Searching a Tree with Permanently Noisy Advice

Lucas Boczkowski, Amos Korman, Yoav Rodeh

Searching a Tree with Permanently Noisy Advice

Lucas Boczkowski
CNRS, IRIF, Univ. Paris Diderot, Paris, France
lucas.boczkowski@irif.fr

Amos Korman
CNRS, IRIF, Univ. Paris Diderot, Paris, France
amos.korman@irif.fr

Yoav Rodeh
Dep. of Physics of Complex Systems. Weizmann Institute, Rehovot, Israel
yoav.rodeh@gmail.com

Abstract

We consider a search problem on trees using unreliable guiding instructions. Specifically, an agent starts a search at the root of a tree aiming to find a treasure hidden at one of the nodes by an adversary. Each visited node holds information, called *advice*, regarding the most promising neighbor to continue the search. However, the memory holding this information may be unreliable. Modeling this scenario, we focus on a probabilistic setting. That is, the advice at a node is a pointer to one of its neighbors. With probability q each node is *faulty*, independently of other nodes, in which case its advice points at an arbitrary neighbor, chosen uniformly at random. Otherwise, the node is *sound* and points at the correct neighbor. Crucially, the advice is *permanent*, in the sense that querying a node several times would yield the same answer. We evaluate efficiency by two measures: The *move complexity* denotes the expected number of edge traversals, and the *query complexity* denotes the expected number of queries.

Let Δ denote the maximal degree. Roughly speaking, the main message of this paper is that a phase transition occurs when the *noise parameter* q is roughly $1/\sqrt{\Delta}$. More precisely, we prove that above the threshold, every search algorithm has query complexity (and move complexity) which is both exponential in the depth d of the treasure and polynomial in the number of nodes n. Conversely, below the threshold, there exists an algorithm with move complexity $\mathcal{O}(d\sqrt{\Delta})$, and an algorithm with query complexity $\mathcal{O}(\sqrt{\Delta}\log\Delta\log^2 n)$. Moreover, for the case of regular trees, we obtain an algorithm with query complexity $\mathcal{O}(\sqrt{\Delta}\log n\log\log n)$. For q that is below but close to the threshold, the bound for the move complexity is tight, and the bounds for the query complexity are not far from the lower bound of $\Omega(\sqrt{\Delta}\log_\Delta n)$.

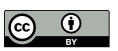
In addition, we also consider a *semi-adversarial* variant, in which faulty nodes are still chosen at random, but an adversary chooses (beforehand) the advice of such nodes. For this variant, the threshold for efficient moving algorithms happens when the noise parameter is roughly $1/\Delta$. In fact, above this threshold a simple protocol that follows each advice with a fixed probability already achieves optimal move complexity.

2012 ACM Subject Classification Theory of Computation → Probabilistic Computation, Database Theory, Theory of Randomized Search Heuristics

Keywords and phrases Data structures - Graph search - Average Case Analysis - Permanents Faults

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.54

Funding 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).



1 Introduction

This paper considers a basic search problem on trees, in which the goal is to find a treasure that is placed at one of the nodes by an adversary. Each node of the tree holds information, called *advice*, regarding which of its neighbors is closer to the treasure, and the search may consult the advice at some nodes in order to accelerate the search. Crucially, we assume that advice at nodes may be faulty with some probability. Many works consider noisy queries in the context of search, but it is typically assumed that queries can be resampled (see e.g., [11, 18, 4, 10]). In contrast, we assume that each location is associated with a single *permanent* advice. That is, faults are in the physical memory associated with the node, and hence querying the node again would yield the same answer. This difference is dramatic, as the search under our model does not allow for simple amplification procedures (similar to [6] albeit in the context of sorting). Searching in contexts of permanently faulty nodes has been studied in a number of works [7, 12, 15, 16, 17], but only 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 techniques (see more details in Section 1.3).

1.1 The Noisy Advice Model

We start with some notation. Further notations are given in Section 1.4. Let T be an n-node tree[1] rooted at some arbitrary node σ. 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. 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 maximal depth of a node. Finally, let Δ_u denote the degree of node u and let Δ denote the maximal degree in the tree.

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 vertex $u \neq \tau$ is *faulty* with probability q_u (the meaning of being faulty will soon be explained). Otherwise, u is considered *sound*, in which case its advice is correct. We call q_u the *noise parameter* of u, and define the *general noise parameter* as $q = \max\{q_u \mid u \in T\}$.

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 oblivious adversary. That is, an adversary specifies for each node what advice it would have assuming it is faulty. Then, faulty nodes are still chosen randomly as in the main model, but their advice is specified by the adversary.

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 protocol terminates when the agent queries the treasure. We evaluate a search algorithm A by two measures: The move complexity, termed $\mathcal{M}(A)$, is the expected

[1] We present the model for trees, but it should be clear that it can be similarly defined for arbitrary graphs (see also Section 5).

number of edge traversals, and the query complexity, termed $\mathcal{Q}(A)$, is the expected number of queries[2]. Expectation is taken over both the randomness involved in sampling advice and the possible probabilistic choices made by A. We note that when considering walking algorithms, we assume that the agent does not know the structure of the tree in advance, and discovers it as it moves. Conversely, when focusing on minimizing the query complexity only, we assume that the tree structure is known to the algorithm.

The noise parameters $(q_u)_{u \in T}$ govern the accuracy of the environment. On the one extreme, if $q_u = 0$ for all nodes, then advice is always correct. This case allows to find the treasure in d moves, by simply following each encountered advice. Alternatively, it also allows to find the treasure using $\mathcal{O}(\log n)$ queries, by performing a separator based search. On the other extreme, if $q_u = 1$ for all nodes, 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.

The reader wishing to increase its familiarity with the model and its algorithmic challenges may read Appendix A, where a natural greedy algorithm is analyzed, and shown to be inefficient.

1.2 Our Results

Consider the noisy advice model on trees with maximum degree Δ and depth D. 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 complexities.

The proof that there is no algorithm with low expected complexities when the noise exceeds $1/\sqrt{\Delta}$ is rather simple, and in fact, holds even if the algorithm has access to the advice of all internal nodes. Intuitively, the argument is as follows (the formal proof appears in Section 4.1). Consider a complete Δ-ary tree of depth D and assume that the treasure τ is placed at a leaf. The first observation (Lemma 10) is that the expected number of leaves having more advice point to them than to τ is a lower bound on the query complexity. The next observation is that there are roughly Δ^D leaves whose distance from τ is $2D$. 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 $\Delta^D q^{2D}$, which explodes when $q \gg 1/\sqrt{\Delta}$. This essentially establishes the lower bound for the noise regime.

The main technical difficulties we had to face appeared when we aimed to show that the $1/\sqrt{\Delta}$ lower bound is, in fact, tight, and moreover, that there exist extremely efficient algorithms when the noise is above the threshold. In this regard, we note two technical contributions. The first appears in the construction of the moving algorithm $\mathtt{A}_{walk}$. Even though the algorithm should be designed to quickly find an adversarially placed treasure, it is in fact based on a Bayesian approach. The challenging part is identifying the correct prior. Constructing algorithms that ensure worst-case guarantees through a Bayesian approach was done in [4] which studies a closely related, yet much simpler problem of search on the line. Apart from [4] we are unaware of other works that follow this approach. The second technical contribution corresponds to the query setting, where we mimic the resampling of advice at separator nodes, by locally applying the moving algorithm.

[2] The success probability after a fixed number of rounds is another quantity of interest. It is left for future work.

1.2.1 Upper Bounds

In Section 2, we present a walking algorithm that 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.

Using this walking algorithm, we derive two query algorithms (in Section 3). The first is optimal up to a factor of $\mathcal{O}(\log^2(\Delta)\log n)$ and the second is restricted to regular trees, but is optimal up to a factor of $\mathcal{O}(\log(\Delta)\log\log n)$. Note that the query algorithms use the knowledge of the tree structure, as well as bounds on the regime of noise.

Before stating our theorems, we need the following definition.

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

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

Note that since $\Delta_v \geq 2$, the condition is always satisfiable when taking a small enough ε. In the following theorems the $\mathcal{O}$ notation hides only a polynomial a polynomial term in $1/\varepsilon$.

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

▶ **Theorem 2.** *There exists a deterministic walking algorithm $\mathtt{A}_{walk}$ such that for any constant $\varepsilon > 0$, if Condition ($\star$) holds with parameter ε then $\mathcal{M}(\mathtt{A}_{walk}) = \mathcal{O}(\sqrt{\Delta}d)$.*

▶ **Theorem 3.** 1. *For any $\varepsilon > 0$, there exists a deterministic query algorithm $\mathtt{A}_{sep}$ such that if Condition ($\star$) holds with parameter ε then the query complexity is $\mathcal{Q}(\mathtt{A}_{sep}) = \mathcal{O}(\sqrt{\Delta}\log\Delta\cdot\log^2 n)$.*
2. *Assume that $q < c/\sqrt{\Delta}$ for a small enough constant $c > 0$. Then there exists a deterministic query algorithm $\mathtt{A}_{2-layers}$ such that, restricted to (not necessarily complete) Δ-ary trees, $\mathcal{Q}(\mathtt{A}_{2-layers}) = \mathcal{O}(\sqrt{\Delta}\log n\cdot\log\log n)$.*

1.2.2 Lower Bounds

We establish (in Section 4 and Appendix G) the following lower bounds.

▶ **Theorem 4.** *The following holds for any randomized algorithm A and any integer $\Delta \geq 3$.*
1. ***Exponential complexity above the threshold.***
   Consider a complete Δ-ary tree. For every constant $\varepsilon > 0$, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}}\cdot(1+\frac{1}{\Delta-1})$, then both $\mathcal{Q}(A)$ and $\mathcal{M}(A)$ are exponential in D.
2. ***Lower bounds for any q.***
   (a) Consider a complete Δ-ary tree. Then $\mathcal{Q}(A) = \Omega(q\Delta\log_\Delta n)$.
   (b) For any integer d, there is a tree with at most $d\Delta$ nodes, and a placement of the treasure at depth d, such that $\mathcal{M}(A) = \Omega(dq\Delta)$.

Observe that taken together, Theorems 2,4,3 and Condition ($\star$) imply that for any $\varepsilon > 0$ and large enough Δ, efficient search can be achieved if $q < (1-\varepsilon)/\sqrt{\Delta}$ but not if $q > (1+\varepsilon)/\sqrt{\Delta}$.

1.2.3 Memory-less Algorithms

Query algorithms assume the knowledge of the tree and hence cannot avoid memory complexity which is linear in n. In contrast, our walking algorithm $\mathtt{A}_{walk}$ uses memory that is composed of advice accumulated during the walk, and hence remains low, in expectation.

Finally, we analyse the performance of simple memoryless algorithms called *probabilistic following*, suggested in [14]. 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 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 any 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 a semi-adversarial model. In fact, it turns out that in the semi-adversarial model, probabilistic following algorithms are the best possible, as the threshold for efficient search, with respect to any algorithm, is roughly $1/\Delta$. Due to lack of space these results are discussed and proved in Appendix H.

1.3 Related Work

In computer science, search algorithms have been the focus of numerous works. Due to their importance, trees are particularly popular structures to investigate search, see e.g., [19, 3, 21, 20]. Within the literature on search, many works considered noisy queries [11, 18, 10], however, it was typically assumed that noise can be *resampled* at every query. As mentioned, dealing with permanent faults incurs 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 $\mathcal{O}(1)$ expected number of queries. In contrast, in our model, it can be easily seen that $\Omega(n)$ is a lower bound for both the 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 [15, 16, 17], as part of the more general framework of the *liar models* [1, 2, 5, 8, 22]. Data structures with adversarial memory faults have been investigated in the so called faulty-memory RAM model introduced in [13]. In particular, data structures supporting the same operations as search trees with adversarial memory faults were studied in [12, 7]. Interestingly, the data structures developed in [7] 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. All these worst case models are, however, significantly different from the randomized one we consider, both in terms of techniques and results. The subject of queries with probabilistic memory faults, as the ones we study here, has been explicitly studied in the context of sorting [6].

The noisy advice model considered in this paper actually originated in the recent biologically centered work [14], aiming to abstract navigation relying on guiding instructions in the context of collaborative transport by ants. There, 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. In that work, the authors modelled 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)

[23, 9]. 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.

1.4 Notations

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.

1.5 Organization

In Section 2 we present our optimal walking algorithm. Section 3 presents our query algorithms, while most of the details regarding the more elaborated algorithm on regular trees are deferred to Appendix E. In Section 4 we show the lower bounds for both the move and query complexities. In Section 5, we give a list of open problems. Theorem 5 and the threshold of $\Theta(1/\Delta)$ that applies to the semi-adversarial setting are proved in Appendix H.

2 Optimal Walking Algorithm

In this section we prove Theorem 2. At a very high level, at any given time, the walking 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.

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 induced by assuming that it is placed in one of the vertices according to some known distribution and see how they measure up in our worst case setting. As mentioned, this approach is similar to [4], which studies the closely related, yet much simpler problem of search on the line. Of course, the success of this scheme highly depends on the choice of the prior distribution we take.

To make our life easier, let us first assume that the structure of the tree is known to the algorithm. Also, we assume the treasure is placed in one of the leaves of the tree according to some known distribution θ, and denote by $\texttt{adv}$ 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 is never for vertices in $T(u)$ as 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 $p_w + q_w/\Delta_w$ as it is normally quite close to 1, 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 do not want to assume that our algorithm knows q_w, but we do assume that in the worst scenario $q_w \sim 1/\sqrt{\Delta}_w$. Assigning this value and rescaling we finally define:

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

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[3]. 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[4]. 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 to take 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, we show in Appendix C that this approach fails on other (non-complete) Δ-ary trees.

[3] It is tempting to define $\texttt{score}(u)$ as the sum of weighted advice from the root to u. However, when comparing two vertices, the advice of their least common ancestor would be counted twice, which we prefer to avoid.

[4] Actually, a similar formula could be derived choosing θ to be the uniform distribution over all nodes, but for technical reasons it is easier to restrict it to leaves only.

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

In our context, there is no distribution over treasure location and we are free to choose θ as we like. We 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:

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

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 wonderful 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. It is hard to imagine distributions on leaves that allow us to calculate the probability of being in a subtree without knowing anything about it!

In the walking algorithm, if v is a candidate for exploration, these nodes 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 6. 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 $\mathtt{adv}$ define:

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

Algorithm $\mathtt{A}_{walk}$ 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. Note that 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 Definition 1. The next lemma provides a large deviation bound tailored to our setting. The proof appears in Appendix B.

▶ **Lemma 7.** *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 $(p_i + \frac{q_i}{\Delta_i}, q_i(1 - \frac{2}{\Delta_i}), \frac{q_i}{\Delta_i})$, 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,*

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

The next theorem states that $\mathtt{A}_{walk}$ is optimal up to a constant factor for the regime of noise below the threshold. It establishes Theorem 2.

▶ **Theorem 8.** *Assume that Condition $(\star)$ holds for some fixed $\varepsilon > 0$. Then $\mathcal{M}(\mathtt{A}_{walk}) = \mathcal{O}(d\sqrt{\Delta})$, where 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 time 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,

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

Where $C(u_k) = T(u_{k-1}) \setminus T(u_k)$, and so $\cup_{i=1}^{k} C(u_i) = T \setminus T(u_k)$. Recall that scores are defined so that u has a larger score than u_k, if the sum of weighted arrows on the path $\langle u_k, u \rangle$ is at least $\frac{2}{3}\log(\beta(u)/\beta(u_k))$. Setting m to be this value, Lemma 7 allows to calculate this probability exactly. 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 $c = 1 - \varepsilon$,

$$\frac{E_k}{2} \leq \sum_{i=1}^{k} \sum_{u \in C(u_i)} \frac{c^{d(u_k,u)-1}}{e^{\frac{3}{4}\cdot\frac{2}{3}\log\left(\frac{\beta(u)}{\beta(u_k)}\right)}} \sqrt{\prod_{v \in \langle u, u_k \rangle} \frac{1}{\Delta_v}} \cdot d(u_k, u)$$

$$= \frac{1}{c} \sum_{i=1}^{k} \sum_{u \in C(u_i)} \frac{c^{d(u_k,u)}}{\sqrt{\frac{\beta(u)}{\beta(u_k)}}} \sqrt{\frac{\Delta_{u_i}}{\frac{\beta(u_k)}{\beta(u_i)} \cdot \frac{\beta(u)}{\beta(u_i)}}} \cdot d(u_k, u)$$

$$\leq \frac{\sqrt{\Delta}}{c} \sum_{i=1}^{k} c^{d(u_k,u_i)} \sum_{u \in C(u_i)} c^{d(u_i,u)} \frac{\beta(u_i)}{\beta(u)} \cdot \Big(d(u_k, u_i) + d(u_i, u)\Big).$$

By Claim 6, 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:

$$\frac{E_k}{2} \leq \frac{\sqrt{\Delta}}{c(1-c)} \sum_{i=1}^{k} c^{d(u_k,u_i)} d(u_k, u_i) + \frac{\sqrt{\Delta}}{(1-c)^2} \sum_{i=1}^{k} c^{d(u_k,u_i)}$$

$$\leq \frac{(1+c)\sqrt{\Delta}}{(1-c)^3} \leq \frac{2\sqrt{\Delta}}{\varepsilon^3} = \mathcal{O}\left(\sqrt{\Delta}\right),$$

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

3 Query Algorithms

3.1 An $\mathcal{O}(\sqrt{\Delta}\log\Delta\log^2 n)$ Queries Algorithm

Our next goal is to prove the first item in Theorem 3. As is common in search on trees, our technique in this section is based on separators. We say a node u is a *separator* of T if all the connected components of $T \setminus \{u\}$ are of size at most $|T|/2$. It is well known that such a node exists. Assume there is some local procedure, that given a vertex u decides with probability $1-\delta$ in which one of the connected components of $T \setminus \{u\}$, the treasure resides. Applying this procedure on a separator of the tree, and then focusing the search recursively only on the component it pointed out, results in a type of algorithm we call a *separator based* algorithm. It uses the local procedure at most $\lceil \log_2 n \rceil$ times, and by a union bound, finds the treasure with probability at least $1 - \lceil \log_2 n \rceil \delta$. Broadly speaking, we will be interested in the expected running time of this sort of algorithm conditioned on it being successful. This sort of conditioning complicates matters slightly. In what follows, we assume that the set of separators for the tree is fixed.

Proof. *(of the first item in Theorem 3)* The algorithm we build is denoted $\mathtt{A}_{sep}$. It runs a separator based algorithm in parallel to some arbitrary exhaustive search algorithm. The meaning of *in parallel* here simply means that the two algorithms are run in an alternating fashion. Fix some small h. The local exploration procedure, denoted $\mathtt{local}_h$, for a vertex u proceeds as follows.

Procedure $\mathtt{local}_h(u)$. Consider the tree $T_h(u)$ rooted at u consisting of all vertices satisfying $\log_\Delta \beta(v) < h$ together with their children. So a leaf of $v \in T_h(u)$ is either a leaf of T, or satisfies $\Delta^h \leq \beta(v) < \Delta^{h+1}$. Denote the second kind a *nominee*. Call a nominee *promising* if the number of weighted arrows pointing to v is large, specifically, if $\sum_{w \in [u,v\rangle} X_w \geq \frac{2}{3} h \log \Delta$, where $X_w = \log \Delta_w$ if the advice at w is pointing to v, $X_w = -\log \Delta_w$ if it is pointing to u, and $X_w = 0$ otherwise. Viewing it as a query algorithm, we now run the walking algorithm $\mathtt{A}_{walk}$ on $T_h(u)$ (starting at its root u) until it either finds the treasure or finds a promising nominee. In the latter case, $\mathtt{local}_h(u)$ declares that the treasure is on the connected component of $T \setminus \{u\}$ containing this nominee. If $\tau \in T_h(u)$ then set $\tau_u = \tau$. Otherwise let τ_u be the leaf of $T_h(u)$ closest to the treasure, and so in this case τ_u is a nominee. Say that u is *h-misleading* if either (1) $\tau \notin T_h(u)$ and τ_u is not promising, or (2) there is some promising nominee $v \in T_h(u)$ that is not in the same connected component of $T \setminus \{u\}$ as τ_u. Note that if u is not h-misleading then $\mathtt{local}_h(u)$ necessarily outputs the correct component of $T \setminus \{u\}$, namely, the one containing the treasure. The proof of the following lemma appears in Appendix D. The part regarding regular trees will be needed later.

▶ **Lemma 9.** *For any u, $\mathbb{P}(u \text{ is } h\text{-misleading}) \leq (\Delta+1)(1-\varepsilon)^h$. Also, for any event X such that X occurring always implies that u is not misleading, we have $\mathbb{P}(X)\,\mathcal{Q}(\mathtt{local}_h(u) \mid X) = \mathcal{O}(\sqrt{\Delta}\log\Delta \cdot h)$. In the case the tree is regular, these bounds become $2(1-\varepsilon)^h$ and $\mathcal{O}(\sqrt{\Delta}\cdot h)$ respectively. The constant hidden in the $\mathcal{O}$ notation only depends polynomially on $1/\varepsilon$.*

Taking $h = -3\log(2n)/\log(1-\varepsilon)$, gives $\mathbb{P}(u \text{ is misleading}) \leq 1/n^2$. Denote by $\mathtt{Good}$ the event that none of the separators encountered are misleading. By a union bound, $\mathbb{P}(\mathtt{Good}^{\,c}) \leq 1/n$.

$$\mathcal{Q}(\mathtt{A}_{sep}) = \mathbb{P}(\mathtt{Good})\,\mathcal{Q}(\mathtt{A}_{sep} \mid \mathtt{Good}) + \mathbb{P}(\mathtt{Good}^{\,c})\,\mathcal{Q}(\mathtt{A}_{sep} \mid \mathtt{Good}^{\,c}). \tag{1}$$

As $\mathtt{A}_{sep}$ runs an exhaustive search algorithm in parallel, the second term is $\mathcal{O}(1)$. For the first term, note that conditioning on $\mathtt{Good}$, all local procedures either find the treasure or give the correct answer, and so there are $\mathcal{O}(\log n)$ of them and they eventually find the treasure. Denote by u_i the i-th vertex that $\mathtt{local}_h$ is executed on. By linearity of expectation, and applying Lemma 9, the first term of (1) is $\mathbb{P}(\mathtt{Good}) \sum_i \mathcal{Q}(\mathtt{local}_h(u_i) \mid \mathtt{Good}) = \mathcal{O}(\log n \cdot \sqrt{\Delta} \log \Delta \cdot h) = \mathcal{O}(\sqrt{\Delta} \log \Delta \log^2 n)$. As $\log(1+x) > x$ always, then $-1/\log(1-\varepsilon) \leq 1/\varepsilon$, and the hidden factor in the $\mathcal{O}$ is as stated. ◀

3.2 An Almost Tight Result for Regular Trees

We now turn our attention to the second item in Theorem 3. Due to space constraints its full proof is deferred to Appendix E. At a high level, we run two algorithms in parallel (i.e., in an alternating fashion): $\mathtt{A}_{fast}$, and $\mathtt{A}_{mid}$. Algorithm $\mathtt{A}_{fast}$ is actually $\mathtt{A}_{sep}$ applied with parameter $h = \Theta(\log \log n)$ instead of $\Theta(\log n)$. Using Lemma 9, with probability $1 - 1/\log^{\mathcal{O}(1)}(n)$, the local procedure of $\mathtt{A}_{fast}$ always detects the correct component for each separator, and $\mathtt{A}_{fast}$ needs an expected number of $\mathcal{O}(\sqrt{\Delta} \cdot \log n \cdot \log \log n)$ queries to find the treasure. This is the running time we are aiming for.

Algorithm $\mathtt{A}_{mid}$ is similar to $\mathtt{A}_{sep}$ except it uses a different subroutine for local exploration. It then remains to show that it finds the treasure using a relatively low expected number of queries even conditioning on the event that caused $\mathtt{A}_{fast}$ to fail, namely, the event that there is a misleading separator at the scale $h = \Theta(\log \log n)$. The query complexity of $\mathtt{A}_{mid}$ does blow up under this event but we show that the blowup is not that bad, and can be compensated by the fact that the bad event has small probability. This is the core of the proof, and what requires most work. In fact, the complexity of the arguments led us to restrict the discussion to regular trees and also modify the subroutine for local exploration to ease the analysis.

4 Lower Bounds

We next prove Items (1) and (2a) of Theorem 4. Item (2b) is proved in Appendix G.2.

4.1 Exponential Complexity Above the Threshold

We wish to prove Item (1) in Theorem 4. Namely, that for every fixed $\varepsilon > 0$, and for every complete Δ-ary tree, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta}-1} \cdot (1 + \frac{1}{\Delta-1})$, then every randomized search algorithm has query (and move) complexity which is both exponential in the depth d of the treasure and polynomial in n. In fact, this lower bound holds even if the algorithm has access to the advice of all internal nodes. The following lemma is proved in Appendix G.1:

▶ **Lemma 10.** *Assume the treasure is placed in a leaf τ of the complete Δ-ary tree. Denote by* $\mathtt{adv}$ *the random advice on all internal nodes, then the expected number of leaves u satisfying $|\overrightarrow{\mathtt{adv}}(u)| > |\overrightarrow{\mathtt{adv}}(\tau)|$, is a lower bound on the query complexity of any algorithm.*

Using Lemma 10, all we need to do is approximate the number of leaves u satisfying $|\overrightarrow{\mathtt{adv}}(u)| > |\overrightarrow{\mathtt{adv}}(\tau)|$. When comparing the number of pointers that point towards each of two different nodes, only the pointers of the internal nodes on the path between them may influence on the result. The probability that a leaf u "beats" the treasure τ in the sense of Lemma 10, is at least the probability that exactly one node on the path points to u and

none of the rest point towards the treasure. This probability is at least

$$\frac{q}{\Delta} \cdot \left(q \cdot \left(1 - \frac{1}{\Delta} \right) \right)^{d(u,\tau)-2}.$$

There are precisely $(\Delta - 1)^D$ leaves whose distance from the treasure is $2D$. Therefore, the expected number of leaves that beat the treasure is at least:

$$\frac{q}{\Delta}(\Delta-1)^D q^{2D-2} \cdot \left(1 - \frac{1}{\Delta}\right)^{2D-2} = \frac{\Delta}{q(\Delta-1)^2} \cdot \left(\frac{q^2(\Delta-1)^3}{\Delta^2}\right)^D \geq \frac{\Delta}{q(\Delta-1)^2} \cdot (1+\varepsilon)^2 D.$$

Item (1) in Theorem 4 follows. ◀

4.2 A Lower Bound of $\Omega(\sqrt{\Delta} \cdot \log_\Delta n)$ when $q \sim 1/\sqrt{\Delta}$

We now prove Item (2a) in Theorem 4. We wish to prove that for $\Delta \geq 3$, on the complete Δ-ary tree of depth D, any algorithm needs $\Omega(q\Delta D)$ queries in expectation. Note that, in particular, when q is roughly $1/\sqrt{\Delta}$, and n is the tree size, the query complexity is $\Omega(\sqrt{\Delta} \cdot \log_\Delta n)$. Before proving this lower bound, we need the following observation (proved in Appendix G.3)

▶ Observation 11. Any randomized algorithm that finds a uniformly chosen treasure between k identical objects needs an at least $(k+1)/2$ queries in expectation.

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 any 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 11 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. ◀

5 Open Problems

Closing the small gap between the upper and lower bounds for the query setting remains open. The noisy advice model may well be interesting to study in other search settings. In particular, 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 so easy to compare two nodes as in Theorem 8 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 RWRE, which seem to be lacking in the context of general graph topologies.

▶ Conjecture 12. There exists a probabilistic following algorithm that finds the treasure in expected linear time on any undirected graph assuming $q < c/\Delta$ for a small enough $c > 0$.

References

1. Andrei Asinowski, Jean Cardinal, Nathann Cohen, Sébastien Collette, Thomas Hackl, Michael Hoffmann, Kolja B. Knauer, Stefan Langerman, Michal Lason, Piotr Micek, Günter Rote, and Torsten Ueckerdt. Coloring hypergraphs induced by dynamic point sets and bottomless rectangles. *CoRR*, abs/1302.2426, 2013.
2. Javed A. Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *STOC*, pages 486–493. ACM, 1991.
3. Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999.
4. Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *FOCS*, pages 221–230, 2008.
5. Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *STOC*, pages 130–136. ACM, 1993.
6. Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276, 2008.
7. Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 347–358, 2007.
8. Ferdinando Cicalese and Ugo Vaccaro. Optimal strategies against a liar. *Theor. Comput. Sci.*, 230(1-2):167–193, 2000.
9. 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.
10. Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *STOC*, pages 519–532, 2016.
11. Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994.
12. Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient search trees. In *SODA*, pages 547–553, 2007.
13. Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *STOC*, pages 101–110, 2004.
14. 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. 2016.
15. 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? In *PODC*, pages 355–364, 2010.
16. 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.
17. Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004.
18. Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007.
19. Eduardo Sany Laber and Loana Tito Nogueira. Fast searching in trees. In *Eletronic Notes on Discrete Mathematics*, 2001.
20. Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *SODA*, pages 1096–1105, 2008.

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

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

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

Appendix

A A Natural Attempt that Fails

The following example provides useful intuition. Consider the permissive scenario where the tree is known and all the advice at the internal nodes of the tree is available to the algorithm. Since advice is sampled independently at each node, it seems natural, at least at first glance, to associate each node with a "likelihood rank", being the total number of advice pointers pointing to it in the whole tree. A natural query algorithm $\mathtt{A}_{natural}$ would then be to query the nodes one by one, according to the resulting ranking. In other terms $\mathtt{A}_{natural}$ goes at each step to the unvisited node having most arrows pointing to it among the neighbors of previously seen nodes. Unfortunately, this naive strategy turns out highly inefficient. To see why, consider a complete Δ-ary tree of depth D, except for one child of the root, which is turned into a leaf, trimming its $(\Delta-1)^{D-1}$ descendants. Assume further that this particular child is in fact the treasure location τ. For any leaf $v \neq \tau$, with probability $\frac{q}{\Delta} \cdot q^{D-1}(1-\frac{1}{\Delta})^{D-1}$ the root points towards v and all the rest of the nodes on the path $\langle \sigma, \ell \rangle$ do not point upward (towards the treasure). This makes all nodes of the path better ranked than τ, and so $\mathtt{A}_{natural}$ would query them all before querying τ. There are $(\Delta-1)^D$ such leaves, and hence the expected number of nodes queried before the treasure is at least $q^D(\Delta-1)^{D-1}(1-\frac{1}{\Delta})^D$ in expectation. Even for q as small as c/Δ this number is exponential in the depth of the tree.

The main reason why $\mathtt{A}_{natural}$ fails is that, although the suggested ranking reflects the correct likelihood under a uniform prior of the treasure, this uniform prior is not tailored to our setting of adversarial treasure location. Instead, the algorithm we present in Section 2 relies on a ranking which incorporates in a clever way the tree structure.

B Proof of the Chernoff Estimate

▶ **Lemma 7** ***(restated).*** *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 $(p_i + \frac{q_i}{\Delta_i}, q_i(1-\frac{2}{\Delta_i}), \frac{q_i}{\Delta_i})$, 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,*

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

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

$$\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \geq m\right) = \mathbb{P}\left(e^{s\sum_{i=1}^{\ell} X_i} \geq e^{sm}\right) \leq \frac{\mathbb{E}\left[e^{s\sum_i X_i}\right]}{e^{sm}} = \frac{\prod_i \mathbb{E}\left[e^{sX_i}\right]}{e^{sm}}$$

$$= \frac{1}{e^{sm}} \prod_{i=1}^{\ell} \left(\frac{p_i + \frac{q_i}{\Delta_i}}{e^{\log(\Delta_i)s}} + q_i\left(1 - \frac{2}{\Delta_i}\right) + \frac{q_i}{\Delta_i} e^{\log(\Delta_i)s}\right)$$

$$\leq \frac{1}{e^{sm}} \prod_{i=1}^{\ell} \left(\frac{1}{\Delta_i^s} + q_i + q_i \Delta_i^{s-1}\right).$$

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

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

Where for the last step we used Condition $(\star)$ which says:

$$q_i < \frac{1 - \varepsilon - \Delta_i^{-\frac{1}{4}}}{\sqrt{\Delta_i} + \Delta_i^{\frac{1}{4}}} \implies q_i \Delta_i^{\frac{1}{2}} + q_i \Delta_i^{\frac{1}{4}} + \Delta_i^{-\frac{1}{4}} < 1 - \varepsilon \implies \Delta_i^{-\frac{3}{4}} + q_i + q_i \Delta_i^{-\frac{1}{4}} < \frac{1-\varepsilon}{\sqrt{\Delta_i}}$$

◀

C Taking θ to be the Uniform Distribution is not a Good Idea

As mentioned at the end of Section 2.1, when our tree is a complete Δ-ary tree, choosing θ to be the uniform distribution over the leaves results in an efficient algorithm with respect to the worst case placement of the treasure. Trying to tackle more general trees, perhaps the most natural a priori distribution is the uniform one over the nodes of the tree. As our technical presentation accommodates only distributions on leaves, we take θ to be uniform over the leaves only, and remark that the same result we get here applies to the former case.

Unfortunately, we show that this variant may take exponentially many queries before finding the treasure no matter what q is. This in fact follows from a similar argument to the one mentioned in Section A. The instance we consider is a complete Δ-ary tree of depth D, except for one child of the root, which is turned into a leaf, trimming its $(\Delta-1)^{D-1}$ descendants. We consider the case that this particular child is in fact the treasure location τ.

Recall from Section 2.1 that $\texttt{score}(u) > \texttt{score}(\tau)$ iff:

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

where $\theta(u)$ is now understood as the ratio between the number of leaves in $T(u)$ divided by the total number of leaves in T, as opposed to the total number of leaves if the tree was a complete tree.

In particular, consider any node u at distance $a \cdot D$ from the root for some $a < 2/5$. This node u owns a tree $T(u)$ of size $(\Delta-1)^{D(1-a)}$, hence $\theta(u) \sim (\Delta-1)^{-a}$. In contrast $\theta(\tau) = (\Delta-1)^{-D}$. Therefore,

$$\frac{2}{3} \log\left(\frac{\theta(\tau)}{\theta(u)}\right) = -\frac{2}{3} D(1-a) \log(\Delta),$$

where we write $\log(\Delta)$ in place of $\log(\Delta-1)$ as it does not change the nature of the result, only the choice of the constant $2/5$. On the other hand there are only $a\cdot\Delta$ nodes on the path $\langle u,\tau\rangle$ so the left side of (2) is always greater than $-aD\log\Delta$. In other words if $a<2/5$ then $-\frac{2}{3}D(1-a)\log\Delta < -aD\log\Delta$, and any node u at depth $a\cdot D$ has a better score than τ, regardless of the advice on the path $\langle\tau,u\rangle$ which means that our algorithm needs $(\Delta-1)^{2/5D}$ steps at least.

D Proof of Lemma 9

▶ **Lemma 9** ***(restated).*** *For any u, $\mathbb{P}(u \text{ is } h\text{-misleading}) \le (\Delta+1)(1-\varepsilon)^h$. Also, for any event X such that X occurring always implies that u is not misleading, we have $\mathbb{P}(X)\,\mathcal{Q}(\texttt{local}_h(u)\mid X) = \mathcal{O}(\sqrt{\Delta}\log\Delta\cdot h)$. In the case the tree is regular, these bounds become $2(1-\varepsilon)^h$ and $\mathcal{O}(\sqrt{\Delta}\cdot h)$ respectively. The constant hidden in the $\mathcal{O}$ notation only depends polynomially on $1/\varepsilon$.*

Proof. To check the probability that u is misleading, consider two cases:

1. $\tau\notin T_h(u)$, and τ_u is not promising. By Lemma 7, and recalling that $\Delta^h\le\beta(\tau_u)$, the probability τ_u is not promising is:

$$\mathbb{P}\left(\sum_{w\in[u,\tau_u\rangle} -X_w \le \frac{2}{3}h\log(\Delta)\right) = \mathbb{P}\left(\sum_{w\in[u,\tau_u\rangle} X_w \ge -\frac{2}{3}\cdot h\log(\Delta)\right)$$
$$\le \prod_{w\in[u,\tau_u\rangle}\frac{1-\varepsilon}{\sqrt{\Delta_w}}\cdot e^{\frac{3}{4}\cdot\frac{2}{3}h\log(\Delta)} = \frac{(1-\varepsilon)^{d(u,\tau_u)}}{\sqrt{\beta(\tau_u)}}\Delta^{\frac{h}{2}} \le (1-\varepsilon)^{d(u,\tau_u)}.$$

   As $d(u,\tau_u)\ge\log_\Delta\beta(\tau_u)\ge h$, this is at most $(1-\varepsilon)^h$.
2. If v is a nominee that is not in the same connected component of $T\setminus\{u\}$ as τ_u, then by Lemma 7, the probability that v is promising is

$$\mathbb{P}\left(\sum_{w\in[u,v\rangle} X_w \ge \frac{2}{3}\log\Delta\cdot h\right) \le \prod_{w\in[u,v\rangle}\frac{1-\varepsilon}{\sqrt{\Delta_w}}\cdot e^{-\frac{3}{4}\cdot\frac{2}{3}h\log\Delta}$$
$$= \frac{(1-\varepsilon)^{d(u,v)}}{\sqrt{\beta(v)}}\Delta^{-\frac{h}{2}} \le \frac{(1-\varepsilon)^{d(u,v)}}{\Delta^h}.$$

   However, denote by L the set of nominees in T_u. As they are a subset of the leaves of T_u, by the way θ is defined:

$$1 \ge \sum_{x\in L}\theta(v) \ge \sum_{x\in L}\frac{1}{\beta(v)} \ge \sum_{x\in L}\frac{1}{\Delta^{h+1}} = \frac{|L|}{\Delta^{h+1}} \tag{3}$$

   So, $|L|\le\Delta^{h+1}$. Therefore, by a union bound, the probability that there exists a nominee v that renders u misleading is at most $\Delta(1-\varepsilon)^h$.

The probability that u is misleading is then at most $(1+\Delta)(1-\varepsilon)^h$ as stated. In the case where the tree is regular, the analysis is the same, except that in (3), $\beta(v)=\Delta^h$, and so following the same logic, $|L|\le\Delta^h$, and this part contributes only $(1-\varepsilon)^h$.

For the second part of the lemma, consider some event X where u is not misleading. As τ_u is either the actual treasure or promising, and acts as the treasure in the eyes of $\texttt{A}_{walk}$, then the local procedure stops when it encounters τ_u. It might actually stop before (because it found another promising node), so, $\mathcal{Q}(\texttt{local}_h(u)\mid X)\le\mathcal{Q}(\texttt{A}_{walk}(T_h(u))\mid X)$. Therefore,

$$\mathbb{P}(X)\,\mathcal{Q}(\texttt{local}(u)\mid X) \le \mathbb{P}(X)\,\mathcal{Q}(\texttt{A}_{walk}(T_h(u))\mid X) \le \mathcal{Q}(\texttt{A}_{walk}(T_h(u))) = \mathcal{O}(\sqrt{\Delta}\cdot\texttt{depth}(T_h(u)))$$

But the depth of T_u is at most $\mathcal{O}(h \log \Delta)$, since its leaves satisfy $\beta(v) < \Delta^{h+1}$, and $\beta(v) \geq 2^{\mathtt{depth}(v)}$. For the case of a regular tree, $\beta(v) = \Delta^{\mathtt{depth}(v)}$ and so the depth of T_u is at most h, giving the result. ◀

E A More Involved $\mathcal{O}(\sqrt{\Delta} \log n \cdot \log\log n)$ Algorithm for Regular Trees

In this section we give a formal proof for the second item in Theorem 3. That is, we present algorithm $\mathtt{A}_{2-\mathrm{layers}}$ which is designed for (not necessarily complete) Δ-ary trees, and performs extremely well in the regime where $q < c/\sqrt{\Delta}$ for some small enough positive constant c. Specifically, in that regime, it finds the treasure in $\mathcal{O}(\sqrt{\Delta} \log n \cdot \log\log n)$ queries in expectation. Before we continue, let us note that taking a small enough c, the condition $q < c/\sqrt{\Delta}$ we are using here actually implies[5] Condition $(\star)$ with $\varepsilon = (1 - 2^{-1/4})/2$.

Algorithm $\mathtt{A}_{2-\mathrm{layers}}$ runs two algorithms in parallel, namely, $\mathtt{A}_{fast}$, and $\mathtt{A}_{mid}$. Algorithm $\mathtt{A}_{fast}$ is actually $\mathtt{A}_{sep}$, except that it applies the local procedure with parameter h being $h_2 = \lceil \kappa_2 \log\log n \rceil$ rather than $\Theta(\log n)$. Algorithm $\mathtt{A}_{mid}$ is similar to $\mathtt{A}_{sep}$, as it also uses h being $h_1 = \lceil \kappa_1 \log n \rceil$. However it uses a different local exploration procedure, see more details in Section E.1. κ_1 and κ_2 are constant independent of n whose value will be determined later. We will henceforth omit the ceiling $\lceil \cdot \rceil$ in the interest of readability.

Let us first recall some of the definitions that were introduced in Section 3.1. Since only regular trees are considered in this section, some of the definitions are simplified. Here $T_h(u)$ denotes the tree of nodes at distance at most h from u. Call a leaf $v \in T_h(u)$ a *nominee* if its distance to u is exactly h. Denote by $\mathcal{U}(u)$ the set of nominees that are not in the same component as τ_u in $T \setminus \{u\}$. Call a nominee *promising* if $\sum_{w \in [u,v\rangle} X_w \geq \frac{2}{3}h$, where $X_w = 1$ if the advice at w is pointing to v, $X_w = -1$ if it is pointing to u, and $X_w = 0$ otherwise. Note that X_u can never be -1. Let τ_u be the leaf on $T_h(u)$ closest to τ if $\tau \notin T_h(u)$ and $\tau_u = \tau$ otherwise. Recall also that u is called *h-misleading*, if one of the two following happens (1) $\tau_u \neq \tau$ and τ_u is not promising, or (2) There is some promising nominee in $\mathcal{U}(u)$.

Let $\mathtt{Excellent}$ be the event that no separator on the way to the treasure is h_2-misleading. The following claim is a direct consequence of Lemma 9 (regular tree case) and linearity of expectation, summing the query complexity of the $\lceil \log n \rceil$ separators on the way to the treasure.

▶ Claim 13.

$$\mathbb{P}(\mathtt{Excellent}) \cdot \mathcal{Q}(\mathtt{A}_{fast} \mid \mathtt{Excellent}) = \mathcal{O}\left(\sqrt{\Delta} \log n \cdot \log\log n\right).$$

To bound the total expected number of queries, we run in parallel algorithm $\mathtt{A}_{mid}$. All that remains is then to prove that $\mathbb{P}(\mathtt{Excellent}^c) \cdot \mathcal{Q}(\mathtt{A}_{mid} \mid \mathtt{Excellent}^c) = \mathcal{O}(\sqrt{\Delta} \log n)$.

E.1 Algorithm $\mathtt{A}_{mid}$

As mentioned, $\mathtt{A}_{mid}$ is similar to $\mathtt{A}_{sep}$ except that it uses a different local procedure. More precisely, recall that $\mathtt{A}_{sep}$ executes Procedure $\mathtt{local}_h(u)$ by running $\mathtt{A}_{walk}$ on $T_h(u)$ until it either finds the treasure or finds a promising nominee, and in the latter case, it declares that the treasure is on the connected component of $T \setminus \{u\}$ containing this nominee. In the context

[5] Indeed, recall that for regular trees, Condition $(\star)$ reads $q < \frac{1-\varepsilon-\Delta^{-1/4}}{\sqrt{\Delta}+\Delta^{1/4}}$. Now, $\Delta \geq 2$ implies that $1 - \Delta^{-1/4} \geq 1 - 2^{-1/4}$ and $\Delta^{1/4} \leq \sqrt{\Delta}$. Hence $\frac{1-\varepsilon-\Delta_v^{-1/4}}{\sqrt{\Delta}+\Delta^{1/4}} \geq \frac{1-2^{-1/4}-\varepsilon}{2}\frac{1}{\sqrt{\Delta}}$. We may set $\varepsilon = \frac{1-2^{-1/4}}{2}$ so that, as soon as $c < \frac{1-2^{-1/4}-\varepsilon}{2} = \frac{1-2^{-1/4}}{4}$, $q < c\Delta^{-1/2}$ implies Condition $(\star)$ with that choice of ε.

of Algorithm $\mathtt{A}_{mid}$, for technical commodity, we choose to run Procedure $\mathtt{local}_h(u)$ with a simpler exploration routine which we call $\mathtt{A}_{loop}$. It is less efficient than $\mathtt{A}_{walk}$ but its simplicity will be useful for analyzing its behaviour in various, "less clean", circumstances. Indeed, we will need to analyse the performances of $\mathtt{A}_{loop}$, conditioning on the event $\mathtt{Excellent}^{\,c}$, implying that some parts of the tree have to be pointing in the wrong direction.

The fact that $\mathtt{A}_{loop}$ is less efficient than $\mathtt{A}_{walk}$ will not affect the final bound, as its running time will dominate the total running time with very low probability.

Algorithm $\mathtt{A}_{loop}$. Recall in this section we only deal with Δ-regular trees. Define *level* i as the set of all nodes at distance i from the root. At each round, $\mathtt{A}_{loop}$ only compares nodes within a given level i. Specifically, it goes to the node in level i with most arrows pointing at it among the non-visited nodes in level i. Note that it considers only vertices whose parent has been explored already. The index i is incremented modulo the depth of the tree D, on every round. Below is a description in pseudocode. The loop over i explains the name $\mathtt{A}_{loop}$.

Algorithm 1: Algorithm $\mathtt{A}_{loop}$

1 Continuously loop over levels $1, 2, \ldots, D$
2 When considering level i, go to the yet unexplored reachable node at the current level (if one exists) that has most arrows pointing to it.

In what follows we will analyse Algorithm $\mathtt{A}_{loop}$ conditioning on some parts of the tree being misleading. For readability considerations, the interested reader might wish to first see how it behaves on a simpler scenario, without any conditioning. The proof of the following appears in Appendix F.2.

▶ **Lemma 14.** *Consider a (not necessarily complete) Δ-ary tree. Then $\mathcal{Q}(\mathtt{A}_{loop}) = \mathcal{O}(D^3\sqrt{\Delta})$.*

In fact, a slightly more refined analysis shows that $\mathcal{Q}(\mathtt{A}_{loop}) = \mathcal{O}(D^2\sqrt{\Delta})$, but this not needed for our current purposes, and so we omit it.

E.2 Analysis of $\mathtt{A}_{mid}$ Conditioning on $\mathtt{Excellent}^{\,c}$

To complete the proof of the second item in Theorem 3 we will show that if c small enough, then

$$\mathbb{P}(\mathtt{Excellent}^{\,c}) \cdot \mathcal{Q}\left(\mathtt{A}_{mid} \mid \mathtt{Excellent}^{\,c}\right) = \mathcal{O}(\sqrt{\Delta}\log n).$$

E.2.1 Decomposing $\mathtt{Excellent}^{\,c}$

At a high level, we seek to break $\mathtt{Excellent}^{\,c}$ into many elementary bad events. Denote $u_1, \ldots u_\ell$ the sequence of separators on the way to the treasure τ. Note that $\ell \leq \lceil \log n \rceil$. First,

$$\mathtt{Excellent}^{\,c} = \bigcup_{i \leq \ell} \{u_i \text{ is } h_2\text{-misleading}\}.$$

Using the union bound argument in Section F (Claim 21),

$$\mathcal{Q}\left(\mathtt{A}_{mid} \bigcap \mathtt{Excellent}^{\,c}\right) \leq \sum_{i \leq \ell} \mathcal{Q}\left(\mathtt{A}_{mid} \bigcap u_i \text{ is } h_2\text{-misleading}\right), \tag{4}$$

where, to keep the equation light we write $\mathcal{Q}(\mathtt{A} \bigcap E)$ in place of $\mathcal{Q}(\mathtt{A} \mid E) \cdot \mathbb{P}(E)$ where $\mathtt{A}$ is an algorithm and E is an event.

Since we ultimately want to show that the left hand side in the previous equation is $\mathcal{O}(\sqrt{\Delta}\log n)$, it is sufficient to show that for any fixed $i \leq \ell$,

$$\mathcal{Q}\left(\mathtt{A}_{mid} \bigcap u_i \text{ is } h_2\text{-misleading}\right) = \mathcal{O}(\sqrt{\Delta}). \tag{5}$$

From now on, we fix i and focus on the case where u_i is h_2-misleading. Recall that algorithm $\mathtt{A}_{mid}$, just as $\mathtt{A}_{sep}$, proceeds in phases of local exploration, running also an exhaustive search in parallel to handle the case that one of the local explorations ends with a wrong answer. Denote by $\mathtt{Good}$ the event that all separators on the way to the treasure, namely, $u_1, \ldots, u_\ell$, are not h_1-misleading. Under $\mathtt{Good}$, the local exploration phases amount to running $\mathtt{A}_{loop}$ on $T_{h_1}(u_j)$ for $j \leq \ell$. Now,

$$\begin{aligned}\mathcal{Q}\left(\mathtt{A}_{mid} \bigcap u_i \text{ is } h_2\text{-misleading}\right) &= \mathcal{Q}\left(\mathtt{A}_{mid} \bigcap (u_i \text{ is } h_2\text{-misleading} \cap \mathtt{Good})\right) \\ &\quad + \mathcal{Q}\left(\mathtt{A}_{mid} \bigcap (u_i \text{ is } h_2\text{-misleading} \cap \neg\mathtt{Good})\right).\end{aligned}$$

By Lemma 9 (regular tree case),

$$\mathbb{P}(\neg\mathtt{Good}) \leq 2(1-\varepsilon)^{h_1} = 2(1-\varepsilon)^{\kappa_1 \log n}$$

Recall that Condition $(\star)$ is satisfied with the constant $\varepsilon = (1-2^{-1/4})/2$, and so taking κ_1 to be a large enough constant, gives that $\mathbb{P}(\neg\mathtt{Good}) > 1/n$. This means that if $\mathtt{Good}$ does not hold, it is fine to resort to exhaustive search, as the second term above becomes $\mathcal{O}(1)$. Also, since $\mathtt{A}_{mid}$ runs $\mathtt{A}_{loop}$ on local subtrees until it finds a promising nominee, and conditioned on $\mathtt{Good}$, the local "treasure" is such a promising nominee, then the number of queries made by such a local run is bounded above by number of queries $\mathtt{A}_{loop}$ needs to find the treasure there. So by linearity of expectation,

$$\begin{aligned}&\mathcal{Q}\left(\mathtt{A}_{mid} \bigcap u_i \text{ is } h_2\text{-misleading}\right) \\ &\leq \sum_{j \leq \log n} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(u_j)) \bigcap (u_i \text{ is } h_2\text{-misleading} \cap \mathtt{Good})\right) + \mathcal{O}(1) \\ &\leq \sum_{j \leq \log n} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(u_j)) \bigcap u_i \text{ is } h_2\text{-misleading}\right) + \mathcal{O}(1).\end{aligned}$$

The last inequality follows from the fact that for any algorithm A and any two events $E_1 \subseteq E_2$, $\mathcal{Q}(A \bigcap E_1) \leq \mathcal{Q}(A \bigcap E_2)$.

For the sake of lightening notations, we henceforth refer to u_j as σ' and u_i as u. This choice of notations reflects the fact that we are rooting the tree at $u_j = \sigma'$ and running $\mathtt{A}_{loop}$ on $T_{h_1}(\sigma')$. The fact that σ' and u are separators is not relevant in this analysis. We also denote by τ_u the leaf on $T_{h_2}(u)$ that is closest to τ and by τ' the leaf of $T_{h_1}(u)$ that is closest to τ or simply τ if $\tau \in T_{h_1}(u)$. With these notations Equation (5) immediately follows once we prove:

▶ **Lemma 15.** *For any $\sigma', u \in T$,*

$$\mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap u \textit{ is } h_2\textit{-misleading}\right) = \mathcal{O}\left(\frac{\sqrt{\Delta}}{\log n}\right).$$

E.2.2 Decomposing the Event $\{u$ is h_2-misleading$\}$

So far we saw that it is sufficient to analyse the events where one separator is h_2-misleading. We now pursue decomposing these events into even smaller ones. To this aim the following definition is convenient.

▶ **Definition 16.** Let $a, b \in T$ be two nodes such that a is the closest one to τ out of the nodes in $[a, b]$. Noting that a vertex can never point to itself:

- For $S \subseteq \langle a, b]$, denote by $M^S_{\text{sides}}(a, b)$ the event that the nodes of S neither point towards a nor towards b.
- For $S \subseteq [a, b\rangle$, denote by $M^S_{\text{up}}(a, b)$ the event that the nodes of S all point towards b.

▶ Claim 17. For any a, b and S as in Definition 16,

- $\mathbb{P}\left(M^S_{\text{sides}}(a, b)\right) \leq q^{|S|}$,
- $\mathbb{P}\left(M^S_{\text{up}}(a, b)\right) \leq \left(\frac{q}{\Delta}\right)^{|S|}$.

Let us now see in more detail what it means for a node u to be h_2-misleading. First recall from the definition that $|[u, \tau_u]| = h_2$, as otherwise $\tau \in T_{h_2}(u)$ and u cannot be h_2-misleading because of the path $[u, \tau_u]$. Several cases need to be considered.

1. τ_u is not promising, and so the sum of advice on $[u, \tau_u\rangle$ is strictly less than $\frac{2}{3}h_2$. In this case, at least one of the following two must be true:
   a. There are $\frac{1}{6}h_2$ locations on the path $[u, \tau_u\rangle$ where the advice points outside of the path (the value of the corresponding X_i's is 0). This corresponds to $M^S_{\text{sides}}(\tau_u, u)$ for some set $S \subseteq [u, \tau_u\rangle$ of size[6] $|S| = \frac{1}{6}h_2$.
   b. There are $\frac{1}{12}h_2$ locations on $\langle u, \tau_u\rangle$ that point towards u (the value of the corresponding X_i's is 1). This corresponds to $M^S_{\text{up}}(\tau_u, u)$ for some set $S \subseteq [u, \tau_u]$ of size $|S| = \frac{1}{12}h_2$.
2. Some $v \in \mathcal{U}(u)$ is promising. In this case there must be some $\frac{2}{3}h_2$ locations on $[u, v]$ that point towards v. This corresponds to $M^S_{\text{up}}(u, v)$ for some $S \subseteq M^S_{\text{up}}([v, u])$ of size $|S| = \frac{2}{3}h_2$.

Define $\mathcal{C}(u) = \{S \subseteq [u, \tau_u] \mid |S| = \frac{1}{6}h_2\}$ and $\mathcal{D}(u) = \{S \subseteq [u, \tau_u] \mid |S| = \frac{1}{12}h_2\}$. Similarly define $\mathcal{E}(u) = \{(v, S) \mid v \in \mathcal{U}(u),\ S \subseteq [u, v],\ \text{and } |S| = \frac{2}{3}h_2\}$. Combining Definition 16 with the previous paragraph, yields

$$\{u \text{ is } h_2\text{-misleading}\} \subseteq \{\tau_u \text{ is not promising}\} \cup \bigcup_{v \in \mathcal{U}(u)} \{v \text{ is promising}\}$$

$$\subseteq \bigcup_{S \in \mathcal{C}(u)} M^S_{\text{sides}}(\tau_u, u) \bigcup_{S \in \mathcal{D}(u)} M^S_{\text{up}}(\tau_u, u) \bigcup_{(v,S) \in \mathcal{E}(u)} M^S_{\text{up}}(u, v).$$

In fact, $\mathcal{E}(u)$ needs to be further decomposed. For each $v \in \mathcal{E}(u)$, let $k(v) = |[u, v] \cap [\sigma', \tau']|$. For each non-negative integer $k \geq 0$, let

$$\mathcal{E}_k(u) = \{(v, S) \in \mathcal{E}(u) \mid k(v) = k\}.$$

Clearly, $\mathcal{E}(u) = \cup_{k=0}^{h_2} \mathcal{E}_k(u)$.

[6] Here again we omit the $\lceil \cdot \rceil$.

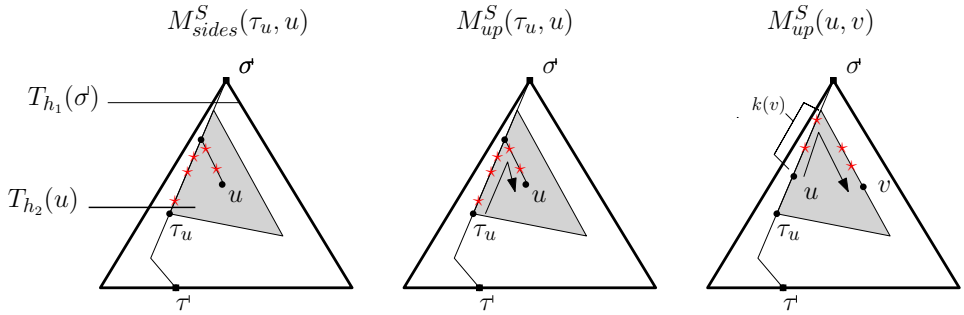


Figure 1 Different relative positions of u, τ_u and σ'. The path $[u, \tau_u]$ and different mistake patterns. In the left one mistakes (depicted as red stars) point outside of $[u, \tau_u]$, in the second they point towards u and in the third towards a nominee of $T_{h_2}(u)$, $v \in \mathcal{U}(u)$.

Using the union bound (Claim 21) as in Equation 4, the aforementioned decomposition implies:

$$\begin{aligned}\mathcal{Q}\left(\mathtt{A}_{loop}\left(T_{h_1}(\sigma')\right) \bigcap u \text{ is } h_2\text{-misleading}\right) \leq & \sum_{S \in \mathcal{C}(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{sides}}(\tau_u, u)\right) \\ & + \sum_{S \in \mathcal{D}(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{up}}(\tau_u, u)\right) \\ & + \sum_{k=0}^{h_2} \sum_{(v,S) \in \mathcal{E}_k(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{up}}(u, v)\right) \end{aligned} \tag{6}$$

To prove Lemma 15, our goal will be to show that each sum in the above equation is at most $\mathcal{O}(\sqrt{\Delta}/\log n)$.

E.3 Analysing Atomic Expressions

To prove that each sum is indeed $\mathcal{O}(\sqrt{\Delta}/\log n)$ we use the following two lemmas (proved in Appendix E.4), which encapsulate the core of this proof, namely, the resilience of $\mathtt{A}_{loop}$ to certain kinds of error patterns.

▶ **Lemma 18.** *Consider a tree T rooted at σ with treasure located at τ. Let $a, b \in T$ be two nodes such that a is the closest one to τ out of the nodes in $[a, b]$. Then,*

$$\mathcal{Q}\left(\mathtt{A}_{loop} \mid M^S_{sides}(a, b)\right) = \mathcal{O}\left(D^4 \Delta^{\frac{|S|+1}{2}}\right).$$

▶ **Lemma 19.** *Consider a tree T rooted at σ with treasure located at τ. Let $a, b \in T$ be two nodes such that a is the closest one to τ out of the nodes in $[a, b]$. Then,*

$$\mathcal{Q}\left(\mathtt{A}_{loop} \mid M^S_{up}(a, b)\right) = \mathcal{O}\left(D^4 \Delta^{K+\frac{1}{2}} 4^{|S|}\right),$$

where $K = |S \cap [\sigma, \tau]|$.

As a first step to bounding the three sums of Equation (6), note that:

$$|\mathcal{C}(u)| \leq 2^{h_2} \tag{7}$$

$$|\mathcal{D}(u)| \leq 2^{h_2}, \tag{8}$$

$$|\mathcal{E}_k(u)| \leq 2^{h_2} \Delta^{h_2 - k}. \tag{9}$$

Indeed, $\mathcal{C}(u), \mathcal{D}(u)$ are sets of subsets of a path of length h_2. For the last term, the number of $v \in \mathcal{U}(u)$ at distance h_2 from u for which $k(v) = k$ is bounded by Δ^{h_2-k}. Now the three sums:

1. $S \in \mathcal{C}(u)$, so $S \subseteq [u, \tau_u]$ and $|S| = \frac{1}{6}h_2$, and τ_u is the closest to τ of all the nodes on the path. By Lemma 18,

$$\mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \mid M^S_{\text{sides}}(\tau_u, u)\right) = \mathcal{O}\left(h_1^4 \Delta^{\frac{|S|+1}{2}}\right).$$

   According to Claim 17,

$$\mathbb{P}(M^S_{\text{sides}}(\tau_u, u)) \leq q^{|S|}.$$

   Combining these bounds and (7) yields

$$\begin{aligned}\sum_{S \in \mathcal{C}(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{sides}}(\tau_u, u)\right) &= \mathcal{O}\left(2^{h_2} \cdot q^{|S|} \cdot h_1^4 \Delta^{\frac{|S|+1}{2}}\right) \\ &= \mathcal{O}\left(\sqrt{\Delta} \cdot 2^{h_2} \cdot c^{|S|} \cdot h_1^4\right),\end{aligned}$$

   because $q < c/\sqrt{\Delta}$. Recall that $h_1 = \kappa_1 \log n$, $h_2 = \kappa_2 \log\log n$, and $|S| = \frac{1}{6}h_2$. κ_1 was already set to be some constant. Taking a large enough κ_2 and a small enough c, both independent of n, the previous expression is $\mathcal{O}(\sqrt{\Delta}/\log n)$ as needed.
2. $S \in \mathcal{D}(u)$, so $S \subseteq [u, \tau_u]$ and $|S| = \frac{1}{12}h_2$. Therefore, by Lemma 19,

$$\mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \mid M^S_{\text{up}}(\tau_u, u)\right) = \mathcal{O}\left(h_1^4 \Delta^{|S|+\frac{1}{2}} 2^{h_2}\right).$$

   Because $K \leq |S|$ and $4^{|S|} \leq 2^{h_2}$. Combined with Claim 17 and (8):

$$\begin{aligned}\sum_{S \in \mathcal{D}(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{up}}(\tau_u, u)\right) &= \mathcal{O}\left(2^{h_2} \cdot \left(\frac{q}{\Delta}\right)^{|S|} \cdot h_1^4 \Delta^{|S|+\frac{1}{2}} 2^{h_2}\right) \\ &= \mathcal{O}\left(\sqrt{\Delta} \cdot 4^{h_2} \cdot q^{|S|} h_1^4\right)\end{aligned}$$

   Again, since $|S| = \frac{1}{12}h_2$, then c and κ_2 can be chosen so that this is $\mathcal{O}(\sqrt{\Delta}/\log n)$.
3. $(v, S) \in \mathcal{E}_k(u)$, where $v \in \mathcal{U}(u)$, $S \subseteq [u, v]$, and $|S| = \frac{2}{3}h_2$. Also, $|[u, v] \cap [\sigma', \tau']| = k$, and so $|S \cap [\sigma', \tau']| \leq k$. As $v \in \mathcal{U}(u)$, then u is the closest to treasure of the vertices on $[u, v]$. By Lemma 19,

$$\mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \mid M^S_{\text{up}}(u, v)\right) = \mathcal{O}\left(h_1^4 \Delta^{k+\frac{1}{2}} 4^{h_2}\right)$$

   Combined with (9) and Claim 17:

$$\begin{aligned}\sum_{k=0}^{h_2} \sum_{(v,S) \in \mathcal{E}_k(u)} \mathcal{Q}\left(\mathtt{A}_{loop}(T_{h_1}(\sigma')) \bigcap M^S_{\text{up}}(u, v)\right) &= \mathcal{O}\left(\sum_{k \leq h_2} 2^{h_2} \Delta^{h_2-k} \cdot \left(\frac{q}{\Delta}\right)^{\frac{2}{3}h_2} h_1^4 \cdot \Delta^{k+\frac{1}{2}} 4^{h_2}\right) \\ &= \mathcal{O}\left(\sqrt{\Delta} \cdot h_2 8^{h_2} h_1^4 \left(q^2\Delta\right)^{\frac{1}{3}h_2}\right). \\ &= \mathcal{O}\left(\sqrt{\Delta} \cdot h_2 8^{h_2} h_1^4 \cdot c^{\frac{1}{3}h_2}\right).\end{aligned}$$

   Similarly to the two previous sums, this whole expression can be made as small as $\mathcal{O}(\sqrt{\Delta}/\log n)$.

Note that we assumed for simplicity that u, τ_u and v are all inside $T_{h_1}(\sigma')$. If they are not, we take nodes that are the closest to them on this subtree, which can only improve the bounds.

This concludes the proof of Lemma 15 and hence completes the proof of the second item in Theorem 3.

E.4 The Lemmas About the Resilience of $\mathtt{A}_{loop}$

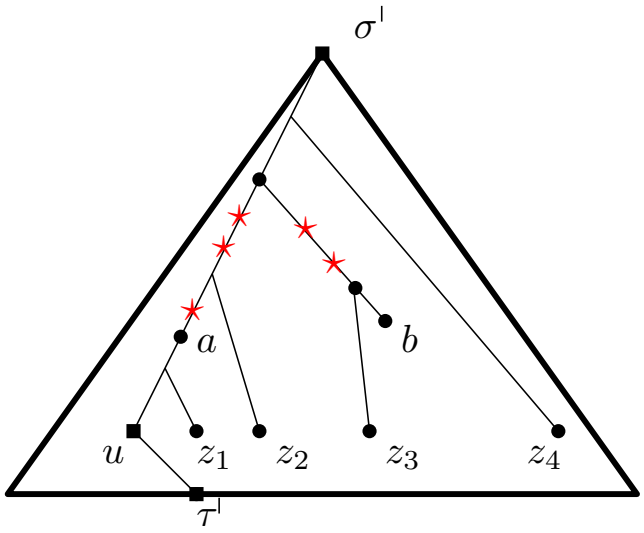


Figure 2 Notations introduced in the proof of Lemma 18 and 19. Points of S are depicted in red. On the figure $n(z_1) = 0, m(z_1) = 0$, $n(z_2) = 1, m(z_2) = 0$, $n(z_3) = 3, m(z_3) = 2$ and $n(z_4) = 3, m(z_4) = 0$.

▶ **Lemma 18** ***(restated).*** *Consider a tree T rooted at σ with treasure located at τ. Let $a, b \in T$ be two nodes such that a is the closest one to τ out of the nodes in $[a, b]$. Then,*

$$\mathcal{Q}\left(\mathtt{A}_{loop} \mid M^S_{sides}(a, b)\right) = \mathcal{O}\left(D^4 \Delta^{\frac{|S|+1}{2}}\right).$$

Proof. As in the proof of Lemma 14, we break the number of queries made by $\mathtt{A}_{loop}$ conditioning on $M^S_{\text{sides}}(a, b)$ into a sum of random variables Q_j which correspond to the number of queries needed to discover the j-th node on the path $[\sigma, \tau]$ once the $(j-1)$-th was discovered. Each Q_j is bounded above by D times the expected number of competitors who beat this node. This is because each phase takes D steps, and only a subset of these nodes will actually be checked by $\mathtt{A}_{loop}$ on layer j before trying the correct node. Hence, a bound on the number of competitors who beat a given $u \in [\sigma, \tau]$ translates to a bound on $\mathcal{Q}\left(\mathtt{A}_{loop}(T) \mid M^S_{\text{sides}}(\tau, \sigma)\right)$ by multiplying it by D^2.

Let u be such a node, and z be a competitor of u (i.e., it is at the same level as u). Define $k(z)$ as half the distance between z and u, and denote $n(z) := |S \cap [\sigma, \tau] \cap [u, z]|$ and $m(z) := |S \cap [\sigma, \tau]^c \cap [u, z]|$. See Figure 2 for illustraion.

First note, that since all advice of $S \subseteq [a, b]$ points sideways w.r.t. to this path, then any of it which is on the path $[u, z]$ also points sideways w.r.t. it, except possibly at one point, which may actually point towards z. The different cases are seen in Figure 2:

- For z_1, the paths do not intersect at all.
- In the case of z_2, if the least common ancestor of u and z_2 was a member of S, then it could point towards z_2, and that would be sideways w.r.t. $[a, b]$.
- For z_3, the least common ancestor of b and z_3 could point towards z_3.
- For z_4, the least common ancestor of a and b could point towards z_4.
- There is also the case where $a \notin [\sigma, \tau]$, which is not depicted on Figure 2. The analysis remains valid, and in fact $n(z) = 0$ for all competitors z.

This one special vertex, if it exists, conditioned on that it points sideways w.r.t. $[a, b]$, points towards z with probabilty $1/(\Delta - 2)$, and otherwise points sideways w.r.t. $[u, z]$.

Fix k, n and m, and consider a competitor z such that $k(z) = k$, $n(z) = n$, and $m(z) = m$. On the path $[u, z]$ the number of advice remaining to be sampled is $2k - n - m - 1$. By Lemma 20:

$$\begin{aligned}\mathbb{P}(z \text{ beats } u) &\le \left(1 - \frac{1}{\Delta - 2}\right)\mathbb{P}\left(\sum_{s=1}^{2k-1-n-m} X_s \ge 0\right) + \frac{1}{\Delta - 2}\mathbb{P}\left(\sum_{s=1}^{2k-1-n-m} X_s \ge -1\right) \\ &= \left(\frac{1}{\sqrt{\Delta}}\right)^{2k-1-n-m} + \frac{4}{\Delta - 2}\left(\frac{1}{\sqrt{\Delta}}\right)^{2k-2-n-m} \\ &= \left(1 + \frac{4\sqrt{\Delta}}{\Delta - 2}\right)\left(\frac{1}{\sqrt{\Delta}}\right)^{2k-1-n-m} \le 7 \cdot \left(\frac{1}{\sqrt{\Delta}}\right)^{2k-1-n-m},\end{aligned}$$

as $\Delta \ge 3$. For fixed k, n, m there are at most Δ^{k-m} nodes z with $k(z) = k$ and $m(z) = m$. Also, for each such node, $n + m \le 2k$. Hence, the total expected number of competitors that beat u is at most:

$$\sum_{k \le D, n+m \le 2k} \Delta^{k-m} \cdot 7 \left(\frac{1}{\sqrt{\Delta}}\right)^{2k-1-m-n}$$

For each choice of k there is exactly one corresponding value of n. This n satisfies $n \le |S|$. There are also at most D choices for m. Thus, the above is at most

$$7 \cdot \sum_{k \le D, n+m \le 2k} \Delta^{(n+1-m)/2} = \mathcal{O}\left(D^2 \Delta^{(|S|+1)/2}\right).$$

◀

▶ **Lemma 19** ***(restated).*** *Consider a tree T rooted at σ with treasure located at τ. Let $a, b \in T$ be two nodes such that a is the closest one to τ out of the nodes in $[a, b]$. Then,*

$$\mathcal{Q}\left(\mathtt{A}_{loop} \mid M^S_{up}(a, b)\right) = \mathcal{O}\left(D^4 \Delta^{K+\frac{1}{2}} 4^{|S|}\right),$$

where $K = |S \cap [\sigma, \tau]|$.

Proof. Let u be a node on the path $[\sigma, \tau]$. Our aim is to show that the expected number of competitors of u that beat it is $\mathcal{O}(D^2 \Delta^{K+\frac{1}{2}} 4^{|S|})$.

As in the proof of Lemma 18, let z be a competitor of u. Define $k(z)$ as half the distance between z and u, namely $k(z) := d(z, u)/2$. Denote $n(z) := |S \cap [\sigma, \tau] \cap [u, z]|$, and $m(z) := |S \cap [\sigma, \tau]^c \cap [u, z]|$.

Fixing k, n and m, take a competitor z such that $k(z) = k$, $n(z) = n$, and $k(z) = k$. The probability that such a z beats u is

$$\mathbb{P}\left(\sum_{s=1}^{2k-1-n-m} X_s \ge -n - m\right) \le 4^{n+m} \Delta^{n+m-k+\frac{1}{2}},$$

by Lemma 20. There are at most Δ^{k-m} such nodes z. We bound the probability that each of these nodes z beats the treasure using the trivial bound 1 or the one above, depending on whether $n + m \le k$ or $n + m > k$. Hence the total expected number of competitors of u who beat it is at most

$$\sum_{k \le D, n+m \le k} \Delta^{k-m} \cdot 4^{n+m} \Delta^{n+m-k+\frac{1}{2}} + \sum_{k \le D, n+m > k} \Delta^{k-m}.$$

Since $n+m \leq |S|$, and $n \leq K$, the first term is at most:

$$4^{|S|} \sum_{k \leq D, n+m \leq k} \Delta^{K+\frac{1}{2}} \leq 4^{|S|} \cdot D^2 \cdot \Delta^{K+\frac{1}{2}},$$

where we used the fact that there are most D distinct values for k and D distinct values for m, while there is only one choice of n for each k. As for the second term, since $n+m > k$, then it is at most:

$$\sum_{k \leq D, n+m > k} \Delta^n \leq \sum_{k \leq D, n+m > k} \Delta^K \leq \sum_{k,m \leq D} \Delta^K \leq D^2 \cdot \Delta^K,$$

concluding the proof. ◀

F Complementary Proofs

F.1 Another Large Deviation Estimate

Here, we introduce another large deviation estimate used for the analysis of the query algorithm for regular trees. It gives better results for large h, yet works only for identical random variables, and so suits regular trees, unlike Lemma 7.

▶ **Lemma 20.** *Consider random variables X_i taking values $\{-1,0,1\}$ with respective probabilities $(1-q, q\left(1-\frac{2}{\Delta}\right), \frac{q}{\Delta})$. If $q < \frac{c}{\sqrt{\Delta}}$ where $c < 1/64$, then for all $0 \leq h \leq l$,*

$$\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \geq -h\right) \leq (4\sqrt{\Delta})^h \Delta^{-\ell/2}$$

Proof. Assume $\sum_{i=1}^{\ell} X_i \geq -h$. Denote by $j := \{i \mid X_i = 1\}$. As the number of -1's is at least h, then $j \leq (\ell - h)/2$. There must also be at least $\ell - h - 2j$ zeros amongst what remains, otherwise the sum is less than $-h$. Using a union bound over the value of j and the locations of the ones and zeros we get:

$$\begin{aligned}
\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \geq -h\right) &\leq \sum_{j=0}^{\frac{\ell-h}{2}} \binom{\ell}{j}\binom{\ell-j}{\ell-h-2j}\left(q(1-\frac{2}{\Delta})\right)^{\ell-h-2j}\left(\frac{q}{\Delta}\right)^j \\
&\leq \sum_{j=0}^{\frac{\ell-h}{2}} \binom{\ell}{j}\binom{\ell-j}{\ell-h-2j} q^{\ell-h-2j}\left(\frac{q}{\Delta}\right)^j \\
&\leq 3^\ell \sum_{j=0}^{\frac{\ell-h}{2}} q^{\ell-h-2j}\left(\frac{q}{\Delta}\right)^j \leq 3^\ell \cdot \frac{\ell}{2}\left(q^{\ell-h} + \left(\frac{q}{\Delta}\right)^{\frac{\ell-h}{2}}\right).
\end{aligned}$$

The last step uses the bound $\sum_{j=1}^{N} \rho^k \leq N \cdot (\rho + \rho^N)$. Note that $x/2 < (4/3)^x$ always, and assigning $q < c/\sqrt{\Delta}$, this is at most:

$$4^\ell \frac{1}{\sqrt{\Delta}^{\ell-h}}\left(c^{\ell-h} + \left(\frac{\sqrt{c}}{\Delta^{3/2}}\right)^{\ell-h}\right) \leq 4^\ell \frac{1}{\sqrt{\Delta}^{\ell-h}}\left(c^{\ell-h} + \sqrt{c}^{\ell-h}\right) \leq 4^\ell \left(\frac{2\sqrt{c}}{\sqrt{\Delta}}\right)^{\ell-h}.$$

Since $c < 1/64$, then $2\sqrt{c} \leq 1/4$ which means that this is at most

$$4^h \left(\frac{1}{\sqrt{\Delta}}\right)^{\ell-h},$$

giving the desired bound. ◀

F.2 Algorithm $\mathtt{A}_{loop}$ without Conditioning

▶ **Lemma 14** ***(restated).*** *Consider a (not necessarily complete) Δ-ary tree. Then $\mathcal{Q}(\mathtt{A}_{loop}) = \mathcal{O}(D^3\sqrt{\Delta})$.*

Proof. Denote by $N_{layer}(u)$ the number of nodes on the same depth as u which have more discovered arrows than u pointing to them. This definition is central because of the following observation. The number of moves needed before finding u_{i+1} once u_i has been found is less than $\mathcal{O}(DN_{layer}(u_i))$. Indeed, once u_i is discovered, only *a subset of* the nodes which have more arrows pointing to them than u_{i+1} on layer $i+1$ are tried before u_{i+1} (at step (2) in the pseudocode description). The loop over the levels (at step (1)) induces a multiplicative factor of $\mathcal{O}(D)$.

Using linearity of expectation, it only remains to estimate $\mathbb{E}\left(N_{layer}(u_i)\right)$ where u_i is the ancestor of the treasure at depth $i \leq d$. There are at most Δ^ℓ nodes on layer i at distance $2\ell - 1$ from u_i, for any $1 \leq \ell \leq i$. Moreover the probability that each of these nodes has more arrows pointing towards it than u_i exactly corresponds to $\mathbb{P}\left(\sum_{j=1}^{2\ell-1} X_j \geq 0\right)$, with the notations of Lemma 20.

Indeed, when comparing the amount of advice pointing to two different nodes u and v, only the nodes of $\langle u, v\rangle$ matter.

When estimating the probability that v beats u, each random variable X_j has to be interpreted as taking value $+1$ if the advice points towards v, -1 if it points towards u, and 0 if it points neither to u nor v. In the case that $u = u_j$ and v is another node on layer j, these events happen respectively with probability q/Δ, $1 - q + q/\Delta$ and $q(1 - 2\frac{1}{\Delta})$.

This means that for each i,

$$\mathbb{E}\left(N_{layer}(u_i)\right) \leq \sum_{\ell=1}^{i} \mathbb{P}\left(\sum_{j=1}^{2\ell-1} X_j \geq 0\right) \Delta^\ell \leq \sum_{\ell=1}^{d} \mathbb{P}\left(\sum_{j=1}^{2\ell-1} X_j \geq 0\right) \Delta^\ell.$$

By Lemma 20 this is at most

$$\mathcal{O}\left(\sum_{\ell=1}^{d} \Delta^{-\ell+\frac{1}{2}} \cdot \Delta^\ell\right) = \mathcal{O}(D\sqrt{\Delta}) = \mathcal{O}\left(D\sqrt{\Delta}\right).$$

◀

F.3 Special Form of Union Bound

▶ Claim 21. Let A be an event that can be decomposed as the union of events $(A_i)_{i\in I}$, $A \subseteq \bigcup_{i\in I} A_i$. Let X be a random variable.

$$\mathbb{E}(X \mid A)\mathbb{P}(A) \leq \sum_i \mathbb{E}(X \mid A_i)\mathbb{P}(A_i)$$

Proof. We denote by $\chi(B)$ the indicator function of event B. Then

$$\begin{aligned}\mathbb{E}(X \mid A)\mathbb{P}(A) = \mathbb{E}(X \cdot \chi(A)) &\leq \mathbb{E}\left(X \cdot \chi\left(\bigcup_i A_i\right)\right) \leq \mathbb{E}\left(X \cdot \sum_i \chi(A_i)\right) \\ &= \sum_i \mathbb{E}\left(X \cdot \chi(A_i)\right) = \sum_i \mathbb{E}(X \mid A_i)\mathbb{P}(A_i).\end{aligned}$$

Where we used the union bound in the form $\chi(\bigcup_i A_i) \leq \sum_i \chi A_i$ and then linearity of expectation. ◀

G Lower bounds

G.1 An Exponential Lower Bound Above the Threshold: Proof of Lemma 10

For the lower bound, assume the algorithm is given the advice $\mathtt{adv}$ for all the internal nodes for free. By Yao's principle, instead of taking the worst case placement of the treasure for a randomized algorithm, we obtain a lower bound by considering only deterministic algorithms when the treasure is placed uniformly at random at one of the leaves.

In this simplified setting, an optimal algorithm can be described explicitly: It sorts the leaves according $\mathbb{P}(\cdot \,|\, \mathtt{adv})$ (Claim 22) and tries them in this order. This order in fact corresponds to ranking nodes by how many arrows point to them (Claim 23). The expected number of nodes which are higher than the treasure in this ordering is therefore a lower bound for this algorithm, and thus for all algorithms.

Let $\mathcal{L}$ be the set of leaves. For a given leaf $u \in \mathcal{L}$ and an advice configuration $\mathtt{adv}$, let $C(\mathtt{A}, \mathtt{adv}, u)$ be the cost (number of queries) of $\mathtt{A}$ when the advice is equal to $\mathtt{adv}$ and the treasure is located at u. We also define the cost $C(\mathtt{A}, u)$ of an algorithm $\mathtt{A}$ when the treasure τ is located at u to be the expected cost of $\mathtt{A}$ before finding τ where the expectation is over advice setting. That is:

$$C(\mathtt{A}, u) = \sum_{\mathtt{adv}} C(\mathtt{A}, \mathtt{adv}, u)\mathbb{P}(\mathtt{adv} \,|\, u)\,.$$

In our setting, the expected number of queries of $\mathtt{A}$ is:

$$C(\mathtt{A}) = \sum_{u \in \mathcal{L}} \mathbb{P}(u) \sum_{\mathtt{adv}} C(\mathtt{A}, \mathtt{adv}, u)\mathbb{P}(\mathtt{adv} \,|\, u)\,.$$

▶ Claim 22. The algorithm $\mathtt{A}$ that tries the locations u in the order given by $\mathbb{P}(u \,|\, \mathtt{adv})$, i.e., the most likely u is tried first and the least likely tried last, minimizes $C(\mathtt{A})$.

Proof. We can write

$$C(\mathtt{A}) = \sum_{\mathtt{adv}} \mathbb{P}(\mathtt{adv}) \sum_{u \in \mathcal{L}} C(\mathtt{A}, \mathtt{adv}, u)\mathbb{P}(u \,|\, \mathtt{adv})\,,$$

where it is understood that $\mathbb{P}(\mathtt{adv})$ is the marginal of $\mathbb{P}(\mathtt{adv}, u)$ with respect to the advice. Note that the term $\mathbb{P}(u \,|\, \mathtt{adv})$, standing for the probability of u holding the treasure given that the advice configuration is $\mathtt{adv}$, is only defined because we assume the treasure is placed according to a known distribution (uniform in our case). For a fixed advice setting $\mathtt{adv}$, it follows from the *rearrangement inequality* that $\sum_{u \in \mathcal{L}} C(\mathtt{A}, \mathtt{adv}, u)\mathbb{P}(u \,|\, \mathtt{adv})$ is minimized when $C(\mathtt{A}, \mathtt{adv}, u)$ and $\mathbb{P}(u \,|\, \mathtt{adv})$ are sorted in the same order with respect to u. This corresponds to algorithm $\mathtt{A}$ trying the locations u in the order given by $\mathbb{P}(u \,|\, \mathtt{adv})$, which is exactly the statement of the claim. Hence, since we assume that all advice is known, the algorithm we have just described is feasible, and, in fact, optimal. Moreover, its query complexity is at least 1 plus the expected number of nodes which are strictly more likely than the treasure, where the expectation is taken over the randomness of the advice. ◀

It only remains to check that a node u is more likely than τ given an advice setting $\mathtt{adv}$ iff more arrows point to u than τ. This will conclude the proof of Lemma 10 and hence of the exponential lower bound in Theorem 4.

▶ Claim 23. For two leaves $u, v \in \mathcal{L}$, and advice configuration $\mathtt{adv}$, $\mathbb{P}(u \,|\, \mathtt{adv}) > \mathbb{P}(v \,|\, \mathtt{adv})$ if and only if there is more advice pointing towards u than advice pointing towards v.

Proof. Recall that, by definition of the model

$$\mathbb{P}\left(\mathtt{adv} \,|\, \tau = u\right) = \left(p + \frac{q}{\Delta}\right)^{|\overrightarrow{\mathtt{adv}}(u)|} \left(q(1 - \frac{1}{\Delta})\right)^{|\overleftarrow{\mathtt{adv}}(u)|},$$

In our regime it will always be the case that $p + \frac{q}{\Delta} > q(1 - \frac{1}{\Delta})$, simply because we assume $q < p$. Hence $\mathbb{P}\left(\mathtt{adv} \,|\, \tau = u\right)$ is an increasing function of $|\overrightarrow{\mathtt{adv}}(u)|$.

Since τ is placed uniformly at random, it follows from Bayes rule that $\mathbb{P}\left(\mathtt{adv} \,|\, \tau = u\right) \propto \mathbb{P}\left(\tau = u \,|\, \mathtt{adv}\right)$. The symbol $\propto$ indicates that we omit the renormalizing factor. Hence, we obtain that $\mathbb{P}\left(\tau = u \,|\, \mathtt{adv}\right) > \mathbb{P}\left(\tau = v \,|\, \mathtt{adv}\right)$ if and only if $|\overrightarrow{\mathtt{adv}}(u)| > |\overrightarrow{\mathtt{adv}}(v)|$. ◀

G.2 A Lower Bound for the Move Complexity

▶ Observation 24. For any Δ and d, there exists a tree of depth d and maximal degree at most Δ for which any search algorithm A has move complexity $\mathcal{M}(A) = \Omega(dq\Delta)$. In particular, when $q \sim 1/\sqrt{\Delta}$, we have $\mathcal{M}(A) = \Omega(d\sqrt{\Delta})$.

Proof. To see why the observation holds consider the caterpillar tree, composed of a path of length n/Δ with each of its nodes being the center of a star graph of degree Δ. Assume that the agent starts at one of the end sides of the path and the treasure at distance d on the caterpillar spine. Recall that we assume that the algorithm does not know the tree structure. in expectation, $\Omega(dq)$ nodes will point at an incorrect neighbor, and to pass from any of those to the next node on the path, will require the agent to perform $\Omega(\Delta)$ trials in expectation. ◀

G.3 Proof of Observation 11

A randomized strategy may be viewed as a convex combination of deterministic strategies. The performance of a randomized strategy is thus a linear combination of the performance of deterministic ones. Hence, it suffices to focus on deterministic strategies.

Since the treasure location is uniform over the k objects, a deterministic strategy finds it after exactly i attemps with probability $1/k$ for any $i \leq k$. In other words, the number of objects that are tried is distributed as a uniform random variable in $[1, k]$. Such a random variable has mean $1/k \sum_{i=1}^{k} i = (k+1)/2$.

H Memoryless Algorithms and the Semi-Adversarial Model

In this section we present our results on the memoryless algorithms described in the introduction. As mentioned, such algorithms can perform well also in a more difficult semi-adversarial setting. Before we present these algorithms let us first describe formally the semi-adversarial variant.

▶ **Definition 25** (The Semi-Adversarial Model)**.** As in the purely-probabilistic Noisy Advice Model, each node is chosen to be *faulty* with probability q, and otherwise it is *sound*. Also, similarly to the original model, a sound vertex always points at its correct neighbors. However, in the semi-adversarial model, a faulty node u no longer points at a neighbor chosen uniformly at random, and instead, the neighbor w which such a node points at 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 and which is sound.

H.1 Lower Bound in the Semi-Adversarial Variant

The following result implies that if $q > 1/\Delta$ then any algorithm must have exponential query and move complexity in the depth D (or polynomial in n).

▶ **Theorem 26.** *Consider an algorithm in the semi-adversarial model. On the complete Δ-ary tree of depth D, the expected number of queries to find the treasure is $\Omega\left((q\Delta)^D\right)$. 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 assume that the treasure is located at a leaf. The adversary behaves as follows. For any advice it gets a chance to manipulate, it would 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. It follows from Observation 11 that the time to find the treasure can only be linear in the number of leaves which is $\Omega(\Delta^D)$. ◀

H.2 Probabilistic Following Algorithms

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 semi-adversarial model. Hence, together with Theorem 26, it implies that in order to achieve efficient search, the noise parameter threshold for the semi-adversarial model is $\Theta(1/\Delta)$.

▶ **Theorem 27.** 1. *Assume that for every u, $q_u < 1/(10\Delta_u)$. Then PF with parameter $\lambda \in [0.7, 0.8]$ finds the treasure in less than $100d$ expected steps, even in the semi-adversarial setting.*
2. *Consider the complete Δ-ary tree and assume that $q > 10/\Delta$. Then for any 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.

It will be convenient to picture the tree as rooted at the target node τ. For any 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 at u', while incorrect advice points at 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 a variable involving the advice at u only and the advice on the set of u's descendants (the two being independent).

We denote by $t(u)$ the time it takes to reach node u. Manipulating average symbols such as $\mathbb{E}$ requires extra care. Indeed, 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 averaging over the advice, while we use E_u to denote expectation over the walk, conditioning on u being the starting node. As a remark, observe that $E_u(t(v))$ depends on the advice configuration, it is a random variable with respect to the advice, while $\mathbb{E}E_u(t(v))$ really is just a number.

The following is the central lemma of this section.

▶ **Lemma 28.** *Assume that for every vertex u, $q_u < 1/(10\Delta_u)$, and $\lambda \in [0.7, 0.8]$. Then for all nodes u, $\mathbb{E}E_u t(u') \leq 100$. The result holds also in the semi-adversarial model.*

Let us now see how we can conclude the proof of the first item in Theorem 27, 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. With these notations, 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, by linearity of expectation:

$$\sum_{i=1}^{d(\sigma,\tau)} \mathbb{E}[E_\sigma \delta_i] \ .$$

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 28, 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 28. 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

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

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, $E_u\tau(u')$ 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*. Note that 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

$$E_u t(u') = 1 + L_u \cdot N_u. \tag{10}$$

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'. Note that $\{T \geq t\}$ 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 (10), which is only an average over the walk, by taking the expectation over all advice in layers $\geq i$. To this aim, note that L_u can be written as

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

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. Note that, 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 semi-adversarial model. Hence when we average, we can first average over layers $> i$ to obtain, denoting $\mathbb{E}^{>i}$, the expectation over the layers $> i$,

$$\begin{aligned} \mathbb{E}^{>i} E_u t(u') &= 1 + \left(1 + \sum_{v \neq u', v \sim u} p_{u,v} \mathbb{E}^{>i} E_v t(u)\right) N_u, \\ &= 1 + \left(1 + \sum_{v \neq u', v \sim u} p_{u,v} \mathbb{E} E_v t(u)\right) N_u. \end{aligned} \tag{11}$$

and using the fact that,

$$\sum_{v \neq u'} p_{u,v} \leq 1, \tag{12}$$

together with the induction assumption at rank $i+1$, we obtain

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

From now on we replace 100 by a parameter $\kappa > 0$, for mere aesthetic reasons. Averaging over the layer i of advice we obtain

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

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

$$\begin{aligned} \mathbb{E} N_u &\leq \left(\frac{1}{\lambda + \frac{(1-\lambda)}{\Delta_u}} - 1\right) + q_u \left(\frac{\Delta_u}{1-\lambda} - 1\right) \\ &\leq \frac{1}{\lambda} - 1 + \frac{q_u \Delta_u}{1-\lambda} \end{aligned} \tag{13}$$

And so it follows that

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

Hence if $q_u\Delta_u < 0.1$ and we choose $\lambda \in [0.7, 0.8]$ (for instance, we made no attempt in optimizing these constants), we see that $\mathbb{E}N_u < 0.8$. This is because

$$\frac{1}{\lambda} - 1 + \frac{0.1}{1-\lambda} \leq \frac{10}{7} - 1 + \frac{0.1}{1-0.8} < 0.93$$

Hence it follows that

$$\mathbb{E}E_u t(u') \leq 1 + 0.93(1+\kappa) < \kappa.$$

The last inequality holds by choice of $\kappa = 100$. By our (backwards) induction, we have just shown that, if $q < \frac{1}{10\Delta}$ and we set $\lambda \in [0.7, 0.8]$ then for all nodes u in the tree

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

This concludes the proof of Lemma 28 and hence also of the first part of Theorem 27. ◀

Let us explain how the lower bound in the second part of Theorem 27 is derived in the case that $q\Delta > 10$. We assume we are in a complete Δ-ary tree under our usual 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'. In this case, 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 any choice of λ, since $\Delta \geq 2$. We proceed very similarly, by induction, and use Equality (11) together with the previous bound on $\mathbb{E}(N_u)$ to obtain that for any 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 $\Omega(3^D)$. ◀

▶ Remark. Note that the proof uses crucially the tree structure and does not extend to general graphs straightforwardly. Specifically, on a tree there is a single path from σ to τ and so the points u_i are uniquely defined, they are not random. Moreover an excursion from a node u at Layer i that does not visit it's parent can only remain in layers $\geq i$. This was used through the fact that $E_v t(u)$ depends only on the advice at layers $\geq i$, if we start at a node $v \in \mathcal{L}_i$.