Navigating in Trees with Permanently Noisy Advice

Lucas Boczkowski, Uriel Feige, Amos Korman, Yoav Rodeh

Navigating in Trees with Permanently Noisy Advice *

Lucas Boczkowski[1], Uriel Feige[2], Amos Korman[1], and Yoav Rodeh[3]

[1]CNRS, IRIF, Univ Paris Diderot, Paris, France.
[2]The Weizmann Institute of Science, Rehovot, Israel.
[3]Ort-Braude College, Karmiel, Israel.

Abstract

We consider a search problem on trees in which an agent starts at the root of a tree and aims to locate an adversarially placed treasure, by moving along the edges, while relying on local, partial information. Specifically, each node in the tree holds a pointer to one of its neighbors, termed *advice*. A node is faulty with probability q. The advice at a non-faulty node points to the neighbor that is closer to the treasure, and the advice at a faulty node points to a uniformly random neighbor. Crucially, the advice is *permanent*, in the sense that querying the same node again would yield the same answer.

Let Δ denote the maximum degree. For the expected number of moves (edge traversals) we show that a phase transition occurs when the *noise parameter* q is roughly $1/\sqrt{\Delta}$. Below the threshold, there exists an algorithm with expected number of moves $\mathcal{O}(D\sqrt{\Delta})$, where D is the depth of the treasure, whereas above the threshold, every search algorithm has expected number of moves which is both exponential in D and polynomial in the number of nodes n.

In contrast, if we require to find the treasure with probability at least $1-\delta$, then for every fixed $\varepsilon > 0$, if $q < 1/\Delta^{\varepsilon}$ then there exists a search strategy that with probability $1-\delta$ finds the treasure using $(\delta^{-1}D)^{O(\frac{1}{\varepsilon})}$ moves. Moreover, we show that $(\delta^{-1}D)^{\Omega(\frac{1}{\varepsilon})}$ moves are necessary.

*This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648032). This work was also supported in part by the Israel Science Foundation (grant No. 1388/16). A preliminary version of this paper appeared in ESA 2018. The current journal version contains many additional results.

Contents

1 Introduction

This paper considers a navigation problem on trees, in which an agent moves on the edges of a tree, aiming to find a treasure that is placed at one of the nodes by an adversary. Each node of the tree holds information, called *advice*. The advice, if correct, specifies which neighbor of the node is closer to the treasure. However, with some small probability, the advice is faulty and points to an incorrect neighbor. The agent may query the advice at its node location in order to accelerate the search.

Searching with advice on trees is an extension of binary search to tree topologies. This type of extension has been the focus of numerous works [11, 12, 15, 16, 27, 28], some including noise or errors in the advice. The problem may also be viewed as searching a poset [28, 27], instead of a completely ordered set as in typical binary search. Some authors also motivate the problem using the notion of "bug detection", where the tree models dependencies between programs [27]. When the searcher is restricted to walk on the edges of the underlying graph,

it is possible to view the problem as a routing problem with unreliable local information [19, 21, 20]. An interesting application was also given in [14], in the context of interactive learning.

The crucial feature of our model, that distinguishes it from most existing literature on search with noisy advice, is the permanent nature of the faults. Given the tree and the location of the treasure, there is a sampling procedure (which may be partly controlled by an adversary) that determines the advice at every node of the tree. Depending on the outcome of the sampling procedure, the advice at a node may either be correct or faulty (we also refer to the latter case as noise). The advice is *permanent* – it does not change after the sampling stage. Every query to a given node yields the same advice – there is no re-sampling of advice. The difference between permanent noise and re-sampled one (as in e.g., [3, 15, 16, 24]) is dramatic, since the re-sampled advice model allows algorithms to boost the confidence in any given piece of advice by repeatedly querying the same advice. Permanent noise was considered in [7] for the task of sorting, but this task is very different than the search task considered in our paper (in particular, no algorithm can find the true sorted order when noise is permanent). Searching with permanent faulty nodes has also been studied in a number of works [8, 17, 20, 21, 22], but assuming that the faulty nodes are chosen by an adversary. The difference between such worst case scenarios and the probabilistic version studied here is again significant, both in terms of results and in terms of techniques (see more details in Section 1.3).

The model of permanent faults aims to model faults that occur in the physical memory associated with the node, rather than, for example, the noise that is associated with the actual mechanism behind the query. Interestingly, the topic of noisy permanent advice is also meaningful outside the realm of classical computer science, and was shown to be relevant in the context of ant navigation [19]. The authors therein conducted experiments in which a group of ants carry a large load of food aiming to transport it to their nest, while basing their navigation on unreliable advice given by pheromones that are laid on the terrain. Indeed, although the directions proposed by pheromones typically lead to the nest, trajectories as experienced by small ants may be inaccessible to the load, and hence directional cues left by ants sometimes lead the load towards dead-ends.

The current paper introduces the algorithmic study of search with permanent probabilistically noisy advice. Similarly to many other works on search we focus on trees, which is a very important topological structure in computer science. Extending our work to general graphs seems technically challenging and remains for future work, see Section 7.

1.1 The Noisy Advice Model

We start with some notation. Additional notation is introduced in Section 1.4. We present the model for trees, but we remark that the definitions can be extended to general graphs (see also Section 7). Let T be an n-node tree rooted at some arbitrary node σ. The *distance* $d(u, v)$ is the number of edges on the path between u and v. The *depth* of a node u is $d(u) = d(\sigma, u)$. Let $d = d(\tau)$ denote the depth of τ, and let the depth D of the tree be the maximum depth of a node. Finally, let Δ_u denote the degree of node u and let Δ denote the maximum degree in the tree. For an integer $\Delta \geq 2$, a *complete* Δ*-ary* tree is a tree such that every internal node has degree precisely Δ.

We consider an agent that is initially located at the root σ of T, aiming to find a node τ,

called the *treasure*, which is chosen by an adversary. That is, the goal of the agent is to be located at τ, and once it is there, the algorithm terminates. The agent can move by traversing edges of the tree. At any time, the agent can query its hosting node in order to "see" the corresponding advice and to detect whether the treasure is present there. The search terminates when the agent queries the treasure. For the purpose of upper bounds, we consider the *move complexity*, which is the number of edge traversals (where the same edge might be counted multiple times, due to backtracking). For lower bound purposes, we consider the *query complexity*, which is the number of queries made. As there is no point of querying the same node twice, the number of queries is not larger than the number of distinct nodes visited, and hence is at most one larger than the number of moves. We do not address in this paper query complexity upper bounds, though the interested reader may find such upper bounds in an expanded version of this paper [4].

Each node $u \neq \tau$ is assumed to be provided with an *advice*, termed $\texttt{adv}(u)$, which provides information regarding the direction of the treasure. Specifically, $\texttt{adv}(u)$ is a pointer to one of u's neighbors. It is called *correct* if the pointed neighbor is one step closer to the treasure than u is. Each node $u \neq \tau$ is *faulty* with probability q (the meaning of being faulty will soon be explained), independently of other nodes. Otherwise, u is considered *sound*, in which case its advice is correct. We call q the *noise parameter*. Unless otherwise stated, this parameter is the same across all nodes, but in some occasions, we also allow it to vary across nodes. In that case q is defined as $\max_u(q_u)$.

Random and semi-adversarial variants. We consider two models for faulty nodes. The main model assumes that the advice at a faulty node points to one of its neighbors chosen uniformly at random (and so possibly pointing at the correct one). We also consider an adversarial variant, called the *semi-adversarial model*, where this neighbor is chosen by an adversary. The adversary may either be *oblivious* or *adaptive*. An oblivious adversary first decides on adversarial advice for each node, afterwards each node becomes faulty independently with probability q, and then the true advice of faulty nodes is replaced by the respective adversarial advice. An adaptive adversary first sees the locations of all faulty nodes and only afterwards decides on the advice at the faulty nodes.

Noise assumption. The noise parameter q governs the accuracy of the environment. If $q = 0$ for all nodes, then advice is always correct. This case allows to find the treasure in D moves, by simply following each encountered advice. On the other extreme, if $q = 1$, then advice is essentially meaningless, and the search cannot be expected to be efficient. An intriguing question is therefore to identify the largest value of q that allows for efficient search.

Expectation and high probability. Importantly, we consider two kinds of guarantees: expectation, and high probability. In the first case, we measure the performance of the algorithm as the expected number of moves before the treasure is found. Expectation is taken over both the randomness involved in sampling the noisy advice, and over the possible probabilistic choices made by the search algorithm. In the second case, we consider the number of moves spent by algorithms that find the treasure with high probability (say, probability 0.9). An upper bound on expectation can be converted into a high probability upper bound, by use of the Markov inequality. However, the converse need not hold. Indeed, as our work shows, in our setting the two kind of guarantees lead to quite different thresholds and techniques.

A simple example that demonstrates this difference is the star graph. If the center node is faulty with some small probability q, then finding the treasure requires $\Omega(qn)$ moves in expectation, but can be done in at most 2 moves, with probability $1 - q$.

Full-information model. For lower bound purposes, we find it instructive to also consider the following *full-information* model. Here the structure of the tree is known, the algorithm is given as input the advice of all nodes except for the leaves, and the treasure is at one of the leaves. The queried node can be an arbitrary leaf, and the answer reveals whether the leaf holds the treasure.

1.2 Our Results

We introduce the algorithmic study of search problems with probabilistic permanent faulty advice. The results all assume the underlying graph is a tree.

Our results are grouped according to the convergence guarantee, which can refer to either *expectation* or *high probability*. As is always the case with non-negative random variables, the median cannot be much larger than the average, but it might be much smaller. Our results imply that in the context of searching with noisy permanent advice, in a large regime of noise, there is an exponential gap between the median and the average. The high expectation running time is the consequence of a small fraction of the possible error patterns (the pattern of errors in the advice) for which the search is very slow, but for almost all error patterns, the treasure is found much faster than what the high expectation suggests. We start with the average case, i.e., fast convergence in expectation.

Expected move complexity: Phase transition phenomena. Consider the noisy advice model on trees with maximum degree Δ. Roughly speaking, we show that $1/\sqrt{\Delta}$ is the threshold for the noise parameter q, in order to obtain search algorithms with low expected move complexity. Essentially, above the threshold, there exists trees (specifically, complete Δ-ary trees) such that for any algorithm, the expected number of moves required to find the treasure is exponential d, the depth of the treasure. Conversely, below the threshold there exists an algorithm whose expected move complexity is almost linear, that is, $\mathcal{O}(d\sqrt{\Delta})$.

The proof that there is no algorithm with a small expected number of moves when the noise exceeds $1/\sqrt{\Delta - 1}$ is rather simple. In fact, it holds even in the full information model. Intuitively, the argument is as follows (the formal proof appears in Section 3). Consider a complete Δ-ary tree of depth D and assume that the treasure τ is placed at a leaf. The first observation is that the expected number of leaves having more advice point to them than to τ is a lower bound on the move complexity. The next observation is that there are more than $(\Delta - 1)^D$ leaves whose distance from τ is $2D$, and for each of those leaves u, the probability that more advice points towards it than towards τ can be approximated by the probability that all nodes on path connecting u and τ are faulty. As this latter probability is q^{2D}, the expected number of leaves that have more pointers leading to them is roughly $(\Delta - 1)^D q^{2D}$. This term explodes when $q > 1/\sqrt{\Delta - 1}$.

One of the main challenges we faced in the paper was show that for noise probability below $1/\sqrt{\Delta - 1}$ (by a constant factor) the lower bound no longer holds, and in fact, there are extremely efficient algorithms. Interestingly, the optimal algorithm we present is based on a Bayesian approach, which assumes the treasure location is random, yet it works even

under worst case assumptions. The challenging part in the construction was identifying the correct prior. Constructing algorithms that ensure worst-case guarantees through a Bayesian approach was done in [3] which studies a closely related, yet much simpler problem of search on the line. Apart from [3] we are unaware of other works that follow this approach.

We also analyze the oblivious semi-adversarial model, and show that the expected move complexity has a threshold also in this model, but it is much lower, around $1/\Delta$.

High probability guarantee. We then turn our attention to studying the move complexity under a given probability guarantee. We show that for every fixed $\varepsilon > 0$, if $q < 1/\Delta^{\varepsilon}$ then there exists a search strategy that with probability $1 - \delta$ finds the treasure using $(\delta^{-1}d)^{O(\frac{1}{\varepsilon})}$ moves. Moreover, we show that $(\delta^{-1}d)^{\Omega(\frac{1}{\varepsilon})}$ moves are necessary. The upper bounds hold even in the adaptive semi-adversarial variant, whereas the lower bound holds even in the purely randomized variant.

The key concept towards proving the upper bound is a notion of node *fitness*. Essentially, a node is declared fit if it has many pointers to it on the path coming from the root. This is good evidence that the node is either on the path to the treasure, or at least not too far from it. The idea is to explore the component of fit nodes to which the root, i.e., the starting point, belongs. If the component contains the treasure, the nodes on the root to treasure path, and not too many additional nodes, then the treasure is found quickly. With the appropriate formalization of fitness, efficient search can be achieved with high probability.

Unlike the lower bound when considering the expected number of moves, the lower bound in the high probability case uses the fact that obtaining the advice of nodes requires spending queries. Hence this lower bound does not hold in the full information model. Consider the complete Δ-ary tree of depth D, and assume that the treasure is at a leaf. Set $q = \Delta^{-\varepsilon}$ and $h = \varepsilon^{-1} \log_{\Delta}(D/\delta)$. Consider the length D path from the root to the treasure. On this path, with probability at least δ, there exists a segment of length h, where all nodes are faulty. Let us denote by H the subtree rooted at the highest endpoint of such a segment of h consecutive faulty nodes. The algorithm needs to explore at least a constant fraction of H before finding how to proceed towards the treasure. The lower bound follows as the size of the subtree H is $\Delta^h = (D/\delta)^{\varepsilon^{-1}}$.

Figure 1 summarizes the results presented in this paper regarding walking algorithms.

1.2.1 Results in Expectation

In Section 2 we present an algorithm whose expected move complexity is optimal up to a constant factor for the regime of noise below the threshold. Furthermore, this algorithm does not require prior knowledge of either the tree's structure, or the values of Δ, q, d, or n.

Before presenting the result, we extend the model slightly, by allowing each node v to have a distinct noise parameter q_v. This greater flexibility makes our results stronger. It also happens to be convenient from a technical standpoint. When q_v does not depend on v, we say the noise is *uniform*. The following technical definition is used in our results, in place of the more crude $q \ll \frac{1}{\sqrt{\Delta}}$ given in Table 1.

Definition 1.1. *Condition ($\star$) holds with parameter $0 < \varepsilon < 1$ if for every node v, we have*

$$q_v < \frac{1 - \varepsilon - \Delta_v^{-\frac{1}{4}}}{\sqrt{\Delta_v} + \Delta_v^{\frac{1}{4}}}. \tag{1}$$

	Upper Bound		Lower Bound	
	Regime	Moves	Regime	Moves
Expectation	$q \ll \frac{1}{\sqrt{\Delta}}$	$\mathcal{O}(d\sqrt{\Delta})$	$q \gg \frac{1}{\sqrt{\Delta}}$	$e^{\Omega(d)}$
Expectation (S.A.)	$q \ll \frac{1}{\Delta}$	$\mathcal{O}(d)$	$q \gg \frac{1}{\Delta}$	$e^{\Omega(d)}$
High Probability	$q < \Delta^{-\varepsilon}$	$d^{O(\varepsilon^{-1})}$	$q > \Delta^{-\varepsilon}$	$d^{\Omega(\varepsilon^{-1})}$

Figure 1: A summary of our results, in a simplified form. The precise definition of the symbol $\ll$ will be clarified later. S.A. stands for oblivious semi-adversarial. The High Probability upper bound includes the adaptive semi-adversarial model.

Since $\Delta_v \geq 2$, the condition is always satisfiable when taking a small enough ε.

All our algorithms are deterministic, hence, expectation is taken with respect only to the sampling of the advice.

Theorem 1.2. *For every $\varepsilon > 0$, if Condition ($\star$) holds with parameter ε, then there exists a deterministic algorithm $\mathtt{A}_{walk}$ that requires $\mathcal{O}(\sqrt{\Delta}d)$ moves in expectation. The algorithm does not require prior knowledge of either the tree's structure, or any information regarding the values of Δ, d, n, or the q_v's.*

In the above theorem (and some other places in this paper) the $\mathcal{O}$ notation hides terms that depends on ε. For Theorem 1.2, this hidden term is ε^{-3}.

In Section 3 we establish the following lower bound.

Theorem 1.3. *Consider a complete Δ-ary tree of depth D, and assume that the treasure is at a leaf. For every constant $\varepsilon > 0$, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}}$, then every randomized search algorithm has move (and query) complexity that in expectation is exponential in D. The result holds also in the full-information model.*

Observe that taken together, Theorems 1.2, 1.3 and Condition ($\star$) (see Eq. (1)) imply that for every given $\varepsilon > 0$ and large enough Δ, efficient search can be achieved if $q < (1-\varepsilon)/\sqrt{\Delta}$ but not if $q > (1+\varepsilon)/\sqrt{\Delta}$.

We further complete our lower bounds with the following result, proved in Section 3.1.2.

Theorem 1.4. *For a complete Δ-ary tree of depth D, the expected number of queries for every algorithm is $\Omega(q\Delta D)$ (or equivalently, $\Omega(q\Delta \log_\Delta n)$).*

For $q \sim \sqrt{\Delta}$ the lower bound in Theorem 1.4 is $\Omega(\sqrt{\Delta}D)$, implying that that upper bound stated in Theorem 1.2 is tight.

In Section 4 we analyze the performance of simple memoryless algorithms called *probabilistic following*, suggested in [19]. At every step, the algorithm follows the advice at the current vertex with some fixed probability λ, and performs a random walk step otherwise. It turns out that such algorithms can perform well, but only in a very limited regime of noise. Specifically, we prove:

Theorem 1.5. *There exist positive constants c_1, c_2 and c_3 such that the following holds. If for every vertex u, $q_u < c_1/\Delta_u$ then there exists a probabilistic following algorithm that finds the treasure in less than $c_2 d$ expected steps. On the other hand, if $q > c_3/\Delta$ then for every probabilistic following strategy the move complexity on a complete Δ-ary tree is exponential in the depth of the tree.*

Since this algorithm is randomized, expectation is taken over both the randomness involved in sampling advice and the possible probabilistic choices made by the algorithm.

Interestingly, when $q_u < c_1/\Delta_u$ for all vertices, this algorithm works even in the oblivious semi-adversarial model. In fact, it turns out that in the semi-adversarial model, probabilistic following algorithms are the best possible up to constant factors, as the threshold for efficient search, with respect to any algorithm, is roughly $1/\Delta$.

1.2.2 Results in Probability Guarantee

We start the investigation of algorithms having a good probability guarantee with the following upper bound. The proof is presented in Section 5. The $O(1)$ term in the exponent is to be understood as an absolute constant, that does not depend on either d or ε.

Theorem 1.6. *Let $0 < \varepsilon < 1/2$ be a constant, and suppose that $q = \Delta^{-\varepsilon}$, and that Δ is sufficiently large ($\Delta \geq 2^{6/\varepsilon^2}$ suffices). Let $0 < \delta < 1$ be a constant. Then there exists an algorithm A'_{walk} in the walk model that discovers τ in $(\frac{d}{\delta})^{O(\frac{1}{\varepsilon})}$ moves with probability $1 - \delta$. Moreover, the statement holds even in the adaptive semi-adversarial variant.*

A remark about the parameters. *The restriction of $\varepsilon < \frac{1}{2}$ is inessential to Theorem 1.6, and is included because the algorithms of Theorem 1.2 already handle the case $\varepsilon \geq \frac{1}{2}$. The requirement that Δ is sufficiently large as a function of ε is natural, particularly for the semi-adversarial setting. For example, taking $\Delta \leq (3/2)^{1/\varepsilon}$ and keeping the assumption that $q = \Delta^{-\varepsilon}$ will lead to $q \geq 2/3$. In the semi-adversarial setting, such levels of noise could not be overcome efficiently. To see why, consider for instance a complete binary tree. The strategy of an adversary could be, at each faulty node, to point to a uniformly chosen neighbor, amongst the two that do not lead to τ. The result would then be that at every node, each direction of the advice is uniform, making it useless. On the other hand, if we require $q \leq \min(c, \Delta^{-\varepsilon})$ for some suitable constant $c > 0$ that depends only on ε, then the requirement that Δ is sufficiently large can be removed. One can take $c = 2^{-6/\varepsilon}$, and define $\Delta_0 = 2^{6/\varepsilon^2}$. For $\Delta \geq \Delta_0$ Theorem 1.6 applies because Δ is sufficiently large, whereas for $\Delta \leq \Delta_0$ Theorem 1.6 applies because we may pretend that the largest degree is Δ_0, and this does not affect the proofs.*

The upper bound shown in Theorem 1.6 is matched up to the constant in the exponent, by the following lower bound, presented in Section 6.

Theorem 1.7. *Let $0 < \varepsilon < 1/2$ be an arbitrary constant, and suppose that $q = \Delta^{-\varepsilon}$, and that D is sufficiently large, as a function of Δ and ε. Consider the complete Δ-ary tree of depth D, with the treasure placed in one of its leaves. Let A be an algorithm with success probability $1 - \delta$. Then, with constant probability, A needs at least $(\delta^{-1}D)^{\frac{1-\varepsilon}{\varepsilon}(1-o_D(*))}$ queries (and consequently also moves) before finding τ. ($o_D(\cdot)$ denotes a function of δ, ε, Δ and D, that tends to 0 when fixing the former parameters and letting D grow to infinity.) The statement also holds with respect to randomized algorithms.*

1.3 Related Work

In computer science, search algorithms have been the focus of numerous works, often on a tree topology, see e.g., [26, 2, 28, 27]. Within the literature on search, many works considered noisy queries [16, 24, 15]. However, it was typically assumed that noise can be *resampled* at every query. Dealing with permanent faults entails challenges that are fundamentally different from those that arise when allowing queries to be resampled. To illustrate this difference, consider the simple example of a star graph and a constant $q < 1$. Straightforward amplification can detect the target in $\mathcal{O}(1)$ queries in expectation. In contrast, in our model, it can be easily seen that $\Omega(n)$ is a lower bound for both the expected move and the query complexities, for any constant noise parameter.

A search problem on graphs in which the set of nodes with misleading advice is chosen by an adversary was studied in [20, 21, 22], as part of the more general framework of the *liar models* [1, 6, 9, 29]. Data structures with adversarial memory faults have been investigated in the so called faulty-memory RAM model, introduced in [18]. In particular, data structures (with adversarial memory faults) that support the same operations as search trees were studied in [17, 8]. Interestingly, the data structures developed in [8] can cope with up to $O(\log n)$ faults, happening at any time during the execution of the algorithm, while maintaining optimal space and time complexity. It is important to observe that all these models take worst case assumptions, leading to technical approaches and results which are very different from what one would expect in average-case analysis. Persistent probabilistic memory faults, as we study here, have been explicitly studied in [7], albeit in the context of sorting. Persistent probabilistic errors were also studied in contexts of learning and optimization, see [23].

The noisy advice model considered in this paper actually originated in the recent biologically centered work [19], aiming to abstract navigation relying on guiding instructions in the context of collaborative transport by ants. In that work, the authors modeled ant navigation as a probabilistic following algorithm, and noticed that an execution of such an algorithm can be viewed as an instance of Random Walks in Random Environment (RWRE) [30, 13]. Relying on results from this subfield of probability theory, the authors showed that when tuned properly, such algorithms enjoy linear move complexity on grids, provided that the bias towards the correct direction is sufficiently high.

An important theme of our work is the distinction between bounds on the expectation and bounds that hold with high probability. When randomization is an aspect of the algorithm rather than of the input instance, there is not much of a difference between expected running time and median running time, if one is allowed to restart the algorithm several times with fresh randomness. However, there might be a substantial difference if restarting with fresh randomness is not allowed. In our model, this can be seen, for instance, in the aforementioned star graph example in which each node is faulty with probability q. Here, finding the treasure requires $\Omega(qn)$ moves in expectation, but can be done in at most 2 moves, with probability $1 - q$. In general, for settings in which the random aspect comes up in the generation of the input instances (as in our case), generating a fresh random instance is not an option, and such a difference may or may not arise. In the context of designing polynomial time algorithms for distributions over instances of NP-hard problems, it is often the case that high probability algorithms are designed first, and algorithms with low expected runtime (over the same input distribution) are designed only later (see for example [25, 10]).

1.4 Notation

Denote $p = 1 - q$, and for a node u, $p_u = 1 - q_u$. For two nodes u, v, let $\langle u, v\rangle$ denote the simple path connecting them, excluding the end nodes, and let $[u, v\rangle = \langle u, v\rangle \cup \{u\}$ and $[u, v] = [u, v\rangle \cup \{v\}$. For a node u, let $T(u)$ be the subtree rooted at u. We denote by $\overrightarrow{\texttt{adv}}(u)$ (resp. $\overleftarrow{\texttt{adv}}(u)$) the set of nodes whose advice points towards (resp. away from) u. By convention $u \notin \overrightarrow{\texttt{adv}}(u) \cup \overleftarrow{\texttt{adv}}(u)$. Unless stated otherwise, log is the natural logarithm.

The nodes on the path from the root σ to the treasure τ are named as $[\sigma, \tau] := \{v_0 = \sigma, v_1, \ldots, v_{d-1}, v_d = \tau\}$. We say that node v is a *descendant* of node u if u lies on the path from σ to v, and v is a *child* of u if it is both a descendant of u and a neighbor of u.

2 Optimal Algorithm in Expectation

In this section we prove Theorem 1.2. At a high level, at any given time, the algorithm processes the advice seen so far, identifies a "promising" node to continue from on the border of the already discovered connected component, moves to that node, and explores one of its neighbors. The crux of the matter is identifying a notion of *promising* that leads to an efficient algorithm. We do so by introducing a carefully chosen *prior* for the treasure location.

2.1 Algorithm Design following a Greedy Bayesian Approach

In our setting the treasure is placed by an adversary. However, we can still study algorithms that assume that it is placed according to some known distribution, and see how they measure up in our worst case setting. A similar approach is used in [3], which studies the related (but simpler) problem of search on the line. The success of this scheme highly depends on the choice of the prior distribution we take.

Suppose first that the structure of the tree is known to the algorithm, and that the treasure is placed according to some known distribution θ supported on the leaves. Let $\texttt{adv}$ denote the advice on the nodes we have already visited. Aiming to find the treasure as fast as possible, a possible greedy algorithm explores the vertex that, given the advice seen so far, has the highest probability of having the treasure in its subtree.

We extend the definition of θ to internal nodes by defining $\theta(u)$ to be the sum of $\theta(w)$ over all leaves w of $T(u)$. Given some u that was not visited yet, and given the previously seen advice $\texttt{adv}$, the probability of the treasure being in u's subtree $T(u)$, is:

$$\begin{aligned}\mathbb{P}\left(\tau \in T(u) \,|\, \texttt{adv}\right) &= \frac{\mathbb{P}\left(\tau \in T(u)\right)}{\mathbb{P}\left(\texttt{adv}\right)} \mathbb{P}\left(\texttt{adv} \,|\, \tau \in T(u)\right) \\ &= \frac{\theta(u)}{\mathbb{P}\left(\texttt{adv}\right)} \prod_{w \in \overrightarrow{\texttt{adv}}(u)} \left(p_w + \frac{q_w}{\Delta_w}\right) \prod_{w \in \overleftarrow{\texttt{adv}}(u)} \frac{q_w}{\Delta_w}.\end{aligned}$$

The last factor is q_w/Δ_w because it is the probability that the advice at w points exactly the way it does in $\texttt{adv}$, and not only away from τ. Note that the advice seen so far does not involve vertices in $T(u)$, because we consider a walking algorithm, and u has not been visited yet. Therefore, if $\tau \in T(u)$ then correct advice in $\texttt{adv}$ points to u. We ignore the term $p_w + q_w/\Delta_w$ because this will not affect the results by much, and applying a log we can

approximate the relative strength of a node by:

$$\log(\theta(u)) + \sum_{w \in \overleftarrow{\texttt{adv}}(u)} \log\left(\frac{q_w}{\Delta_w}\right).$$

We replace q_w by its upper bound $1/\sqrt{\Delta_w}$ (consequently, the algorithm need not know the exact value of q_w). After scaling, we obtain our definition for the score of a vertex:

$$\texttt{score}(u) = \frac{2}{3}\log(\theta(u)) - \sum_{w \in \overleftarrow{\texttt{adv}}(u)} \log(\Delta_w).$$

When comparing two specific vertices u and v, $\texttt{score}(u) > \texttt{score}(v)$ iff:

$$\sum_{w \in \langle u,v\rangle \cap \overrightarrow{\texttt{adv}}(u)} \log(\Delta_w) - \sum_{w \in \langle u,v\rangle \cap \overrightarrow{\texttt{adv}}(v)} \log(\Delta_w) > \frac{2}{3}\log\left(\frac{\theta(v)}{\theta(u)}\right).$$

This is because any advice that is not on the path between u and v contributes the same to both sides, as well as advice on vertices on the path that point sideways, and not towards u or v. Since we use this score to compare two vertices that are neighbors of already explored vertices, and our algorithm is a walking algorithm, then we will always have all the advice on this path. In particular, the answer to whether $\texttt{score}(u) > \texttt{score}(v)$, does not depend on the specific choices of the algorithm, and does not change throughout the execution of the algorithm, even though the scores themselves do change. The comparison depends only on the advice given by the environment.

Let us try and justify the score criterion at an intuitive level. Consider the case of a complete Δ-ary tree, with θ being the uniform distribution on the leaves. Here $score(u) > score(v)$ if (cheating a little by thinking of $\log(\Delta)$ and $\log(\Delta - 1)$ as equal):

$$\left|\overrightarrow{\texttt{adv}}(u) \cap \langle u, v\rangle\right| - \left|\overrightarrow{\texttt{adv}}(v) \cap \langle u, v\rangle\right| > \frac{2}{3}\big(d(u) - d(v)\big).$$

If, for example, we consider two vertices $u, v \in T$ at the same depth, then $score(u) > score(v)$ if there is more advice pointing towards u than towards v. If the vertices have different depths, then the one closer to the root has some advantage, but it can still be beaten.

For general trees, perhaps the most natural θ is the uniform distribution on all nodes (or just on all leaves - this choice is actually similar). It is also a generalization of the example above. Unfortunately, however, while this works well on the complete Δ-ary tree, the uniform distribution fails on other (non-complete) Δ-ary trees (see a preliminary version of this work [5] for details).

2.2 Algorithm $\texttt{A}_{walk}$

In our context, there is no distribution over treasure location and we are free to choose θ as we like. Take θ to be the distribution defined by a simple random process. Starting at the root, at each step, walk down to a child uniformly at random, until reaching a leaf. For a leaf v, define $\theta(v)$ as the probability that this process eventually reaches v. Our extension of θ can be interpreted as $\theta(v)$ being the probability that this process passes through v. Formally,

$\theta(\sigma) = 1$, and $\theta(u) = (\Delta_\sigma \prod_{w \in \langle \sigma, u \rangle} (\Delta_w - 1))^{-1}$. It turns out that this choice, slightly changed, works remarkably well, and gives an optimal algorithm in noise conditions that practically match those of our lower bound. For a vertex $u \neq \sigma$, define:

$$\beta(u) = \prod_{w \in [\sigma, u\rangle} \Delta_w. \tag{2}$$

It is a sort of approximation of $1/\theta(u)$, which we prefer for technical convenience. Indeed, for all u, $1/\beta(u) \leq \theta(u)$. A useful property of this β (besides the fact that it gives rise to an optimal algorithm) is that to calculate $\beta(v)$ (just like θ), one only needs to know the degrees of the vertices from v up to the root.

In the walking algorithm, if v is a candidate for exploration, the nodes on the path from σ to v must have been visited already and so the algorithm does not need any a priori knowledge of the structure of the tree. The following claim will be soon useful:

Claim 2.1. *The following two inequalities hold for every $c < 1$:*

$$\sum_{v \in T} \frac{c^{d(v)}}{\beta(v)} \leq \frac{1}{1-c}, \quad \sum_{v \in T} \frac{d(v) c^{d(v)}}{\beta(v)} \leq \frac{c}{(1-c)^2}.$$

Proof. To prove the first inequality, follow the same random walk defining θ. Starting at the root, at each step choose uniformly at random one of the children of the current vertex. Now, while passing through a vertex v collect $c^{d(v)}$ points. No matter what choices are made, the number of points is at most $1 + c + c^2 + ... = 1/(1-c)$. On the other hand, $\sum_{v \in T} \theta(v) c^{d(v)}$ is the expected number of points gained. The result follows since $1/\beta(v) \leq \theta(v)$. The second inequality is derived similarly, using the fact that $c + 2c^2 + 3c^3 + \ldots = c/(1-c)^2$. □

For a vertex $u \in T$ and previously seen advice `adv` define:

$$\texttt{score}(u) = \frac{2}{3} \log\left(\frac{1}{\beta(u)}\right) - \sum_{w \in \overleftarrow{\texttt{adv}}(u)} \log(\Delta_w). \tag{3}$$

This is similar to the definition of $\texttt{score}(u)$ given in Section 2.1, except that $\theta(u)$ is now replaced by its approximation $\frac{1}{\beta(u)}$.

Algorithm $\texttt{A}_{walk}$ keeps track of all vertices that are children of the vertices it explored so far, and repeatedly walks to and then explores the one with highest score according to the current advice, breaking ties arbitrarily. As stated in the introduction, the algorithm does not require prior knowledge of either the tree's structure, or the values of Δ, q, D or n.

2.3 Analysis

Recall the definition of Condition ($\star$) from Equation (1). The next lemma provides a large deviation bound tailored to our setting.

Lemma 2.2. *Consider independent random variables $X_1, \ldots, X_\ell$, where X_i takes the values $(-\log \Delta_i, 0, \log \Delta_i)$ with respective probabilities $(\frac{q_i}{\Delta_i}, q_i(1 - \frac{2}{\Delta_i}), p_i + \frac{q_i}{\Delta_i})$, for parameters $p_i, q_i = 1 - p_i$ and $\Delta_i > 0$. Assume that Condition ($\star$) holds for some $\varepsilon > 0$. Then for every integer (positive or negative) m,*

$$\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \leq m\right) \leq e^{\frac{3m}{4}} (1-\varepsilon)^\ell \prod_{i=1}^{\ell} \frac{1}{\sqrt{\Delta_i}}.$$

Proof. For every $s \in \mathbb{R}$,

$$\begin{aligned}\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \le m\right) &= \mathbb{P}\left(e^{s\sum_{i=1}^{\ell} -X_i} \ge e^{-sm}\right) \le e^{sm}\mathbb{E}\left[e^{s\sum_i -X_i}\right] = e^{sm}\prod_i \mathbb{E}\left[e^{-sX_i}\right] \\ &= e^{sm}\prod_{i=1}^{\ell}\left(\frac{p_i + \frac{q_i}{\Delta_i}}{e^{\log(\Delta_i)s}} + q_i\left(1-\frac{2}{\Delta_i}\right) + \frac{q_i}{\Delta_i}e^{\log(\Delta_i)s}\right) \\ &\le e^{sm}\prod_{i=1}^{\ell}\left(\frac{1}{\Delta_i^s} + q_i + q_i\Delta_i^{s-1}\right).\end{aligned}$$

We take $s = \frac{3}{4}$, and get:

$$\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \le m\right) \le e^{\frac{3m}{4}}\prod_{i=1}^{\ell}\left(\Delta_i^{-\frac{3}{4}} + q_i + q_i\Delta_i^{-\frac{1}{4}}\right) \le e^{\frac{3m}{4}}\prod_{i=1}^{\ell}\frac{1-\varepsilon}{\sqrt{\Delta_i}},$$

where for the last step we used Condition ($\star$) which says $q_i < \frac{1-\varepsilon-\Delta_i^{-\frac{1}{4}}}{\sqrt{\Delta_i}+\Delta_i^{\frac{1}{4}}}$ implying that:

$$\begin{aligned} q_i\Delta_i^{\frac{1}{2}} + q_i\Delta_i^{\frac{1}{4}} + \Delta_i^{-\frac{1}{4}} < 1-\varepsilon, \text{ and hence} \\ \Delta_i^{-\frac{3}{4}} + q_i + q_i\Delta_i^{-\frac{1}{4}} < \frac{1-\varepsilon}{\sqrt{\Delta_i}}.\end{aligned}$$

□

The next theorem establishes Theorem 1.2.

Theorem 2.3. *Assume that Condition ($\star$) holds for some fixed $\varepsilon > 0$. Then $\mathtt{A}_{walk}$ requires only $\mathcal{O}(d\sqrt{\Delta})$ moves in expectation. The constant hidden in the $\mathcal{O}$ notation only depends polynomially on $1/\varepsilon$.*

Proof. Denote the vertices on the path from σ to τ by $\sigma = u_0, u_1, \ldots, u_d = \tau$ in order. Denote by E_k the expected number of moves required to reach u_k once u_{k-1} is reached. We will show that for all k, $E_k = \mathcal{O}(\sqrt{\Delta})$, and by linearity of expectation this concludes the proof.

Once u_{k-1} is visited, $\mathtt{A}_{walk}$ only goes to some of the nodes that have score at least as high as u_k. We can therefore bound E_k from above by assuming we go through all of them, and this expression does not depend on the previous choices of the algorithm and the nodes it saw before seeing u_k. The length of this tour is bounded by twice the sum of distances of these nodes from u_k. Hence,

$$E_k \le 2\sum_{i=1}^{k}\sum_{u\in C(u_i)} \mathbb{P}\left(\mathtt{score}(u) \ge \mathtt{score}(u_k)\right)\cdot d(u_k, u),$$

where $C(u_k) = T(u_{k-1}) \setminus T(u_k)$, and so $\bigcup_{i=1}^{k} C(u_i) = T \setminus T(u_k)$. Recall from Eq. (3) that

scores are defined so that

$$\begin{aligned}
&\texttt{score}(u_k) \le \texttt{score}(u) \\
&\iff \\
&\sum_{w \in \overleftarrow{\texttt{adv}}(u)} \log(\Delta_w) - \sum_{w \in \overleftarrow{\texttt{adv}}(u_k)} \log(\Delta_w) \le \frac{2}{3}\left(\log\left(\frac{1}{\beta(u)}\right) - \log\left(\frac{1}{\beta(u_k)}\right)\right) \\
&\iff \\
&\sum_{w \in \langle u, u_k \rangle} \begin{cases} \log(\Delta_w) & w \text{ points towards } u_k \\ -\log(\Delta_w) & w \text{ points towards } u \\ 0 & \text{otherwise} \end{cases} \le \frac{2}{3}\log\left(\frac{\beta(u_k)}{\beta(u)}\right)
\end{aligned}$$

Indeed, a vertex x on the path should point towards u_k: this happens with probability $p_x + q_x/\Delta_x$. Otherwise, it points towards u with probability q_x/Δ_x, and elsewhere with probability $q_x(1 - 2/\Delta_x)$. Denoting $c = 1 - \varepsilon$, setting $m = \frac{2}{3}\log(\beta(u_k)/\beta(u))$, and applying Lemma 2.2 we can upper bound the probability of this happening:

$$\begin{aligned}
\frac{E_k}{2} &\le \sum_{i=1}^{k} \sum_{u \in C(u_i)} e^{\frac{3}{4}\cdot\frac{2}{3}\log\left(\frac{\beta(u_k)}{\beta(u)}\right)} \cdot c^{d(u_k,u)-1} \sqrt{\prod_{v \in \langle u, u_k \rangle} \frac{1}{\Delta_v}} \cdot d(u_k, u) \\
&= \frac{1}{c}\sum_{i=1}^{k} \sum_{u \in C(u_i)} \frac{c^{d(u_k,u)}}{\sqrt{\frac{\beta(u)}{\beta(u_k)}}} \sqrt{\frac{\Delta_{u_i}}{\frac{\beta(u_k)}{\beta(u_i)} \cdot \frac{\beta(u)}{\beta(u_i)}}} \cdot d(u_k, u) \\
&\le \frac{\sqrt{\Delta}}{c}\sum_{i=1}^{k} c^{d(u_k,u_i)} \sum_{u \in C(u_i)} c^{d(u_i,u)} \frac{\beta(u_i)}{\beta(u)} \cdot \Big(d(u_k, u_i) + d(u_i, u)\Big).
\end{aligned}$$

By Claim 2.1, applied to the tree rooted at u_i, we get:

$$\sum_{u \in C(u_i)} \frac{c^{d(u_i,u)}\beta(u_i)}{\beta(u)} < \frac{1}{1-c}, \quad \text{and} \quad \sum_{u \in C(u_i)} \frac{c^{d(u_i,u)}\beta(u_i)}{\beta(u)} d(u_i, u) < \frac{c}{(1-c)^2}.$$

And so:

$$\begin{aligned}
\frac{E_k}{2} &\le \frac{\sqrt{\Delta}}{c(1-c)}\sum_{i=1}^{k} c^{d(u_k,u_i)} d(u_k, u_i) + \frac{\sqrt{\Delta}}{(1-c)^2}\sum_{i=1}^{k} c^{d(u_k,u_i)} \\
&\le \frac{(1+c)\sqrt{\Delta}}{(1-c)^3} \le \frac{2\sqrt{\Delta}}{\varepsilon^3} = \mathcal{O}\left(\sqrt{\Delta}\right),
\end{aligned}$$

where we again used the equality $c + 2c^2 + 3c^3 + \ldots = c/(1-c)^2$. □

3 Lower bounds in Expectation

Several of our lower bounds will invoke the following straightforward observation (whose proof we omit).

Observation 3.1. *Any randomized algorithm trying to find a treasure chosen uniformly at random between k identical objects will need an expected number of queries that is at least $(k+1)/2$.*

3.1 The Random Noise Model

3.1.1 Exponential complexity above the threshold

We prove here Theorem 1.3. Namely, that for every fixed $\varepsilon > 0$, and for every complete Δ-ary tree, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}}$, then every randomized search algorithm has query (and move) complexity which is exponential in the depth D of the treasure.

Proof. Our lower bound holds also in the query model, as we assume that the algorithm gets as input the full topology of the tree. Moreover, to simplify the proof, we give the algorithm access to additional information, and prove the lower bound even against this stronger algorithm. The algorithm is strengthened in two respects: for every internal (non-leaf) node, the algorithm is told whether the node is faulty, and moreover, if the internal node is non-faulty, the advice of the node is revealed to the algorithm. This information for all internal nodes is revealed to the algorithm for free, without the algorithm making any query. Given that the algorithm is told which nodes are faulty, we may assume that faulty nodes have no advice at all, because faulty advice is random and hence can be generated by the algorithm itself.

Given the pattern of faults, a leaf is called *uninformative* if the whole root to leaf path is faulty. Denote the number of leaves by $N = \Delta(\Delta-1)^{D-1}$, and the expected number of uninformative leaves by $\mu = Nq^D$. Let p_i denote the probability that there are exactly i uninformative leaves.

The adversary places the treasure at a random leaf. Conditioned on there being i uninformative leaves, the probability of the treasure being at an uninformative leaf is exactly $\frac{i}{N}$. If the treasure is located at an uninformative leaf, the advice of all nonfaulty internal nodes points to the root (recall that there is no advice on faulty nodes), and the algorithm can infer that the treasure is at an uninformative leaf. As all uninformative leaves are equally probable, the expected number of queries that the algorithm needs to make is exactly $\frac{i+1}{2}$. Hence the expected number of queries (in this stronger model) is:

$$\sum_i \frac{i+1}{2}\frac{i}{N}p_i > \frac{1}{2N}\sum_i i^2 p_i \geq \frac{1}{2N}\mu^2 = \frac{Nq^{2D}}{2}.$$

In the last inequality we used the fact that $E[X^2] \geq (E[X])^2$ for every random variable X. (In our use the random variable is i, the number of uninformative leaves.)

Hence if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}}$, the expected number of queries is at least $\frac{1}{2}(1+\varepsilon)^{2D}$. □

3.1.2 A Query Lower Bound of $\Omega(\sqrt{\Delta}D)$ when $q \sim 1/\sqrt{\Delta}$

We now prove Theorem 1.4. Specifically, we prove that for $\Delta \geq 3$, on the complete Δ-ary tree of depth D, every algorithm needs $\Omega(q\Delta D)$ queries in expectation. Note that, in particular, when q is roughly $1/\sqrt{\Delta}$, the query complexity becomes $\Omega(\sqrt{\Delta}D)$.

To prove the lower bound of $\Omega(q\Delta D)$, consider the complete Δ-ary tree of depth D. We prove by induction on D, that if the treasure is placed uniformly at random in one of the leaves, then the expected query complexity of every algorithm is at least $q(\Delta/2-1)D$. If $D = 0$, then there is nothing to show. Assume this is true for D, and we shall prove it for $D+1$. Let $T_1, \ldots, T_{\Delta-1}$ be the subtrees hanging down from the root (in the induction, the "root" is actually an internal node, and so has $\Delta - 1$ children), each having depth D. Let i

be the index such that $\tau \in T_i$, and denote by Q the number of queries before the algorithm makes its first query in T_i. We will assume that the algorithm gets the advice in the root for free. Denote by Y the event that the root is faulty . In this case, Observation 3.1 applies, and we need at least $\Delta/2 - 1$ queries to hit the correct tree. We subtracted one query from the count because we want to count the number of queries strictly before querying inside T_i. We therefore get $\mathbb{E}[Q] \geq \mathbb{P}(Y) \cdot \mathbb{E}[Q \mid Y] \geq q(\Delta/2 - 1)$. By linearity of expectation, using the induction hypothesis, we get the result for a uniformly placed treasure over the leaves, and so it holds also in the adversarial case. □

3.2 The Semi-Adversarial Variant

Recall that in contrast to the purely probabilistic model, in the oblivious semi-adversarial model, a faulty node u no longer points to a neighbor chosen uniformly at random, and instead, the neighbor w which such a node points to is chosen by an adversary. Importantly, for each node u, the adversary must specify its potentially faulty advice w, before it is known which nodes will be faulty. In other words, first, the adversary specifies the faulty advice w for each node u, and then the environment samples which node is faulty.

In the semi-adversarial noise model, if $q > 1/(\Delta - 1)$ then any algorithm must have expected query and move complexity that are exponential in the depth D.

Theorem 3.2. *Consider an algorithm in the oblivious semi-adversarial model. On the complete Δ-ary tree of depth D, the expected number of queries to find the treasure is $\Omega\left((q(\Delta-1))^D\right)$. The lower bound holds even if the algorithm has access to the advice of all internal nodes in the tree.*

Proof. Consider the complete Δ-ary tree and restrict attention to the cases where the treasure is located at a leaf. The adversary behaves as follows. For every advice it gets a chance to manipulate, it always make it point towards the root. With probability q^D the adversary gets to choose all the advice on the path between the root and the treasure. Any other advice points towards the root as well (either because it was correct to begin with or because it was set by the adversary). Hence with probability q^D the tree that the algorithm sees is the same regardless of the position of the treasure. When this happens, the expected time to find the treasure is $\Omega((\Delta-1)^D)$ (linear in the number of leaves), by Observation 3.1. □

4 Probabilistic Following Algorithms

In this section we present our results on the probabilistic following algorithms described in the introduction. As mentioned, such algorithms can perform well also in the more difficult oblivious semi-adversarial setting.

Recall that a *Probabilistic Following* (PF) algorithm is specified by a *listening* parameter $\lambda \in (0, 1)$. At each step, the algorithm "listens" to the advice with probability λ and takes a uniform random step otherwise. The first item in the next theorem states that if the noise parameter is smaller than c/Δ for some small enough constant $0 < c < 1$, then there exists a listening parameter λ for which Algorithm PF achieves $\mathcal{O}(D)$ move complexity. Moreover, this result holds also in the oblivious semi-adversarial model. Hence, together with Theorem 3.2, it implies that in order to achieve efficient search, the noise parameter threshold for the semi-adversarial model is $\Theta(1/\Delta)$.

Theorem 4.1. *1. Assuming $q_u < 1/(10\Delta_u)$ for every u, then PF with parameter $\lambda = 0.7$ finds the treasure in less than $100D$ expected steps, even in the oblivious semi-adversarial setting.*

2. Consider the complete Δ-ary tree and assume that $q > 10/\Delta$. Then for every choice of λ the hitting time of the treasure by PF is exponential in the depth of the tree, even assuming the faulty advice is drawn at random.

Proof. Our plan is to show that the expected time to make one step in the correct direction is $\mathcal{O}(1)$, from any starting node. Conditioning on the advice setting, we make use of the Markov property to relate these elementary steps to the total travel time. The main delicate point in the proof stems from dealing with two different sources of randomness. Namely the randomness of the advice and that of the walk itself.

In this section, it is convenient to picture the tree as rooted at the target node τ. For every node u in the tree, we denote by u' the parent of u with respect to the treasure. With this convention, correct advice at a node u points to u', while incorrect advice points to one of its children. The fact the walk moves on a tree means that for a given advice setting, the expected (over the walk) time it takes to reach u' from u can be written conveniently as a product of two independent random variables: one random variable that depends only on the advice at u, and the other depends only on the advice on the set of u's descendants.

We denote by $t(u)$ the time it takes to reach node u. We also introduce a convention regarding the notation used to denote expectations. In our setting there are two sources of randomness, the first being the randomness used in drawing the advice, and the second being the randomness used in the walk itself. We write $\mathbb{E}$ for expectation taken over the advice, while we use E_u to denote expectation over the walk, conditioning on u being the starting node. Observe that $E_u(t(v))$ is a random variable with respect to the advice, whereas $\mathbb{E}E_u(t(v))$ is just a number.

The following is the central lemma of this section.

Lemma 4.2. *Assume that for every vertex u, $q_u < 1/(10\Delta_u)$, and $\lambda = 0.7$. Then for all nodes u, $\mathbb{E}E_u t(u') \leq 100$. The result holds also in the oblivious semi-adversarial model.*

Let us now see how we can conclude the proof of the first item in Theorem 4.1, given the lemma. Consider a designated source σ. Let us denote by $\sigma = u_d, u_{d-1}, \ldots, u_0 = \tau$ the nodes on the path from σ to τ. Let δ_i be the random variable indicating the time it takes to reach u_{i-1} after u_i has been visited for the first time. The time to reach τ from σ is precisely $\sum_{i=1}^{d(\sigma,\tau)} \delta_i$. Hence, the expected time to reach τ from σ is $\sum_{i=1}^{d(\sigma,\tau)} \mathbb{E}[E_\sigma \delta_i]$. Conditioning on the advice setting, the process is a Markov chain and we may write

$$E_\sigma \delta_i = E_{u_i} t(u_{i-1}).$$

Taking expectations over the advice ($\mathbb{E}$), under the assumptions of Lemma 4.2, it follows that $\mathbb{E}(E_\sigma \delta_i) \leq 100$, for every $i \in [d(\sigma,\tau)]$. And this immediately implies a bound of $100 \cdot d(\sigma,\tau)$.

Proof of Lemma 4.2. We start with partitioning the nodes of the tree according to their distance from the root τ. More precisely, for $i = 1, 2, \ldots, D$, where D is the depth of the tree, let us define

$$\mathcal{L}_i := \{u \in T : d(u,\tau) = i\} .$$

The nodes in $\mathcal{L}_i$ are referred to as *level-i* nodes. We treat the statement of the lemma for nodes $u \in \mathcal{L}_i$ as an induction hypothesis, with i being the induction parameter. The induction goes backwards, meaning we assume the assumption holds at level $i+1$ and show it holds at level i. The case of the maximal level (base case for the induction) is easy since, at a leaf the walk can only go up and so if u is a leaf $\mathbb{E}E_u(t(u')) = 1 < 100$.

Assume now that $u \in \mathcal{L}_i$. We first condition on the advice setting. A priori, $E_u t(u')$ depends on the advice over the full tree, but in fact it is easy to see that only advice at layers $\geq i$ matter. Recall from Markov Chain theory that an *excursion* to/from a point is simply the part of the walk between two visits to the given point. We denote L_u the average (over the walk only) length of an excursion from u to itself that does not go straight to u', and we write N_u to denote the expected (over the walk only) number of excursions before going to u'. We also refer to this number as a number of *attempts*. The variable N_u can be 0 if the walk goes directly to u' without any excursion. We decompose $t(u')$ in the following standard way, using the Markov property

$$E_u t(u') = 1 + L_u \cdot N_u. \tag{4}$$

Indeed, the expectation $E_u t(u')$ can be seen as the expectation (over the walk) of $1 + \sum_{i=1}^{T} Y_i$ where the Y_i's are the lengths of each excursion from u and T is the (random) number of such excursions before hitting u'. The term 1+ accounts for the step from u to u'. The event $\{T \geq t\}$ is independent of $Y_1, \ldots, Y_t$ and so using Wald's identity we have that $E_u t(u') = 1 + E_u T \cdot E_u Y_1$. The term $E_u T$ is equal to N_u (by definition) while $E_u Y_1$ is equal to L_u (by definition).

We now want to average equality (4), which is only an average over the walk, by taking the expectation over all advice in layers $\geq i$. To this aim, we write L_u as follows

$$L_u = 1 + \sum_{v \neq u', v \sim u} p_{u,v} E_v t(u),$$

where we write $u \sim v$ when u and v are neighbors in the tree and $p_{u,v}$ is the probability to go straight from u to v given the advice setting. By assumption on the model, $E_v t(u)$ depends on the advice at layers $\geq i+1$ only, if we start at a node $v \in \mathcal{L}_{i+1}$, while both $p_{u,v}$ and N_u depend only on the advice at layer $= i$ of the tree. This is true also in the oblivious semi-adversarial model. Hence when we average, we can first average over the advice in layers $> i$ to obtain, denoting $\mathbb{E}^{>i}$, the expectation over the layers $> i$,

$$\begin{aligned}
\mathbb{E}^{>i} E_u t(u') &= 1 + \left(1 + \sum_{v \neq u', v \sim u} p_{u,v} \mathbb{E}^{>i} E_v t(u)\right) N_u, \\
&= 1 + \left(1 + \sum_{v \neq u', v \sim u} p_{u,v} \mathbb{E} E_v t(u)\right) N_u. \qquad (5)
\end{aligned}$$

and using the fact that, $\sum_{v \neq u'} p_{u,v} \leq 1$, together with the induction assumption at rank $i+1$, we obtain

$$\mathbb{E}^{>i} E_u t(u') \leq 1 + (1 + 100)\, N_u.$$

Averaging over the layer i of advice we obtain

$$\mathbb{E} E_u t(u') \leq 1 + 101 \mathbb{E} N_u.$$

It only remains to analyse the term $\mathbb{E}N_u$. If the advice at u is correct, which happens with probability $p_u = 1 - q_u$, then the number of attempts follows a (shifted by 1) geometric law with parameter $\lambda + \frac{(1-\lambda)}{\Delta_u}$. In words, when the advice points to u' which happens with probability at most 1, the walker can go to the correct node either because she listens to the advice, which happens with probability λ, or because she did not listen, but still took the right edge, which happens with probability $\frac{(1-\lambda)}{\Delta_u}$. Similarly, when the advice points to a node $\neq u'$, which happens with probability at most q_u, then N_u follows a geometric law (shifted by 1) with parameter $\frac{(1-\lambda)}{\Delta_u}$. The conclusion is that

$$\begin{aligned}\mathbb{E}N_u &\leq \left(\frac{1}{\lambda + \frac{(1-\lambda)}{\Delta_u}} - 1\right) + q_u\left(\frac{\Delta_u}{1-\lambda} - 1\right) \\ &\leq \frac{1}{\lambda} - 1 + \frac{q_u\Delta_u}{1-\lambda} \qquad (6)\end{aligned}$$

And so it follows that

$$\mathbb{E}E_u t(u') \leq 1 + 101 \cdot \left(\frac{1}{\lambda} - 1 + \frac{q_u\Delta_u}{1-\lambda}\right)$$

Hence if $q_u\Delta_u < 0.1$ and we choose $\lambda = 0.7$ (this choice is a tradeoff between two considerations: λ bounded away above $\frac{1}{2}$ is required so that in expectation we make constant progress towards the treasure when the advice is correct, and λ bounded away below 1 is required so as to have probability larger than q for advancing when the advice is incorrect), we see that $\mathbb{E}N_u < 0.9$. This is because

$$\frac{1}{\lambda} - 1 + \frac{0.1}{1-\lambda} \leq \frac{10}{7} - 1 + \frac{0.1}{1-0.7} < 0.9$$

Hence it follows that $\mathbb{E}E_u t(u') \leq 1 + 0.9 \cdot 101 < 100$. By our (backwards) induction, we have just shown that, if $q < \frac{1}{10\Delta}$ and we set $\lambda = 0.7$ then for all nodes u in the tree

$$\mathbb{E}E_u t(u') < 100.$$

This concludes the proof of Lemma 4.2 and hence also of the first part of Theorem 4.1. □

Let us explain how the lower bound in the second part of Theorem 4.1 is derived in the case that $q\Delta > 10$. We assume the complete Δ-ary tree under our the uniform noise model. With probability q there is fault at u and with probability $1 - \frac{1}{\Delta}$ the advice does not point to u'. Recall that N_u denotes the expected (over the walk only) number of excursions starting from u before going to u's parent. Then, N_u follows a geometric law with parameter $\frac{1-\lambda}{\Delta}$. Hence

$$\mathbb{E}(N_u) \geq q\Delta\left(1 - \frac{1}{\Delta}\right)\frac{1}{1-\lambda} - 1 \geq \frac{10(1-\frac{1}{\Delta})}{1-\lambda} - 1 \geq 10\left(1 - \frac{1}{\Delta}\right) - 1 \geq 3,$$

for every choice of λ, since $\Delta \geq 2$. We proceed by induction similar to the proof of the first part of Theorem 4.1, and use Equality (5) together with the lower bound on $\mathbb{E}(N_u)$ to obtain that for every node on layer i, u with parent u', $\mathbb{E}E_u t(u') \geq 1 + 3\min_{v\in\mathcal{L}_{i+1}} \mathbb{E}E_v t(v')$, so in particular

$$\min_{u\in\mathcal{L}_i} \mathbb{E}E_u t(u') \geq 1 + 3 \min_{v\in\mathcal{L}_{i+1}} \mathbb{E}E_v t(v').$$

The expected hitting time of the target τ, even starting at one of its children is therefore of order $\Omega(3^D)$. □

5 Upper Bounds in High Probability

Our goal in this section is to prove Theorem 1.6, stating that for any constants $0 < \delta < 1$ and $0 < \varepsilon < 1/2$, if $q = \Delta^{-\varepsilon}$, and Δ is sufficiently large (specifically, $\Delta \geq 2^{6/\varepsilon^2}$, see remark in Section 1.2.2), then there exists a walking algorithm A'_{walk} (parameterized by δ, Δ and ε) that discovers τ in $(d/\delta)^{O(\frac{1}{\varepsilon})}$ moves with probability $1 - \delta$. Moreover, the statement holds even in the adaptive semi-adversarial variant.

5.1 The Meta Algorithm

Underlying the upper bound presented in Theorem 1.6 is a simple, yet general, scheme. It is based on a binary notion of *fitness*, which we define later. This notion depends on the parameters of the model. It is carefully crafted such that the following *fitness properties* hold:

- F1. Whether or not a node u is fit only depends on the advice on the path $[\sigma, u]$, excluding u.
- F2. For every node u on the path $[\sigma, \tau]$, $P(u \text{ is fit}) \geq 1 - \frac{\delta}{2D}$.
- F3. With probability at least $\geq 1 - \frac{\delta}{2}$, the connected component of fit nodes that contains the root is of size bounded by $f(D, \delta)$, for some function f.

Once fitness is appropriately defined so that properties F1 - F3 hold, a *depth first search* algorithm can be applied in the walk model. It consists of exploring in a depth-first fashion the connected component of fit nodes containing the root. We refer to this algorithm as A'_{walk}. Property F1 ensures that A'_{walk} is well-defined. The time to explore a component is at most twice its size, because each edge is traversed at most twice.

Claim 5.1. *Property F2 implies that A'_{walk} eventually finds τ with probability $\geq 1 - \frac{\delta}{2}$.*

Proof. Using Property F2, the probability that all nodes on the root to treasure path $[\sigma, \tau]$ are fit is at least as large as $1 - \frac{\delta D}{2D} = 1 - \frac{\delta}{2}$. Under this event, τ belongs to the same component of fit nodes as the root, and hence A'_{walk} eventually finds it. □

By Property F3, the A'_{walk} algorithm needs a number of steps which is upper bounded by $2f(D, \delta)$ with probability $1 - \frac{\delta}{2}$. Using a union bound we derive the following claim.

Claim 5.2. *If the fitness is defined so that properties F1-F3 are satisfied, then A'_{walk} finds τ in at most $2f(D, \delta)$ steps with probability $\geq 1 - \delta$.*

5.2 Upper Bound with High Probability

This subsection is devoted to the proof of Theorem 1.6. We assume the following, w.l.o.g.

- The noise model is the adaptive semi-adversarial variant. Hence the results apply also to the oblivious semi-adversarial and to the random variants.
- The algorithm knows the depth d of the treasure. This assumption can be removed by an iterative process that guesses the depth to be 1,2,.... By running each iteration for a limited time as specified in the theorem, the asymptotic runtime is not violated.

Given that the algorithm knows the depth d of the treasure, we further assume w.l.o.g. that it never searches a node at depth greater than d. Equivalently, we may (and do) assume that the depth of the tree is $d = D$, i.e., that the treasure is located at a leaf.

- The tree is balanced: all leaves are at depth D, and all non-leaf nodes have degree exactly Δ. To remove this assumption, whenever the algorithm visits a node v at depth $i < D$ of degree $d_v < \Delta$, it can connect to it $\Delta - d_v$ "auxiliary trees", where each auxiliary tree has depth $D - i - 1$ and is balanced. The advice in all nodes of these auxiliary trees points towards v, which is a valid choice in the adaptive semi-adversarial model.

Our algorithm A'_{walk} follows the general scheme presented in Section 5.1. It is based on a notion of fitness presented below. With this notion in hand, A'_{walk} simply visits, in a depth-first fashion, the component of fit nodes to which the root σ belongs.

Definition 5.3. *[Advice-fitness] Let $h_1 = \frac{2}{\varepsilon}\log_\Delta(4\delta^{-1}D)$ and $h_2 = \frac{6}{\varepsilon^2}\log_\Delta(4\delta^{-1}D)$. Let u be a node and u_{-h_2} be the ancestor of u at distance h_2 from u, or σ if u is at distance $< h_2$ from σ. The node u is said to be* fit *if the number of locations on the path $[u_{-h_2}, u]$ that do not point towards u is less than h_1. It is said to be* unfit *otherwise. Moreover, a fit node is said to be* reachable *if it is in the connected component of fit nodes that contains the root (as in Property F3). Equivalently, a node is reachable if either it is the root, or it is fit and its parent is reachable.*

Note that by definition, all nodes at depth $< h_1$ are fit and reachable. The notion of fitness clearly satisfies the first fitness property F1. We want to show that it also satisfies the properties F2 and F3 with $f(D, \delta) = (\delta^{-1}D)^{O(1/\varepsilon)}$. Let us first give two useful conditions satisfied by our choice of h_1 and h_2.

Claim 5.4. *The following inequalities hold:*

1. $2^{h_2}\Delta^{-\varepsilon h_1} \le \frac{\delta}{4D}$,
2. $2^{h_2}\Delta^{(1+\varepsilon)h_1 - \varepsilon h_2} \le \frac{\delta}{4D}$.

Proof. Equation (1). Replacing $h_{1/2}$ by their values, we bound the left hand side in Equation (1) as follows

$$2^{\frac{6}{\varepsilon^2}\log_\Delta(4\delta^{-1}D)}\Delta^{-\varepsilon\frac{2}{\varepsilon}\log_\Delta(4\delta^{-1}D)} = 2^{(6\varepsilon^{-2} - 2\log\Delta)\log_\Delta(4\delta^{-1}D)}.$$

We assume that $\Delta \ge 2^{6\varepsilon^{-2}}$ so that $6\varepsilon^{-2} \le \log\Delta$ and $6\varepsilon^{-2} - 2\log\Delta \le -\log\Delta$. Hence the left hand side in Equation (1) is not greater than $2^{-\log\Delta\log_\Delta(4\delta^{-1}D)} = \frac{\delta}{4D}$.

Equation (2). Using the fact that $\varepsilon \le 1$ and that $h_2 = \frac{3}{\varepsilon}h_1$, we obtain that

$$(1+\varepsilon)h_1 - \varepsilon h_2 \le 2h_1 - 3h_1 = -h_1 \le -\varepsilon h_1.$$

The result follows from Equation (1). □

Proposition 5.5. *The notion of advice-fitness introduced above satisfies Property F2, namely, that for every node u on the path $[\sigma, \tau]$, we have $P(u \text{ is fit}) \ge 1 - \frac{\delta}{2D}$. Hence, using Claim 5.1, with probability at least $1 - \frac{\delta}{2}$, algorithm A'_{walk} succeeds in finding τ.*

Proof. Let $u \in [\sigma, \tau]$. We want to upper bound the probability that u is unfit. The probability to be fit decreases with the distance to σ until reaching depth h_2, by definition. Hence, it suffices to check the case where u is at distance at least h_2 from σ.

Let S be a set of h_1 nodes on $[u_{-h_2}, u]$. The set S is completely faulty with probability $q^{h_1} \leq \Delta^{-\varepsilon h_1}$. The number of such sets S is at most 2^{h_2}. Hence $P(u \text{ is unfit}) \leq 2^{h_2}\Delta^{-\varepsilon h_1}$. We conclude using Equation (1) from Claim 5.4, that $P(u \text{ is unfit}) \leq \frac{\delta}{4D} \leq \frac{\delta}{2D}$. □

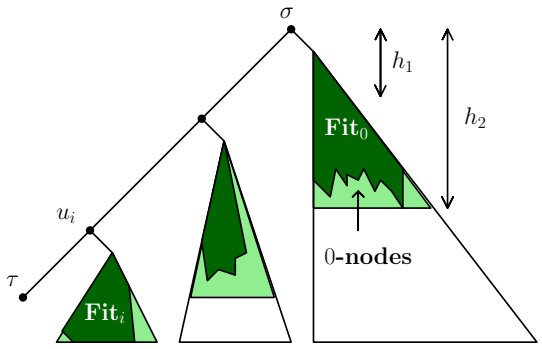

Figure 2: The partition of fit vertices introduced in the proof of Proposition 5.6. The colored nodes in the subtree on the right are the close 0-nodes, where those colored with dark green are the reachable fit 0-nodes. There are no fit 0-nodes at depth greater than h_2 in this example.

Proposition 5.6. *The notion of advice-fitness obeys Property F3 with* $f(D, \delta) = (\delta^{-1}D)^{O(\varepsilon^{-1})}$. *Hence the move complexity of* A'_{walk} *is less than* $(\delta^{-1}D)^{O(\varepsilon^{-1})}$, *with probability* $\geq 1 - \frac{\delta}{2}$.

Proof. Let Fit be the connected set of reachable nodes (as defined in Definition 5.3). Our goal is to show that with high probability, namely, with probability at least $1 - \frac{\delta}{2}$, we have $|\text{Fit}| = (\delta^{-1}D)^{O(\varepsilon^{-1})}$.

For $i \geq 0$, the term *i-node* will refer to any node whose common ancestor with τ is at depth i. An i-node is further said to be *close* if its depth lies in the range $[i, i + h_2]$. Let Fit_i be the set of close i-nodes in Fit (see Figure 2).

Our first goal is to show that with high probability, Fit does not contain any 0-node at depth h_2 (Claim 5.8). Under this high probability event, A'_{walk} visits only fit 0-nodes that are close (i.e., at depth at most h_2), because A'_{walk} visits only reachable nodes, and fit 0-nodes that are not close are disconnected from the root at depth h_2. Hence all the 0-nodes visited by A'_{walk} are in Fit_0. By symmetry, a similar statement holds for each layer i, and the corresponding subset Fit_i. Thus, under a high probability event, the nodes visited by A'_{walk} during its execution form a subset of $\bigcup_{i=0}^{D} \text{Fit}_i$ (namely, $f(D, \delta) \leq |\bigcup_{i=0}^{D} \text{Fit}_i|$).

Denote the expected number of fit 0-nodes at depth h by

$$\alpha_h := \sum_{u \text{ is a 0-node at depth } h} P(u \text{ is fit}).$$

We have

$$\mathbb{E}\left(|\text{Fit}_0|\right) = \sum_{u \text{ is a close 0-node}} P(u \text{ is fit}) = \sum_{h \leq h_2} \alpha_h. \tag{7}$$

Claim 5.7. *1.* $\sum_{h \le h_1} \alpha_h \le 2\Delta^{h_1}$.

2. For every $h_1 < h \le h_2$*, we have* $\alpha_h \le \Delta^{h_1(1+\varepsilon)} 2^h \Delta^{-\varepsilon h}$.

Proof. Every node at depth at most h_1 is fit. There are at most $2\Delta^{h_1}$ such nodes. Estimation (1) follows.

We now show the second estimate. Let U_h be a node chosen uniformly at random among all 0-nodes at depth h. Then $P(U_h \text{ is fit}) = \sum_{u \text{ is a close 0-node}} P(U_h = u) P(u \text{ is fit})$, and hence we may write:

$$\alpha_h = (\Delta - 1)^h P(U_h \text{ is fit}).$$

Choosing U_h randomly rather than arbitrarily is of crucial importance in the adaptive semi-adversarial variant, because the adversary might choose to direct all the faulty advice towards a specific node u. This could result in some terms in the sum (in the definition of α_h) being much bigger than the average. So it is important to avoid bounding the average by the max. We draw a uniform path $\sigma = U_0, U_1, \ldots, U_h$ of length h from the root, in the component of 0-nodes. Consider a node U_i on this path. With probability q, it is faulty. In this case, regardless of how the adversary could set its advice, the advice at U_i points to U_{i+1} with probability of at most $\frac{1}{\Delta - 1}$ over the choice of U_{i+1}.

It follows that the number of ancestors of U_h whose advice points to U_h may be viewed as the sum of h Bernoulli variables B_i with parameter $\frac{q}{\Delta-1}$. Moreover, the previous argument means that $P(B_i = 1 \mid B_j, j < i) = \frac{q}{\Delta-1} = P(B_i = 1)$. The B_i variables are thus uncorrelated and hence independent since they are Bernoullis. The node U_h is fit if at least $h - h_1$ ancestors point to it. Thus, by a union bound over the $\binom{h}{h-h_1} \le 2^h$ possible locations of faults, $P(U_h \text{ is fit}) \le 2^h \left(\frac{q}{\Delta-1}\right)^{h-h_1}$. Hence

$$\alpha_h \le 2^h (\Delta - 1)^h \left(\frac{q}{\Delta - 1}\right)^{h-h_1} \le 2^h q^{h-h_1} \Delta^{h_1} = \Delta^{h_1(1+\varepsilon)} 2^h \Delta^{-\varepsilon h}.$$

In the last step, we used $q = \Delta^{-\varepsilon}$. This concludes the proof of Claim 5.7. □

For $h = h_2$, combining Claim 5.7 and Equation (2) stated in Claim 5.4 implies:

$$\alpha_{h_2} \le \frac{\delta}{4D} \tag{8}$$

For $i \in [D]$, denote by Z_i (Z for *zero*) the event that there are no fit i-nodes at depth $i + h_2$. Applying Markov inequality on Equation (8) implies that $P(Z_0) \ge 1 - \delta(4D)^{-1}$. Since the same argument can be applied to any $i \le D$, we get

Claim 5.8. *For every* $i \le D$*, we have* $P(Z_i) \ge 1 - \frac{\delta}{4D}$ *and hence* $P(\bigcap Z_i) \ge 1 - \frac{\delta}{4}$.

If follows from the assumption on $\Delta \ge 2^{6\varepsilon^{-2}}$ that $\Delta^\varepsilon \ge 2^6 \ge 8$, so that $2\Delta^{-\varepsilon} < \frac{1}{4}$. Using Eq. (7), Claim 5.7 and the definition of h_1 we get:

$$\begin{aligned}
\mathbb{E}\left(|\mathrm{Fit}_0|\right) &\le \sum_{h \le h_2} \alpha_h \le 2\Delta^{h_1} + \sum_{h=h_1}^{h_2} \Delta^{h_1(1+\varepsilon)} \left(2\Delta^{-\varepsilon}\right)^h \\
&\le 2\Delta^{h_1} + \Delta^{h_1(1+\varepsilon)} \sum_{h \ge h_1} 4^{-h} \le 2\Delta^{h_1} + 2\Delta^{h_1(1+\varepsilon)} \le 4\Delta^{h_1(1+\varepsilon)} \\
&\le 4 \cdot (4\delta^{-1} D)^{2\varepsilon^{-1}+2} \le (4\delta^{-1} D)^{2\varepsilon^{-1}+3}.
\end{aligned}$$

The computation is the same for every $i \leq D$, yielding that $\mathbb{E}(|\mathrm{Fit}_i|) \leq (4\delta^{-1}D)^{2\varepsilon^{-1}+3}$, and by linearity of expectation, we obtain

$$\mathbb{E}(|\bigcup_{i=0}^{D} \mathrm{Fit}_i|) \leq D \cdot (4\delta^{-1}D)^{2\varepsilon^{-1}+3}.$$

Using the Markov inequality, with probability at least $1 - \frac{\delta}{4D}$, this variable is upper bounded by $4\delta^{-1}D^2 \cdot (4\delta^{-1}D)^{2\varepsilon^{-1}+3} \leq (4\delta^{-1}D)^{2\varepsilon^{-1}+5}$. As explained in the beginning of the proof, under the event $\bigcap_{i \leq D} Z_i$ (which occurs with probability at least $1 - \frac{\delta}{4}$ thanks to Claim 5.8), we have $\mathrm{Fit} \subset \bigcup_{i \leq D} \mathrm{Fit}_i$. Using a union bound, we conclude that with probability at least $1 - \frac{\delta}{2}$, we have $|\mathrm{Fit}| \leq (4\delta^{-1}D)^{2\varepsilon^{-1}+5} = (\delta^{-1}D)^{O(\varepsilon^{-1})}$, as desired. □

Claim 5.2 in combination with Propositions 5.5 and 5.6 proves Theorem 1.6.

6 Lower Bound for High Probability Algorithms

The goal of this section is to prove Theorem 1.7. Informally, the theorem shows that the upper bound in Theorem 1.6 is the right bound, up to a constant factor in the exponent.

The proof is done for the query complexity, in a complete Δ-ary tree of depth $D = \log_\Delta n$. This also implies at least the same lower bound for the move complexity. Throughout the proof, T denotes a complete Δ-ary tree of depth D. In our lower bound, the adversary places τ at a leaf of T chosen at random according to the uniform distribution. We denote by F the set of faulty nodes (without directional advice). Since this set as well as τ are chosen uniformly at random, we may assume without loss of generality that the algorithm is deterministic. The presentation of our proof is simplified if we assume that the algorithm is told which nodes of T are faulty (namely, are in F). We can make this assumption because it only strengthens our lower bound.

We reserve the letter H to denote a subtree of T, containing the descendants of its root (with respect to the original root σ). A subset S is said to be *completely faulty* if all nodes in S are faulty. Overloading this expression, we say that a leaf v of some subtree H is *completely faulty* if the path from the root of H towards v is completely faulty. The relevant reference subtree H will be specified if it is not clear from the context. The number of completely faulty leaves of a subtree H is denoted $B(H)$ or simply B if H is clear from the context. When considering a subset $S \subseteq T$, we write S^* to denote the pair $(S, S \cap F)$. In words, this corresponds to a subset with the information of which nodes are faulty.

6.1 Proof of Theorem 1.7

At a high level, the argument is as follows. On the path from the root to τ, with probability at least δ, there exists a segment $[v_i, v_{i+h-1}]$ of length $h \simeq \frac{1}{\varepsilon} \log_\Delta(\frac{D}{\delta})$, where all nodes are faulty (Lemma 6.1). On the other hand, the tree rooted at v_i of depth h, typically hosts many such completely faulty leaves (Lemma 6.2). In some sense these leaves are *indistinguishable*. This means that any algorithm has constant probability of trying a constant fraction of them before finding v_{i+h}, the one leading to τ.

Let us make this intuition more precise. For each choice of faulty locations F and treasure location $\tau = v$, we define $u(v, F)$ to be the first node on the path $[\sigma, v]$ such that u and its

$h-1$ "direct" descendants towards v, i.e., the next $h-1$ nodes on the path to v, are faulty, if such u exists, and otherwise we say $u(v,F)$ is *not defined*.

Denote by $\mathbf{H}(v,F)$ the subtree rooted at $u(v,F)$ of depth h. A central object in the proof is $\mathbf{H}^*(v,F)$ which corresponds to the couple $(\mathbf{H}(v,F),(F\cap\mathbf{H}(v,F)))$ (the subtree together with the faulty locations on it). Henceforth, we will often drop the dependency on v,F in the interest of readability, but we keep the bold notation to emphasize that $\mathbf{H}$ is a random object (it depends on F). If $u(v,F)$ is not defined, we also say that $\mathbf{H}$ is not defined.

The following lemma lower bounds the probability that $\mathbf{H}$ is well defined.

Lemma 6.1. *Let $0<\delta<\frac{1}{16}$. If h satisfies $q^h\geq\frac{8\delta h}{D}$ and $D\geq\max[h,8\delta h]$ (for D that does not satisfy this condition the statement does not make sense), then $P(\boldsymbol{H}$ is well defined$)\geq 4\delta$.*

Proof. Recall that we write $[\sigma,\tau]:=\{v_0=\sigma,v_1,\ldots,v_{D-1},v_D=\tau\}$. For a given $i,h\in\mathbb{N}$, let us denote by $F_{i,h}$ the event that $[v_i,v_{i+h-1}]$ is completely faulty.

With this definition, $\mathbf{H}$ is well defined if the event $F_{i,h}$ holds for at least one value of i in the range $0\leq i\leq D-h$. Hence what we want to show is that

$$P\left(\text{NOT}\bigcup_{i=0}^{D-h}F_{i,h}\right)\leq 1-4\delta. \tag{9}$$

For every fixed i and h, $P(F_{i,h})=q^h$. Indeed, there are h nodes on the path $[v_i,v_{i+h-1}]$ and each is independently faulty with probability q. The parameter h satisfies $q^h\geq\frac{8\delta h}{D}$ by assumption.

The events $F_{i'\cdot h,h}$ are independent when i' varies in $[D/h]$. The probability that none of these holds is

$$(1-q^h)^{D/h}\leq\left(1-\frac{8\delta h}{D}\right)^{\frac{D}{h}}\leq e^{-\frac{8\delta h}{D}\frac{D}{h}}=e^{-8\delta}\leq 1-8\delta/2=1-4\delta.$$

The last inequality holds for sufficiently small δ (e.g., $\delta\leq\frac{1}{16}$). □

From now on, we assume for simplicity that h is chosen so that $q^h=\frac{8\delta h}{D}$. (h being an integer, this is only an approximate equality in general. We ignore this point in the discussion, assuming h has been appropriately rounded.) Recall that $q=\Delta^{-\varepsilon}$, so taking logarithms we see that $\varepsilon h=\log_\Delta\frac{D}{\delta}-\log_\Delta 8h$. Viewing δ and Δ as fixed and letting D go to infinity, the previous equality entails that $h=\frac{1}{\varepsilon}(1-o_D(1))\log_\Delta\frac{D}{\delta}$, where the term $o_D(\cdot)$ tends to 0 when D tends to infinity[1]. Let $B(\mathbf{H}^*)$ denote the number of completely faulty leaves in $\mathbf{H}^*=\mathbf{H}^*(v,F)$. Lemma 6.2 below is proven in Section 6.2. It bounds the typical value of $B(\mathbf{H}^*)$ in those cases that $\mathbf{H}$ is well defined.

Lemma 6.2. *Let C be a sufficiently large constant that depends only on q. Then*

$$P\left(B(\boldsymbol{H}^*)\leq(q\Delta)^{h-C}\mid\boldsymbol{H}\text{ is well defined}\right)\leq 0.5.$$

The following two intermediate results express how "indistinguishable" is formalized in this context.

[1]Indeed, h tends to infinity when D tends to infinity, so $\log_\Delta 8h=o(h)$.

Claim 6.3. *Consider a leaf v and a subtree $\boldsymbol{H}^*$ (together with the faulty locations in it). Then, the value of $p_{v,H^*} := P_F(\boldsymbol{H}^*(v,F) = H^* \mid \tau = v)$ is the same for all v such that $p_{v,H^*} > 0$.*

Proof. Let H be a fixed subtree of depth h rooted at some node u. By definition, the statement that $P(\mathbf{H}^*(v,F) = H^* \mid \tau = v) > 0$ is equivalent to the following two statements:

- $\mathcal{A}$: There are no h consecutive faulty nodes in $[\sigma, u]$ and, if $u \neq \sigma$, then u's parent is not faulty.
- $\mathcal{B}$: Leaf v is a descendant of u and the leaf of H^* which is an ancestor of v is completely faulty in H^*.

The probability of $\mathcal{A}$ depends only (in some complicated way) on the length of $[\sigma, u]$ and q and hence does not depend on v. With these notations, if v,H^* and F are such that $\mathcal{B}$ holds, then

$$P(\mathbf{H}^*(v,F) = H^* \mid \tau = v) = q^{|F \cap H|}(1-q)^{|H \setminus F|} P(\mathcal{A}).$$

The right hand side does not depend on v. The claim follows. □

Lemma 6.4. *Conditioning on the subtree $\boldsymbol{H}^*$ (and hence its existence), the leaf of $\boldsymbol{H}^*$ which leads to the treasure is uniform amongst all completely faulty leaves v of $\boldsymbol{H}^*$.*

Proof. Denote by $\mathcal{L}$ the set of leaves of T. Using Bayes rule, and because we assume that τ is uniform over all leaves $\mathcal{L}$,

$$P(\tau = v \mid \mathbf{H}^*) = P(\tau = v) \cdot \frac{P(\mathbf{H}^* \mid \tau = v)}{P(\mathbf{H}^*)} = \frac{1}{|\mathcal{L}|} \cdot \frac{P(\mathbf{H}^* \mid \tau = v)}{P(\mathbf{H}^*)}.$$

It follows that $P(\tau = v \mid \mathbf{H}^*)$ has the same value for all leaves v of T such that $P(\tau = w \mid \mathbf{H}^*) > 0$. Indeed, we saw that the right hand term is independent of w, as soon as w is a descendant of a completely faulty leaf of $\mathbf{H}^*$ (Claim 6.3), and otherwise it is 0.

Since the tree T is complete and regular, each leaf of $\mathbf{H}^*$ is the ancestor of the same number of leaves in T, and so each completely faulty leaf of $\mathbf{H}^*$ is equally likely to lead to the treasure. □

We now condition on the event that $\mathbf{H}$ is well defined and that it has more than $s = (q\Delta)^{h-C}$ completely faulty leaves. This event holds with probability at least $4\delta \times 0.5$ (combining the results of Lemma 6.1 and Lemma 6.2). Under this conditioning, with probability at least 0.5 over treasure location the completely faulty leaf leading to the treasure is visited after at least $0.5s$ other faulty leaves have been visited. Indeed there are s faulty leaves, each being equally likely to lead to the treasure (Lemma 6.4). Overall, with probability $4\delta \cdot 0.5 \cdot 0.5 = \delta$, more than $0.5s$ nodes need to be visited. We saw that, $h = \frac{1}{\varepsilon}(1 - o_D(1)) \log_\Delta(\frac{D}{\delta})$, hence $s = (q\Delta)^{h-C} = (q\Delta)^{h \cdot (1-o_D(1))} = (\Delta^{1-\varepsilon})^{\frac{1}{\varepsilon}(1-o_D(1)) \log_\Delta \frac{D}{\delta}}$. After simplification, this is $(\delta^{-1}D)^{\frac{1-\varepsilon}{\varepsilon}(1-o_D(1))}$.

6.2 Proof of Lemma 6.2

The proof of Lemma 6.2 is broken into intermediate claims. To begin with we ignore the treasure τ, and consider a fixed complete Δ-ary tree H of depth h, with root σ. Each node of H is faulty (namely, belongs to the set F) independently with probability q. Let B denote the number of completely faulty leaves in H.

Claim 6.5. *It holds that* $P\left(B > \frac{1}{2}(q\Delta)^h\right) = \Omega(q)$.

Proof. The proof uses a second moment argument. For every given leaf, the probability of the full path from the root being faulty is q^h and there are Δ^h leaves. Hence $\mathbb{E}(B) = (q\Delta)^h$. Let us denote by $\mathcal{L}$ the set of leaves. Using the Boolean indicator variable χ_i to denote that leaf i is completely faulty, we have $B = \sum_i \chi_i$. Hence, we may write $\mathbb{E}(B^2) = \mathbb{E}((\sum_i \chi_i)^2)$ as

$$\mathbb{E}(B^2) = \mathbb{E}(B) + \sum_{u \neq v \in \mathcal{L}} P(u \text{ and } v \text{ are completely faulty}),$$

where the sum is taken on the ordered pairs $u \neq v$ where both are in $\mathcal{L}$. Fix a leaf $v \in \mathcal{L}$, and let $\mathcal{L}_\ell$ denote the set of leaves whose common ancestor with v has depth $h - \ell$. For every $\ell \in [1, h]$, $|\mathcal{L}_\ell| \leq \Delta^\ell$. Moreover, for $u \in \mathcal{L}_\ell$, u and v being completely faulty is equivalent to v being completely faulty and the $\ell - 1$ nodes connecting u to the root-to-v path being faulty. Hence, $P(u \text{ and } v \text{ are completely faulty}) = q^{\ell+h-1}$. Altogether,

$$\begin{aligned}\mathbb{E}(B^2) \leq (q\Delta)^h + \Delta^h \sum_{\ell=1}^{h} \Delta^\ell q^{\ell+h-1} &= O\left(q^{-1}(q\Delta)^h \sum_{\ell=1}^{h} (q\Delta)^\ell\right) \\ &= O\left(q^{-1}(q\Delta)^{2h}\right) = O(q^{-1}\mathbb{E}(B)^2).\end{aligned}$$

Using the Paley-Zygmund inequality, we get $P(B \geq \frac{1}{2}\mathbb{E}(B)) \geq \frac{1}{4}\frac{\mathbb{E}(B)^2}{\mathbb{E}(B^2)} = \Omega(q)$. □

Claim 6.6. *For a constant C that depends only on q, $P(B \leq (q\Delta)^{h-C} \mid B \geq 1) \leq 0.5$.*

Proof. First observe that for any constant C, we may consider only $h \geq C$, since otherwise, if $h < C$, then $(q\Delta)^{h-C} < 1$ and the requirement trivially holds.

Since $B \geq 1$ there exists a path $[\sigma, v]$ which is completely faulty. For every $u \in [\sigma, v]$ define T_u as the subtree rooted at u excluding the subtree rooted at the child of u on $[\sigma, v]$. The subtrees T_u are pairwise disjoint and form a partition of T. For $u \in [\sigma, v]$, define $B_u := B(T_u)$, the number of completely faulty leaves of T_u. Note that $B = \sum_u B_u \geq \max_u B_u$. Moreover since, for $u \neq u'$, $T_u \cap T_{u'} = \emptyset$, the variables B_u are independent.

Let S be the prefix of size C of $[\sigma, v]$. All subtrees rooted at a node $u \in S$ are of depth $> h - C$. Since $q\Delta > \Delta^{1/2}$, we have $(q\Delta)^{-C} < 1/2$ for every $C \geq 2$. Using Claim 6.5, together with the independence of the B_u's, the probability that all B_u's are smaller than $(q\Delta)^{h-C} \leq \frac{1}{2}(q\Delta)^h$ is less than $(1 - cq)^C$ for a constant c. The result follows, if C is large enough (as a function of q). □

If it exists, by definition, $\mathbf{H}^*$ has at least one completely faulty leaf, which is the one leading to τ. Outside of the branch leading to τ, the nodes of $\mathbf{H}^*$ are still independently faulty with probability q. This means that the number of completely faulty leaves of $\mathbf{H}^*$, $B(\mathbf{H}^*) \mid \{\mathbf{H}^* \text{ is well defined}\}$ is distributed as $B(H^*) \mid \{B(H^*) \geq 1\}$ for every fixed subtree H of depth h.

Using this together with Claim 6.6 finishes the proof of Lemma 6.2. Indeed, we obtain

$$P\left(B(\mathbf{H}^*) \leq (q\Delta)^{h-C} \mid \mathbf{H} \text{ is well defined}\right) \leq 0.5.$$

7 Open Problems

As mentioned, the model with permanent noise can be extended to general graphs. Essentially, when a node u is correct its advice points to one of the neighbors of u on a shortest path to the treasure. As there might be several such neighbors, one may consider an adversary that chooses which of these neighbors to point to. In the purely probabilistic setting, with probability q, each node is faulty, in which case its advice points to a random neighbor. The semi-adversarial setting can similarly be defined. Obtaining efficient search algorithms for general graphs is highly intriguing. Even though the likelihood of a node being the treasure under a uniform prior can still be computed in principle, it is not clear how to compare two nodes as in Theorem 2.3 because there may be more than a single path between them.

In a limited regime of noise, we believe that memoryless strategies might very well be efficient also on general graphs, and we pose the following conjecture. Proving it may require the use of tools from the theory of Random Walks in Random Environments, which seem to be lacking in the context of general graph topologies.

Conjecture 7.1. *There exists a probabilistic following algorithm that finds the treasure in expected linear time on every undirected graph assuming $q < c/\Delta$ for a small enough $c > 0$.*

References

[1] Javed A. Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. *STOC*, pages 486–493, 1991.

[2] Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999.

[3] Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). *FOCS*, pages 221–230, 2008.

[4] Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching on trees with noisy memory. *CoRR*, abs/1611.01403, 2016.

[5] Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching a tree with permanently noisy advice. *ESA*, 2018.

[6] Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. *STOC*, pages 130–136, 1993.

[7] Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. *SODA*, pages 268–276, 2008.

[8] Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 347–358, 2007.

[9] Ferdinando Cicalese and Ugo Vaccaro. Optimal strategies against a liar. *Theor. Comput. Sci.*, 230(1-2):167–193, 2000.

[10] Amin Coja-Oghlan. Solving NP-hard semirandom graph problems in polynomial expected time. *Journal of Algorithms*, 62(1):19–46, 2007.

[11] Argyrios Deligkas, George B. Mertzios, and Paul G. Spirakis. Binary search in graphs revisited. *Algorithmica*, 81(5):1757–1780, 2019.

[12] Dariusz Dereniowski, Stefan Tiegel, Przemysław Uznański, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. *SOSA@SODA: 4:1-4:17*, 2019.

[13] Alexander Drewitz and Alejandro F. Ramiréz. Selected topics in random walk in random environment. *Topics in Percolative and Disordered Systems, Springer Proceedings in Mathematics and Statistics*, 69:23–83, 2014.

[14] Ehsan Emamjomeh-Zadeh and David Kempe. A general framework for robust interactive learning. *NIPS*, 2017.

[15] Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. *STOC*, pages 519–532, 2016.

[16] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994.

[17] Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient search trees. *SODA*, pages 547–553, 2007.

[18] Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). *STOC*, pages 101–110, 2004.

[19] Ehud Fonio, Yael Heyman, Lucas Boczkowski, Aviram Gelblum, Adrian Kosowski, Amos Korman, and Ofer Feinerman. A locally-blazed ant trail achieves efficient collective navigation despite limited information, eLife 2016;5:e20185. *eLife*, 2016.

[20] Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a target with an agent guided by unreliable local advice: How to beat the random walk when you have a clock? *PODC*, pages 355–364, 2010.

[21] Nicolas Hanusse, Dimitris Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theoretical Computer Science*, 402(2-3):190 – 198, 2008.

[22] Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004.

[23] Avinatan Hassidim and Yaron Singer. Submodular optimization under noise. *COLT*, pages 1069–1122, 2017.

[24] Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. *SODA*, pages 881–890, 2007.

[25] Michael Krivelevich and Dan Vilenchik. Solving random satisfiable 3CNF formulas in expected polynomial time. *SODA*, pages 454–463, 2006.

[26] Eduardo Sany Laber and Loana Tito Nogueira. Fast searching in trees. *Eletronic Notes on Discrete Mathematics*, 2001.

[27] Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. *SODA*, pages 1096–1105, 2008.

[28] Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. *FOCS*, pages 379–388, 2006.

[29] Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

[30] Alain-Sol Snitzman. Topics in random walks in random environment. *ICTP Lecture Notes Series*, 2004.