}{1-\epsilon}$. Let $N_1 = \frac{D^*}{1-\epsilon^*}$. Then we have $n < \frac{D^*}{1-\epsilon} \leq \frac{D^*}{1-\epsilon^*} = N_1$. Hence, both when $D < D^*$ and when $n < N_0$, we have $n < N^* = \max(N_0, N_1)$. Let $\epsilon_3 = 1/N^*$. We have $\epsilon \geq 1/n \geq 1/N^* = \epsilon_3$. We obtain $\frac{2}{1+\epsilon/2} \leq \frac{2}{1+\epsilon_3/2} < 2$. Hence the ratio of the cost of DFS (returned by Algorithm `SKE`(c) when `choice` is set to 0) to $opt(T, u)$, is at most $\max(\frac{2}{1+\epsilon^*/2}, \frac{2}{1+\epsilon_3/2}) < 2$.

From now on, we assume that the oracle sets `choice` to 1, hence Algorithm `SKE`(c) returns exploration scheme $\mathcal{X}_c$.

In the analysis of the cost of exploration scheme $\mathcal{X}_c$, we use the following terminology. Assume that the robot enters some node of degree at least 3 by edge e and applies Procedure `DPDFS`(v). If the procedure outputs two edges different from e, then we say that the current node v is a *fork*. Now consider edges traversed during external moves. These edges form a subtree T' of T.

For any node v, there exist at most two incident edges such that any external move of the robot leaving v takes one of them. Hence, all nodes are of maximal degree 3 in this subtree. Nodes of degree exactly 3 in T' are forks. Let $v_1, \ldots, v_q$ be the forks of T', if any, in order of their first visit by the robot. Let e_i be the edge connecting v_i to the subtree pending from it and containing the starting node u. In view of the definition of a fork, the robot never makes an external move on edge e_i from node v_i. Let u' be the last fork v_q, if any, or $u' = u$, if T' does not contain any fork. Finally, let P be a path of length D in the tree and let P' be the set of nodes in T' visited by an external move after u'. P' is a path because it does not contain any fork other than possibly u'. Finally, let C be the length of a shortest covering walk in path P' starting at node u'.

In our analysis, the cost of the scheme $\mathcal{X}_c$ is split into the cost of internal moves, and the cost of external moves.

Claim 1. The total number of internal moves is at most $154\epsilon n$.

To prove Claim 1, let v be any node of degree at least 3 in T. Let $T_1, \ldots, T_p$ be the enumeration of the subtrees pending from v, in decreasing order of their sizes. Since $D = (1-\epsilon)n$, there are at most ϵn nodes in $T \setminus P$. This implies $\sum_{i\geq 3}^{p} |T_i| \leq \epsilon n$. Hence `DPDFS`$(v)$ never goes at distance more than $2\epsilon n$ from node v. A node of T is either explored during a call to `DPDFS`, or it is in T'. Hence any node of T is at distance at most $2\epsilon n$ from a node in T'. In particular, there are two nodes w, w' of T', such that w is at distance at most $2\epsilon n$ from one extremity of P, and w' is at distance at most $2\epsilon n$ from the other extremity of P. The distance between w and w' is at least $D - 4\epsilon n = (1-5\epsilon)n$. This implies that $|T'| \geq (1-5\epsilon)n$.

By Lemma 3, the cost of `DPDFS`(v) depends on the sum of the sizes of the subtrees pending from v, except the two largest. We call those subtrees the small subtrees pending from v. `DPDFS` is executed exactly at these nodes in T' that have degree at least 3 in T. For any non-fork node $v \in T'$, the small subtrees of v are all disjoint from T'. For a fork v_i, there is only one small subtree that contains a node of T': the subtree pending from v_i and containing u. Therefore all small subtrees pending from nodes in T', except the q subtrees pending from the forks and containing u, are disjoint from T' and also disjoint from each other. The total size of these subtrees is at most $|T \setminus T'| \leq 5\epsilon n$.

Assume that $T' \setminus P' \neq \emptyset$. Let n_i, for $i = 1, \ldots, q$, be the size of the subtree T_i pending from v_i and containing u. Fix $i < q$. Since v_i is a fork, there exist two subtrees pending from v_i and not containing u, that have at least n_i nodes each. One of them, called T_i', does not contain the next fork v_{i+1}. T_i and T_i' are both included in the subtree T_{i+1}. This implies $n_{i+1} \geq 2n_i$. Hence the sum of the sizes of the q subtrees pending from the forks and containing u satisfies

$\sum_{i=1}^{q} n_i \leq 2n_q$. Since the edge e_q connecting P' and $T' \setminus P'$ is not selected by `DPDFS` at the corresponding fork $u' = v_q$, we have $|T' \setminus P'| \leq \epsilon n$. Therefore, since $n_q = |T' \setminus P'|$, we have $\sum_{i=1}^{q} n_i \leq 2\epsilon n$.

Hence, by Lemma 3, the total number of internal moves is at most $22(5\epsilon n + 2\epsilon n) = 154\epsilon n$. This concludes the proof of Claim 1.

Claim 2. The total number of external moves is at most $\beta_1 C + 2\epsilon n$.

To prove Claim 2, we study separately the external moves in P' and in $T' \setminus P'$. Let us first consider external moves in P'. At least one of w, w' (defined in the proof of Claim 1) is on P' because $(1 - 5\epsilon)n > \epsilon n$, as $\epsilon < \epsilon_1 < 1/8$. Therefore, since $|T' \setminus P'| \leq \epsilon n$, the path P' is of length at least $(1 - 6\epsilon)n$.

Since $D \geq 2^{2^{c+3}}$, we have $D_0 \geq 16$. Together with $\epsilon < 1/2$, this implies

$$2\epsilon n < 2\epsilon_1 n = \frac{\lambda}{8\gamma} n = \frac{\lambda}{8\gamma} \frac{D}{1-\epsilon} \leq \frac{\lambda}{8} \frac{D_0}{1-\epsilon} \leq \lambda D_0/4 \leq \lambda D_0/2 - 1 \leq \lfloor \lambda D_0/2 \rfloor \ .$$

Hence when the robot is for the first time at distance $\lfloor \lambda D_0/2 \rfloor$ from the starting node, it must be at a node of P'. Moreover, this node is at distance at least $\epsilon_1 |P'|$ from u' because $\epsilon_1 |P'| \leq \epsilon_1 D < \epsilon_1 n \leq 2\epsilon_1 n - \epsilon n$. It is also at distance at most $\lfloor \lambda |P'| \rfloor$ from u' because $|P'| \geq (1 - 6\epsilon)n$ and $\epsilon < 1/16$ imply $|P'| \geq n/2 > D_0/2$. Therefore, in the path P', the robot goes at probing distance x (from u') such that $\lfloor \epsilon_1 |P'| \rfloor \leq x \leq \lfloor \lambda |P'| \rfloor$, then returns to an endpoint of P', and finally returns to the other endpoint of P'. There exists α satisfying $0 < \epsilon_1 \leq \alpha \leq \lambda$, such that $x = \lfloor \alpha |P'| \rfloor$. Recall that C is the optimal cost of exploring the line P' starting at node u' on this line. In view of Lemma 2, the number of external moves of the robot in P' is at most $F_n(\alpha) \cdot C$. By the same lemma and in view of the fact that $n \geq N_0$, we have $F_n(\alpha) \leq F_n(\epsilon_1) \leq \beta_1 < 2$. Consequently, the number of external moves of the robot in P' is at most $\beta_1 \cdot C$.

Let us now consider external moves in $T' \setminus P'$. We proved that when the robot is for the first time at distance $\lfloor \lambda D_0/2 \rfloor$ from the starting node, then it is on P'. We also proved that the robot never visits again $T' \setminus P'$ by an external move after it reaches P' by an external move. In view of the formulation of scheme $\mathcal{X}_c$ (cf. Figure 2), the robot makes an external move on each edge of $T' \setminus P'$ at most twice. Since $|T' \setminus P'| \leq \epsilon n$, the number of external moves of the robot in $T' \setminus P'$ is at most $2\epsilon n$. This completes the proof of Claim 2.

Claim 1 and Claim 2 imply that the total cost of the scheme $\mathcal{X}_c$ is at most $\beta_1 \cdot C + (154 + 2)\epsilon n = \beta_1 \cdot C + 156\epsilon n$.

It remains to bound the ratio ρ of this cost to $opt(T, u)$. The shortest covering

walk starting at u visits u' before any other node of P'. It has then to visit the path P' starting from u'. Therefore, the length of the shortest covering walk starting at u cannot be less than C (the optimal number of moves on P' starting from u'). This gives $\rho \leq \frac{\beta_1 \cdot C + 156\epsilon n}{C}$. We have $\epsilon \leq \epsilon_2 = \frac{2-\beta_1}{624}$. Together with $C \geq |P'| \geq n/2$ (for the latter inequality, see the proof of Claim 2), this implies

$$\beta_1 + \frac{156\epsilon n}{C} \leq \beta_1 + \frac{156\epsilon n}{n/2} \leq \beta_1 + 2 \cdot 156 \frac{2-\beta_1}{624} = \beta_1 + \frac{2-\beta_1}{2} = 1 + \frac{\beta_1}{2} < 2 \ .$$

It follows from the above obtained estimates that the competitive ratio of Algorithm $\texttt{SKE}(c)$ is at most

$$\max\left(\frac{2}{1+\epsilon^*/2}, \frac{2}{1+\epsilon_3/2}, 1 + \frac{\beta_1}{2}\right) < 2 \ ,$$

which completes the proof of the theorem. □

4 The lower bound

This section is devoted to establishing a lower bound on the number of bits of advice for which there exists an algorithm with competitive ratio smaller than 2. This lower bound exactly matches the upper bound shown previously, and it holds even for the class of lines. Indeed, we show that if, for all lines L_k of diameter (i.e., length) $k \leq n$, the number of bits of advice is smaller than $\log \log n$, and differs from it by an unbounded number of bits, then every algorithm has competitive ratio at least 2.

Theorem 6 *Let $\mathcal{O}$ be an oracle and let $f(n)$ denote the maximum of sizes of $\mathcal{O}(L_k)$, for $k \leq n$. Let $g : \mathbb{N} \mapsto \mathbb{R}$ be defined by the formula $f(n) = \log \log n - g(n)$. If g is a function unbounded from above, then every exploration algorithm using oracle $\mathcal{O}$ has competitive ratio at least 2.*

PROOF. We will use the following claim.

Claim 3. For every positive integers M and γ, there exist integers $n_1 > n_2 \geq M$, such that $\mathcal{O}(L_{n_1}) = \mathcal{O}(L_{n_2})$ and $n_1/n_2 \geq \gamma$.

Suppose that the claim does not hold. Take M and γ that refute it. Let $\psi : \{n : \ n > M\} \mapsto \mathbb{R}$ be the sequence defined by the formula $\psi(n) = \frac{\log \gamma \log n}{\log n - \log M}$. The sequence ψ converges to $\log \gamma$, hence it is bounded. Let A be such that

$\psi(n) < A$ for all n. Since g is an unbounded function, there exists $n_0 > M$ for which $g(n_0) > \log A$. Let x be the size of the set $\{\mathcal{O}(L_k) : k \leq n_0\}$. We have

$$x \leq 2^{f(n_0)} = 2^{\log\log n_0 - g(n_0)} < 2^{\log\log n_0 - \log A} < 2^{\log\log n_0 - \log \frac{\log\gamma \log n_0}{\log n_0 - \log M}}$$

and therefore $x < \frac{\log n_0 - \log M}{\log \gamma}$. All integers k with $\mathcal{O}(L_k) = \mathcal{O}(L_{M\gamma^i})$ must be smaller than $M\gamma^{i+1}$, for $i \geq 0$. Hence all oracle values for lines $L_{M\gamma^i}$ are distinct, and there are x such values. We have $n_0 < M\gamma^x$ because $\mathcal{O}(L_{n_0}) = \mathcal{O}(L_{M\gamma^i})$, for some $i < x$, and hence $n_0 < M\gamma^{i+1} \leq M\gamma^x$. Consequently, $\log n_0 \leq \log M + x \log\gamma < \log M + \log n_0 - \log M$. This contradiction proves Claim 3.

We will now show that any algorithm using oracle $\mathcal{O}$ must have competitive ratio at least 2. In view of Lemma 1 it is enough to restrict attention to algorithms producing exploration schemes of type 1 for the class of lines. The probing distance of such a scheme for line L_n depends only on $\mathcal{O}(L_n)$. Consider an algorithm $\mathcal{A}$ producing a scheme of type 1 with probing distance $\phi(\mathcal{O}(L_n))$. Fix any constant $3/2 < \beta < 2$. Choose γ such that $\frac{2\gamma}{\gamma+2} > \beta$ and M such that $\frac{2M-1}{M+1} > \beta$. Hence $\gamma > 6$. Let $n_1 > n_2 \geq M$ be integers for which $\mathcal{O}(L_{n_1}) = \mathcal{O}(L_{n_2})$ and $n_1 \geq \gamma n_2$. Their existence is guaranteed by Claim 3. Let $y = \phi(\mathcal{O}(L_{n_1}))$. Hence the scheme makes the first change of direction after y steps, both in L_{n_1} and in L_{n_2}, unless an endpoint is encountered earlier. Consider two cases.

If $y \leq n_2$ then consider the behavior of $\mathcal{A}$ on L_{n_1}, with the starting node u at distance $y+1$ from the endpoint toward which the robot starts. Since $\gamma > 6$, this is the endpoint closer to u. Then

$$\frac{\mathcal{A}(L_{n_1}, u)}{opt(L_{n_1}, u)} = \frac{y + 2n_1 - 1}{y + n_1 + 1} \geq \frac{n_2 + 2n_1 - 1}{n_2 + n_1 + 1}$$
$$\geq \frac{(2\gamma+1)n_2 - 1}{(\gamma+1)n_2 + 1} \geq \frac{(2\gamma+1) - 1}{(\gamma+1) + 1} = \frac{2\gamma}{\gamma+2} > \beta \ .$$

If $y > n_2$ then consider the behavior of $\mathcal{A}$ on L_{n_2} with the starting node u at distance $n_2 - 1$ from the endpoint toward which the robot starts. Then

$$\frac{\mathcal{A}(L_{n_2}, u)}{opt(L_{n_2}, u)} = \frac{2n_2 - 1}{n_2 + 1} \geq \frac{2M - 1}{M + 1} > \beta \ .$$

This proves that the competitive ratio of algorithm $\mathcal{A}$ is at least 2. $\square$

5 Exploration knowing the diameter

We have shown in Section 3 that very little information (less than $\log \log D$ bits of advice) is needed to beat competitive ratio 2, and in fact, most of this information (all bits except one) concerns the value of the diameter D itself, and is used to establish a lower bound on it. This extra bit, however, cannot be deduced from D alone, and turns out to be crucial. In this section we prove a surprising result that even an algorithm that knows D *exactly* (i.e., is provided with all $\lceil \log D \rceil$ bits of it), but does not have any additional knowledge, cannot beat competitive ratio 2. Notice that a similar argument proves that the exact knowledge of the number n of nodes, with no extra information, is not enough for this purpose either.

Theorem 7 *Let $\mathcal{A}$ be any tree exploration algorithm that, for every tree T, is given the diameter of T as input. Then $\mathcal{A}$ has competitive ratio at least 2.*

PROOF. Consider any exploration algorithm $\mathcal{A}$ that knows only the diameter D of the explored tree. Fix D and let S be the exploration scheme returned by $\mathcal{A}$ for input D. Recall that we can restrict attention to regular schemes only. We construct a tree T of diameter D that will be used to prove a lower bound on the competitive ratio of the algorithm. Suppose that the robot follows scheme S.

The construction proceeds in phases. Inductively, after phase $i-1$ is terminated, for $i \geq 2$, there is a node of degree 3, called v_i, having a neighbor w_i with the subtree rooted at w_i already constructed. The two other edges incident to v_i are pending and the construction in phase i continues starting from them. In phase i, a line attached at node v_i is appended to the subtree constructed previously. Phase 1 starts with the previously constructed part consisting only of the starting node $v = v_1$ with two edges pending. The line appended in phase i has two *sides*, corresponding to the two edges pending at v_i. Call *side 1* the side of v_i in the direction of the first move of the robot in phase i, and call *side 2* the other side of v_i. By the inductive hypothesis, the subtree rooted at w_i is completely explored at the end of phase $i-1$, hence by the regularity of the scheme, the robot does not enter this subtree in phase i.

Phase i is described as follows. (See also Figure 3.) The robot starts at v_i in some direction (on side 1) and, while seeing only nodes of degree 2 on its way, either goes indefinitely without return, or returns after x_i steps. In the first case we say that $x_i = \infty$. In this case, call it *Case 1*, we finish the construction of the tree: a line of length $m+1$ is appended, with the node v_i at distance 1 from the endpoint on side 2. The integer m is adjusted so that the diameter of the tree is exactly D. If the robot returns after x_i steps, we distinguish two

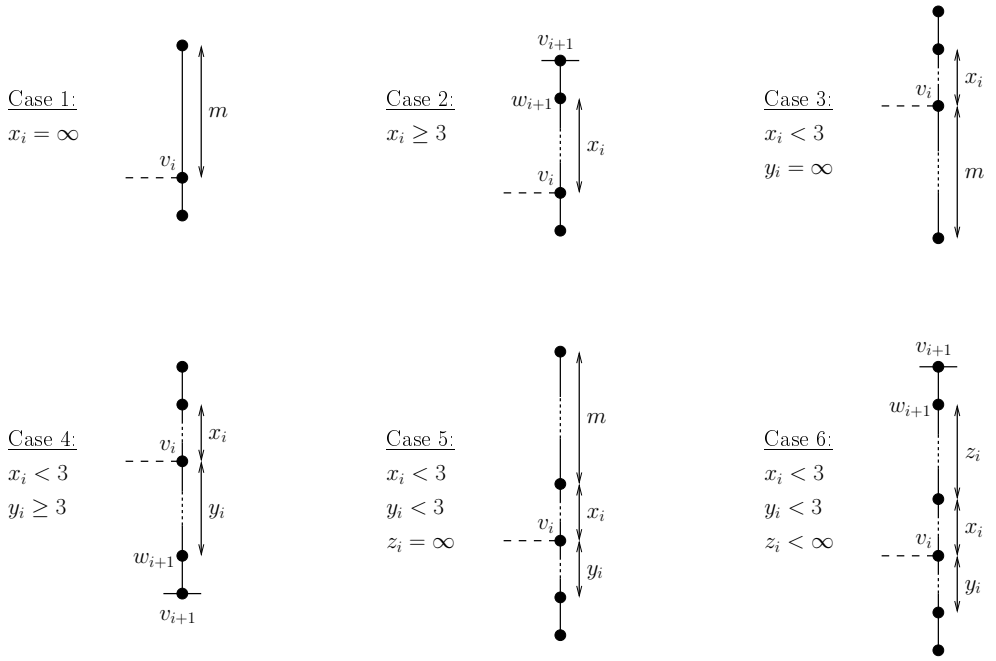

Fig. 3. The different cases and corresponding constructions used to prove Theorem 7

further cases: $x_i \geq 3$ and $x_i < 3$. If $x_i \geq 3$, call it *Case 2*, the construction is continued as follows. We append a line of length x_i+2, with node v_i at distance 1 from the endpoint on side 2, and the other endpoint replaced by a node v_{i+1} of degree 3, at which the construction will be continued in phase $i+1$. This is done unless appending such a line would exceed the diameter D of the tree, in which case we proceed as in Case 1. If $x_i < 3$ then the robot returns and goes on side 2 of node v_i. While seeing only nodes of degree 2 on its way, it either goes indefinitely without return, or returns after making y_i steps on side 2 of v_i. In the first situation, call it *Case 3*, we finish the construction of the tree by appending a line of length $m + x_i + 1$, with node v_i at distance $x_i + 1$ from the endpoint on side 1. The integer m is adjusted so that the diameter of the tree is exactly D. If the robot returns after y_i steps on side 2 of v_i, we distinguish two further cases: $y_i \geq 3$ and $y_i < 3$. If $y_i \geq 3$, call it *Case 4*, the construction is continued as follows. We append a line of length $x_i + y_i + 2$ with node v_i at distance $x_i + 1$ from the endpoint on side 1, and with the other endpoint replaced by a node v_{i+1} of degree 3, at which the construction will be continued in phase $i + 1$. This is done unless appending such a line would exceed the diameter D of the tree, in which case we proceed as in Case 3. If $y_i < 3$, consider two further cases: *Case 5* in which the robot goes indefinitely after the second return, assuming that it sees only nodes of degree 2 on its way, and *Case 6* if it returns after some number $x_i + z_i$ of steps on side 1 of v_i, under this assumption. In Case 5 we finish the construction of the tree by appending a line of length $m + x_i + y_i + 1$ with node v_i at distance $y_i + 1$ from the endpoint on side 2. The integer m is adjusted so that the diameter of the tree is exactly D. In Case 6 the construction is continued as follows.

We construct a line of length $x_i + y_i + z_i + 2$, with node v_i at distance $y_i + 1$ from the endpoint on side 2, and with the other endpoint replaced by a node v_{i+1} of degree 3, at which the construction will be continued in phase $i + 1$. This is done unless appending such a line would exceed the diameter D of the tree, in which case we proceed as in Case 5. This concludes the description of phase i of the construction. Hence, in every phase except the last, one of the Cases 2, 4, 6, occurs, and in the last phase one of the Cases 1, 3, 5, occurs.

Denote by P_i the part of the tree constructed in phase i. Hence all parts P_i are pairwise edge-disjoint. Since $P_1 \cup \cdots \cup P_{i-1}$ is entirely explored by the end of phase $i - 1$, the robot does not enter this subtree, in view of the regularity of the scheme. Hence in phase i the robot moves only in part P_i. Let S_i be the number of moves performed by the robot in phase i, and let opt_i be the length of the shortest covering walk of part P_i.

In Case 1 we have $S_i \geq 2m+1$ and $opt_i \leq m+2$. In Case 2 we have $S_i \geq 3x_i+3$ and $opt_i \leq x_i+3$. In Case 3 we have $S_i \geq 3x_i+2m+1$ and $opt_i \leq 2x_i+m+2$. In Case 4 we have $S_i \geq 4x_i + 3y_i + 3$ and $opt_i \leq 2x_i + y_i + 3$. In Case 5 we have $S_i \geq 4x_i + 3y_i + 2m + 1$ and $opt_i \leq x_i + 2y_i + m + 2$. In Case 6 we have $S_i \geq 5x_i + 4y_i + +3z_i + 3$ and $opt_i \leq x_i + 2y_i + z_i + 3$.

Fix any $\alpha < 2$. We now prove that, for a sufficiently large D, the ratio $\mathcal{A}(T, v)/opt(T, v)$ exceeds α. Suppose that the last phase has number i and let $Z = S_1 + \cdots + S_{i-1}$. In phases $j < i$ only Cases 2, 4, 6, can occur. Since in Case 2 we have $x_j \geq 3$, it follows that $S_j/opt_j \geq 2$. Since in Case 4 we have $y_j \geq 3$, it follows that $S_j/opt_j \geq 2$. Since in Case 6 we have $x_j \geq 1$, it follows that $S_j \geq 2x_j + 4y_j + 3z_j + 6$, and hence $S_j/opt_j \geq 2$. Thus we can conclude that $Z \geq 2(opt_1 + \cdots + opt_{i-1})$.

Since in Case 3 we have $x_i < 3$ and in Case 5 we have $x_i, y_i < 3$, it follows that in each of the Cases 1, 3, 5, we have $S_i \geq 2m + a$ and $opt_i \leq m + b$, for some constants a, b. Hence $\mathcal{A}(T, v) = S_1 + \cdots + S_i \geq Z + 2m + a$ and $opt(T, v) = opt_1 + \cdots + opt_i \leq Z/2 + m + b$. It follows that

$$\frac{\mathcal{A}(T, v)}{opt(T, v)} \geq \frac{Z + 2m + a}{Z/2 + m + b}.$$

On the other hand, Z is at least the size of $P_1 \cup \cdots \cup P_{i-1}$ and $m + 6$ is at least the size of P_i, hence $Z + m + 6 \geq D$. Consequently, $\frac{\mathcal{A}(T,v)}{opt(T,v)}$ exceeds α, for sufficiently large D. $\square$

References

[1] S. Albers and M. R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (2000), 1164-1188.

[2] B. Awerbuch, M. Betke, R. Rivest and M. Singh, Piecemeal graph learning by a mobile robot, Proc. 8th Conf. on Comput. Learning Theory (1995), 321-328.

[3] E. Bar-Eli, P. Berman, A. Fiat and R. Yan, On-line navigation in a room, Journal of Algorithms 17 (1994), 319-341.

[4] M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Information and Computation 176 (2002), 1-21.

[5] M.A. Bender and D. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, Proc. 35th Ann. Symp. on Foundations of Computer Science (FOCS 1994), 75-85.

[6] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen and M. Saks, Randomized robot navigation algorithms, Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA 1996), 74-84.

[7] A. Blum, P. Raghavan and B. Schieber, Navigating in unfamiliar geometric terrain, SIAM Journal on Computing 26 (1997), 110-137.

[8] M. Betke, R. Rivest and M. Singh, Piecemeal learning of an unknown environment, Machine Learning 18 (1995), 231-254.

[9] A.E.F. Clementi, A. Monti and R. Silvestri, Selective families, superimposed codes, and broadcasting on unknown radio networks, Proc. 12th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), 709-718.

[10] A. Czumaj and W. Rytter, Broadcasting algorithms in radio networks with unknown topology, Proc. 44th Ann. Symposium on Foundations of Computer Science (FOCS 2003), 492-501.

[11] X. Deng, T. Kameda and C. H. Papadimitriou, How to learn an unknown environment I: the rectilinear case, Journal of the ACM 45 (1998), 215-245.

[12] X. Deng and A. Mirzaian, Competitive robot mapping with homogeneous markers, IEEE Transactions on Robotics and Automation 12 (1996), 532-542.

[13] X. Deng and C. H. Papadimitriou, Exploring an unknown graph, Journal of Graph Theory 32 (1999), 265-297.

[14] A. Dessmark and A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (2004), 343-362.

[15] K. Diks, P. Fraigniaud, E. Kranakis and A. Pelc, Tree exploration with little memory, Journal of Algorithms 51 (2004), 38-63.

[16] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, Robotic exploration as graph construction, IEEE Transactions on Robotics and Automation 7 (1991), 859-865.

[17] V. Dujmović and S. Whitesides, On validating planar worlds, Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), 791-792.

[18] C.A. Duncan, S.G. Kobourov and V.S.A. Kumar, Optimal constrained graph exploration, Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2001), 807-814.

[19] F. Fich and E. Ruppert. Hundreds of impossibility results for distributed computing, Distributed Computing, 16 (2003), 121–163.

[20] R. Fleischer and G. Trippen, Exploring an unknown graph efficiently, Proc. 13th Ann. European Symposium on Algorithms (ESA 2005), LNCS 3669, 11-22.

[21] P. Flocchini, B. Mans and N. Santoro. Sense of direction in distributed computing, Theoretical Computer Science 291 (2003), 29-53.

[22] P. Fraigniaud, C. Gavoille, D. Ilcinkas and A. Pelc, Distributed computing with advice: information sensitivity of graph coloring, Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007), LNCS 4596, 231-242.

[23] P. Fraigniaud, D. Ilcinkas and A. Pelc. Oracle size: a new measure of difficulty for communication tasks, Proc. 25th Ann. ACM Symposium on Principles of Distributed Computing (PODC 2006), 179-187.

[24] P. Fraigniaud, A. Korman and E. Lebhar, Local MST computation with short advice, Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2007), 154-160.

[25] L. Gasieniec, D. Peleg and Q. Xin, Faster communication in known topology radio networks, Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC 2005), 129-137.

[26] T. Kameda and M. Yamashita. Computing on anonymous networks: Part I – characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems, 7 (1996), 69-89.

[27] D. Kowalski and A. Pelc, Optimal deterministic broadcasting in known topology radio networks, Distributed Computing 19 (2007), 185-195.

[28] N. Lynch. A hundred impossibility proofs for distributed computing. Proc. 8th Ann. ACM Symposium on Principles of Distributed Computing (PODC 1989),1-28.

[29] P. Panaite and A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (1999), 281-295.

[30] P. Panaite and A. Pelc, Impact of topographic information on graph exploration efficiency, Networks, 36 (2000), 96-103.

[31] C. H. Papadimitriou and M. Yannakakis, Shortest paths without a map, Theoretical Computer Science 84 (1991), 127-150.

[32] D. Soguet and N. Nisse, Graph searching with advice, Proc. 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2007), LNCS 4474, 51-65.