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 an adversary chooses the direction of advice at faulty nodes. For this variant, the threshold for efficient moving algorithms happens when the noise parameter is roughly $1/\Delta$. 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, Theory of computation → Database theory, Theory of computation → Theory of randomized search heuristics

Keywords and phrases Data structures, Graph search, Average Case Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.54

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1611.01403`.

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., [12, 19, 4, 11]). 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 [7] albeit in the context of sorting). Searching in contexts of permanently faulty nodes has been studied in a number of works [8, 13, 16, 17, 18], 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 $\texttt{A}$ by two measures: The move complexity, termed $\mathcal{M}(\texttt{A})$, is the expected number of edge traversals, and the query complexity, termed $\mathcal{Q}(\texttt{A})$, is the expected number

[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).

of queries[2]. Expectation is taken over both the randomness involved in sampling advice and the possible probabilistic choices made by $\mathtt{A}$. 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.

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.

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.

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

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 the following lower bound. The main part of the proof is to be found in Section 4. We refer the reader to the full version of this work [5, Section 2 and Appendix A] for the missing parts.

▶ **Theorem 4.** *The following holds for any randomized algorithm $\mathtt{A}$ 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 $\mathcal{Q}(\mathtt{A})$ and $\mathcal{M}(\mathtt{A})$ are exponential in D.
2. ***Lower bounds for any q.***
   (a) Consider a complete Δ-ary tree. Then $\mathcal{Q}(\mathtt{A}) = \Omega(q\Delta\log_\Delta n)$.
   (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 $\mathcal{M}(\mathtt{A}) = \Omega(dq\Delta)$.

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 [15]. 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 the full version of this work [5, Appendix E].

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., [20, 3, 22, 21]. Within the literature on search, many works considered noisy queries [12, 19, 11], 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 [16, 17, 18], as part of the more general framework of the *liar models* [1, 2, 6, 9, 23]. Data structures with adversarial memory faults have been investigated in the so called faulty-memory RAM model introduced in [14]. In particular, data structures supporting the same operations as search trees with adversarial memory faults were studied in [13, 8]. Interestingly, the data structures developed in [8] can cope with up to $O(\log n)$ faults, happening at any time during the execution of the algorithm, while maintaining optimal space and time complexity. 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 [7].

The noisy advice model considered in this paper actually originated in the recent biologically centered work [15], 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) [24, 10]. 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 only shown in the full version of this work [5, Appendix G]. 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 the full version of this work.

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 `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 `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 `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 `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:

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

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:

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

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

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

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 $\mathtt{score}(u) > \mathtt{score}(v)$, 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):

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

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 the full version of this paper [5, Appendix D], that this approach fails on other (non-complete) Δ-ary trees.

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

[3] It is tempting to define $\mathtt{score}(u)$ 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.

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 `adv` define:

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

Algorithm $\texttt{A}_{walk}$ keeps track of all vertices that are children of the vertices it explored so far, and repeatedly walks to and then explores the one with highest score according to the current advice, breaking ties arbitrarily. 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 can be found in Appendix C of the full version [5].

▶ **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 $c = 1 - \varepsilon$,

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

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:

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

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 $\Delta^h \le \beta(v) < \Delta^{h+1}$. Denote the second kind a *nominee.* Call a nominee *promising* if the number of weighted arrows pointing to v is large, specifically, if $\sum_{w\in[u,v\rangle} X_w \ge \frac{2}{3}h\log\Delta$, 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 the full version of this work, [5, Appendix F]. The part regarding regular trees will be needed later.

▶ **Lemma 9.** *For any u, $\mathbb{P}(u$ is h-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 $\mathbb{P}(X)\,\mathcal{Q}(\mathtt{local}_h(u) \mid X) = \mathcal{O}(\sqrt{\Delta}\log\Delta\cdot h)$. 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 $\mathbb{P}(u \text{ is misleading}) \le 1/n^2$. Denote by $\mathtt{Good}$ the event that none of the separators encountered are misleading. By a union bound, $\mathbb{P}(\mathtt{Good}^{\,c}) \le 1/n$.

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

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, we only sketch the argument here and refer the interested reader to the full version for details. 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) of Theorem 4. Items (2a) and (2b) are proved in Appendix A of the full version of this paper.

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 stated here without proof:

▶ **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:

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

Item (1) in Theorem 4 follows.

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 11.** *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. URL: `http://arxiv.org/abs/1302.2426`.
2. Javed A. Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *STOC*, pages 486–493. ACM, 1991. [doi:10.1145/103418.103469,].
3. Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999. [doi:10.1137/S009753979731858X,].
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. [doi:10.1109/FOCS.2008.58,].
5. Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching a tree with permanently noisy advice. *https://arxiv.org/abs/1611.01403*, 2016. [arXiv:1611.01403,].
6. Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *STOC*, pages 130–136. ACM, 1993.
7. Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347112`.

8 Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 347–358, 2007. [doi:10.1007/978-3-540-75520-3_32,].

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

10 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.

11 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *STOC*, pages 519–532, 2016. [doi:10.1145/2897518.2897656,].

12 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. [doi:10.1137/S0097539791195877,].

13 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient search trees. In *SODA*, pages 547–553, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283442`.

14 Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *STOC*, pages 101–110, 2004. [doi:10.1145/1007352.1007375,].

15 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. URL: `https://elifesciences.org/content/5/e20185`.

16 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. [doi:10.1145/1835698.1835781,].

17 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. [doi:10.1016/j.tcs.2008.04.034,].

18 Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004. [doi:10.1016/S0166-218X(03)00189-6,].

19 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283478`.

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

21 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *SODA*, pages 1096–1105, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347202`.

22 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *FOCS*, pages 379–388, 2006. [doi:10.1109/FOCS.2006.32,].

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

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