

Deterministic and Probabilistic Binary Search in Graphs

Ehsan Emamjomeh-Zadeh* David Kempe† Vikrant Singhal‡

August 1, 2017

Abstract

We consider the following natural generalization of Binary Search: in a given undirected, positively weighted graph, one vertex is a *target*. The algorithm’s task is to identify the target by adaptively querying vertices. In response to querying a node q , the algorithm learns either that q is the target, or is given an edge out of q that lies on a shortest path from q to the target. We study this problem in a general noisy model in which each query independently receives a correct answer with probability $p > \frac{1}{2}$ (a known constant), and an (adversarial) incorrect one with probability $1 - p$.

Our main positive result is that when $p = 1$ (i.e., all answers are correct), $\log_2 n$ queries are always sufficient. For general p , we give an (almost information-theoretically optimal) algorithm that uses, in expectation, no more than $(1 - \delta) \frac{\log_2 n}{1 - H(p)} + o(\log n) + O(\log^2(1/\delta))$ queries, and identifies the target correctly with probability at least $1 - \delta$. Here, $H(p) = -(p \log p + (1 - p) \log(1 - p))$ denotes the entropy. The first bound is achieved by the algorithm that iteratively queries a 1-median of the nodes not ruled out yet; the second bound by careful repeated invocations of a multiplicative weights algorithm.

Even for $p = 1$, we show several hardness results for the problem of determining whether a target can be found using K queries. Our upper bound of $\log_2 n$ implies a quasipolynomial-time algorithm for undirected connected graphs; we show that this is best-possible under the Strong Exponential Time Hypothesis (SETH). Furthermore, for directed graphs, or for undirected graphs with non-uniform node querying costs, the problem is PSPACE-complete. For a semi-adaptive version, in which one may query r nodes each in k rounds, we show membership in Σ_{2k-1} in the polynomial hierarchy, and hardness for Σ_{2k-5} .

*Department of Computer Science, University of Southern California, emamjome@usc.edu

†Department of Computer Science, University of Southern California, dkempe@usc.edu

‡Department of Computer Science, University of Southern California, vikrants@usc.edu

1 Introduction

One way of thinking about the classical Binary Search algorithm is as follows: Given a path with a target vertex t at an unknown position, an algorithm adaptively queries vertices, with the goal of discovering t . In response to a query of q , the algorithm either learns that q is the target, or finds out whether the target is to the left or right of q .

This view of Binary Search suggests a natural generalization to trees. The algorithm knows the tree ahead of time, and is trying to find the target t in the tree. When a vertex q is queried which is not the target, it reveals the subtree in which the target is located. As shown by Jordan [24], every tree has a separator vertex with the property that each of its subtrees contains at most half of the vertices. Thus, as pointed out by Onak and Parys [34], iteratively querying such a separator of the current remaining subtree, then eliminating all subtrees known not to contain the target finds a target in a tree of n vertices within at most $\log n$ queries.¹ Since Binary Search is optimal for the line, this bound is also tight in the worst case. For specific graphs, however, fewer queries may be enough: for instance, only a single query suffices for a star. The algorithmic problem of finding the optimal adaptive strategy for a tree has been solved by Onak and Parys [34], who present a linear-time algorithm. This view of Binary Search in turn suggests a further generalization to arbitrary undirected graphs: searching for a target by adaptively querying vertices of a connected undirected graph and receiving directional information.

A sequence of papers [18, 25, 5] have studied a probabilistic version of Binary Search on a path, where each answer to a query is correct only with probability $p > \frac{1}{2}$. The upshot of this line of work is that a more careful variant of Binary Search uses $O(\log n)$ queries in expectation, with an information-theoretically optimal constant depending on p . In our present work, we consider this natural generalization to the noisy model for arbitrary graphs as well.

More formally, we study algorithms under the following general model: given an arbitrary undirected (for some of our results, directed) graph $G = (V, E)$ (with $n = |V|, m = |E|$) with known positive edge weights² ω_e , an unknown target node t is to be located. When a vertex q is queried, the algorithm receives a correct response with probability $p > \frac{1}{2}$, and an incorrect one with probability $1 - p$.

- If the query is answered correctly, the response is that q is the target, or an edge e , incident on q , which lies on a *shortest* q - t path. If there are multiple such edges e , then the one revealed to the algorithm is arbitrary, i.e., it is a correct answer, but could be chosen adversarially among correct answers.
- If the query is answered incorrectly, the response is completely arbitrary, and chosen by an adversary. Of course, the adversary may choose to actually give a correct answer.

This model subsumes target search on a tree, where there is a *unique* path from q to t . Therefore, algorithms for it can be seen as natural generalizations of the fundamental Binary Search algorithm to arbitrary graphs (and hence finite metric spaces). As a first result, we show (in Section 2) that in the absence of incorrect answers, the positive results from the line (and tree) generalize to arbitrary weighted undirected graphs:

Theorem 1 *For $p = 1$, in any connected undirected graph with positive edge weights, a target can be found in at most $\log n$ queries in the worst case.*

¹Throughout this paper, unless noted otherwise, all logarithms are taken base 2.

²Thus, we can equivalently think of searching for a target in a finite metric space, defined by the shortest-path metric. An extension to infinite metric spaces is discussed briefly in Section 5.

In the noisy case as well, the positive results from the line extend to arbitrary weighted undirected graphs, albeit with a more complex algorithm and analysis. In Section 3, we prove the following theorem:

Theorem 2 *For any $p > \frac{1}{2}$ and any connected undirected graph with positive edge weights, a target can be found with probability $1 - \delta$ in $(1 - \delta) \frac{\log n}{1 - H(p)} + o(\log n) + O(\log^2(1/\delta))$ queries in expectation. ($H(p) = -p \log p - (1 - p) \log(1 - p)$ is the entropy.)*

The bound in Theorem 1 is tight even for the line. Notice also that randomization cannot help: it is easy to see that finding a target selected uniformly at random from the line takes $\Omega(\log n)$ queries in expectation, and Yao’s Minimax Principle therefore implies the same lower bound for any randomized algorithm. Moreover, as pointed out by Ben-Or and Hasidim [5], the bound in Theorem 2 is tight up to the $o(\log n) + O(\log^2(1/\delta))$ term even for the line.

Hardness Results

A natural optimization version of the problem is to ask, given an instance, what is the minimum number of queries necessary in the worst case to find a target; or, as a decision problem, whether — given a graph and target number K of queries — an unknown target can be found within K queries. For trees, a sequence of papers discussed in detail below [22, 41, 34] establishes that this problem can be solved in linear time.

For general graphs, even in the special case $p = 1$, the complexity of the problem and some of its variants is quite intriguing. The upper bound from Theorem 1 implies a simple $O(m^{\log n} \cdot n^2 \log n)$ -time algorithm using exhaustive search, thus making it unlikely that the problem is NP-hard. (The reason is that the exhaustive search tree’s height will be no more than $\log n$.) However, we show in Section 4 that the given quasipolynomial time is essentially the limit for the running time: an $O(m^{o(\log n)})$ -time algorithm would contradict the Exponential-Time Hypothesis (ETH), and an $O(m^{(1-\epsilon)\log n})$ -time algorithm for any constant $\epsilon > 0$ would contradict the Strong Exponential-Time Hypothesis (SETH).

A natural further generalization is to make the queries *semi-adaptive*: in each of k rounds, the algorithm gets to query r nodes. We consider this model with $p = 1$. It is immediate that for $k = 1$, the problem is NP-complete. For constant $k > 2$, we show (in Section 4) that the decision problem is in Σ_{2k-1} , the $(2k - 1)^{\text{th}}$ level of the polynomial hierarchy. We give a nearly matching lower bound, by showing that it is also Σ_{2k-5} -hard. Thus, semi-adaptive Binary Search on undirected graphs defines a natural class of problems tracing out the polynomial hierarchy.

Even the fully adaptive version with $p = 1$ can easily become computationally intractable with small modifications. Specifically, we show in Section 4 that the problem is PSPACE-complete for directed graphs³ and also for undirected graphs in which nodes may have non-uniform query costs, and the target is to be identified within a given budget. All the hardness proofs are similar to each other, and they are presented together.

Edge Queries on Trees

In Appendix D, we turn our attention to a more frequently studied variant of the problem on trees, in which queries are posed to edges instead of vertices. The algorithm can query an edge $e = (u, v)$, and the response reveals whether the target is in the subtree rooted at u or at v . (We assume

³Note that for a directed cycle, $n - 1$ queries are necessary in the worst case, as answers to a query to a non-target node reveal nothing about the target.

here that $p = 1$.) This provides less information than querying one of the two vertices, and the difference can be significant when vertices have high degrees.

The edge query variant is considered in several recent papers, which have focused on the algorithmic question of finding the optimal adaptive strategy which minimizes the worst-case number of queries [4, 34, 31]. Upper and lower bounds on the maximum number of queries required in trees of maximum degree Δ had been shown by Ben-Asher and Farchi to be $\log_{\Delta/(\Delta-1)}(n) \in \Theta(\Delta \log n)$ and $\frac{\Delta-1}{\log \Delta} \log n$, respectively [3]. While the authors call these bounds “matching,” there is a gap of $\Theta(\log \Delta)$ between them. In Appendix D, we present an improved algorithm for the problem, which uses at most $1 + \frac{\Delta-1}{\log(\Delta+1)-1} \log n$ queries in the worst case, thus practically matching the lower bound of [3].

Related Work

The algorithmic problem of identifying a target by *edge queries* in DAGs was initiated by Linal and Saks [30], who studied it for several specific classes of graphs. In general, the problem is known to be NP-hard [11, 15]. As a result, several papers have studied it specifically on trees.

On trees, both the vertex and the edge query questions (when $p = 1$) are equivalent to the respective vertex and edge ranking problems (see, e.g., [22, 41, 29]). The vertex ranking problem requires that each vertex be assigned a label ℓ_v such that if two vertices u, v have the same label ℓ , then there is some vertex w on the u - v path with strictly larger label. The labels can be interpreted as the order in which vertices will be queried (decreasingly). For the vertex ranking problem, Iyer, Ratliff, and Vijayan [22] gave an $O(n \log n)$ algorithm, which was improved to linear time by Schäffer [41]. Similarly, the edge ranking problem was solved for trees in linear time by Lam and Yue [29]. Lam and Yue [28] also established that for general graphs, the problem is NP-hard.

Following the work of Schäffer [41] and Lam and Yue [29], linear-time algorithms for both versions (the vertex and edge query models in trees) were rediscovered by Onak and Parys [34] and Mozes, Onak, and Weimann [31]. The former paper gives an $O(n^3)$ algorithm for the edge query model (improving on the $O(n^4 \log^3 n)$ algorithm of Ben-Asher, Farchi, and Newman [4]), while also providing a linear-time algorithm for the vertex query model. The $O(n^3)$ algorithm was subsequently improved to linear time in [31].

Edge ranking and vertex ranking elegantly encode querying strategies for trees; however, they crucially exploit the fact that the different possible responses to a vertex or edge query identify disjoint components. For general graphs, the resulting structures are not as simple. Indeed, it is known that for general graphs, finding an optimal edge ranking is NP-hard [28]. More evidence of the hardness is provided by our PSPACE-completeness result.

A natural generalization of the problem is to introduce non-uniform costs on the vertices or edges. This problem is studied for lines and trees by Laber, Milidiú and Pessoa [27] and Cicalese et al. [13], respectively (among others). In particular, Cicalese et al. [13] show that with edge query costs, the problem is NP-hard when the diameter of the tree is at least 6 and the maximum degree is at least 3. On the other hand, [13] presents a polynomial-time algorithm for the case where the diameter of the given tree is at most 5.

An even more significant departure from the model we consider is to assume that a probability distribution is given over target locations, instead of the worst-case assumption made here. The goal then typically is to minimize the expected number (or total cost) of queries. This type of model is studied in several papers, e.g., [27, 12]. The techniques and results are significantly different from ours.

Noisy Binary Search is a special case of a classical question originally posed by Rényi [39], and subsequently restated by Ulam [42] as a game between two parties: one party asks questions,

and the other replies through a noisy channel (thus lying occasionally). Several models for noise are examined in detail in [38]. Rivest et al. [40] study a model in which the number of queries the adversary can answer incorrectly is a constant (or a function of n , but independent of the total number of queries). When the adversary can answer at most a constant fraction r of queries incorrectly, Dhagat et al. [16], proved the impossibility of finding the target for $r \geq \frac{1}{3}$, and provided an $O(\log n)$ algorithm when $r < 1/3$. Pedrotti [36] further improved the constant factors in this bound. A modified version of this model, called the *Prefix-Bounded Model*, states that for every i , at most an r fraction of the initial i query responses may be incorrect. In this model, Pelc [37] (using different terminology) gives an $O(\log n)$ upper bound on the number of the queries for every $r < \frac{1}{3}$. Borgstrom and Kosaraju [6] generalize this bound to any $r < \frac{1}{2}$. Aslam [2] also studies the Prefix-Bounded Model and provides a reduction from the independent noise model to it.

The k -batch model, which is similar to our notion of semi-adaptive querying, is a different model in which the questioner can ask multiple non-adaptive questions in a series of k batches. Cicalese et al. [14] consider the 2-batch model, under the assumption that at most a given number of query responses are incorrect.

The independent noise model (used in our paper) is analyzed by Feige et al. [18]. They use additional queries to backtrack in a search tree; repeated queries to a vertex are used to obtain improved error probabilities for queries. They obtain a $\Theta(\log n)$ bound on the number of queries with a large (non-optimal) constant depending on p . Karp and Kleinberg [25] consider the problem to find, in an array of coins sorted by increasing probability of “heads,” the one maximizing the probability of “heads” subject to not exceeding a given target. They show that this problem shares a lot of similarities with noisy Binary Search⁴, and use similar techniques for positive results; they also provide information-theoretic lower bounds. Their information theoretic lower bounds for the problem are matched by Ben-Or and Hassidim [5], whose noise model is the same as ours. Ben-Or and Hassidim further generalize their algorithm to achieve improved bounds on the query-complexity of quantum Binary Search.

Horstein [20] provides another motivation for noisy Binary Search (on the continuous line): a sender wants to share a (real) number with a receiver over a noisy binary channel with a perfect feedback channel. That is, the receiver receives each bit incorrectly with probability p , but the sender knows which bit was received. Horstein [20] analyzes noisy Binary Search as an algorithm under this model with a prior on the number to be transmitted: the sender always sends the bit corresponding to the comparison between the number and the median according to the posterior distribution of the receiver. Jedynek et al. [23] show that this algorithm is optimal in the sense of minimizing the entropy of the posterior distribution, while Waeber et al. [43] show that the expected value of the distance between this algorithm’s outcome and the target point decreases exponentially in the number of the queries.

Noisy Binary Search has been a significant area of study in information theory and machine learning, where it has often been rephrased as “Active Bayesian Learning.” Its uses can be traced back to early work on parameter estimation in statistics, e.g., work by Farrell [17] and Burnashev and Zigangirov [9]. It often takes the form of a search for the correct function in a function space that maps a given instance space to an image space. The initial algorithm for the noisy version with image space $\{-1, +1\}$ was proposed by Nowak in [33]. Its running time is $O(\log |H|)$, where H is the hypothesis/function space. A variation of the model was studied by Yan et al. [44]. Here, the instance space, image space and hypothesis space are, respectively, $[0, 1]$, $\{0, 1\}$ and $\{f_\theta : [0, 1] \rightarrow \{0, 1\} \mid \theta \in [0, 1], f_\theta(x) = 1 \text{ if } x \geq \theta, 0 \text{ otherwise}\}$. They provide an $O(k^{-c})$ error bound, where k is the number of queries performed, and c is a constant determined by other

⁴In fact, this is a more general model than noisy Binary Search on a path.

parameters defining the model. The goal is to approximate the correct value of θ , with the constraint that the oracle might lie or abstain from providing an answer. Both abstentions and lies might become more frequent as the query points get closer to the actual threshold. For general image spaces, Naghshvar et al. in [32] also obtain an $O(\log |H|)$ bound, with improved constants for the special case presented in [33].

Notation and Preliminaries

The algorithm is given a positively weighted connected undirected (or directed and strongly connected) graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Edge weights are denoted by $\omega_e > 0$. The distance $d(u, v)$ between vertices u, v is the length of a shortest u - v path with respect to the ω_e . For undirected graphs, $d(u, v) = d(v, u)$.

For a vertex u and edge $e = (u, v)$ incident on u (out of u for directed graphs), we denote by $N(u, e) = \{w \in V \mid d(u, w) = \omega_e + d(v, w)\}$ the set of all vertices w for which e lies on a shortest u - w path. Let t be the (unknown) target vertex. The guarantee made to the algorithm is that when it queries a vertex q which is not the target, with probability p , it will be given an edge e out of q such that $t \in N(q, e)$. Since G is assumed (strongly) connected, such an answer will always exist; we reiterate here that if multiple edges e exist such that $t \in N(q, e)$, an arbitrary one (possibly chosen adversarially) will be returned. If q is the target, then with probability p , this fact is revealed to the algorithm. In both cases, with the remaining probability $1 - p$, the algorithm is given an arbitrary (adversarial) answer, which includes the possibility of the correct answer.

2 Deterministic Model

We first analyze the case $p = 1$, i.e., when responses to queries are noise-free. We establish a tight logarithmic upper bound on the number of queries required, which was previously known only for trees [34].

Theorem 3 *There exists an adaptive querying strategy with the following property: given any undirected, connected and positively weighted graph G with an unknown target vertex t , the strategy will find t using at most $\log n$ queries. (This bound is tight even when the graph is the line.)*

Proof. In each iteration, based on the responses the algorithm has received so far, there will be a set $S \subseteq V$ of *candidate* nodes remaining one of which is the target. The strategy is to always query a vertex q minimizing the sum of distances to vertices in S , i.e., a 1-median of S (with ties broken arbitrarily). Notice that q itself may not lie in S .

More formally, for any set $S \subseteq V$ and vertex $u \in V$, we define $\Phi_S(u) = \sum_{v \in S} d(u, v)$. The algorithm is given as Algorithm 1.

First, note that $\Phi_S(q)$ and $N(q, e)$ can be computed using Dijkstra's algorithm, and q can be found by exhaustive search in linear time, so the entire algorithm takes polynomial time. We claim that it uses at most $\log n$ queries to find the target t . To see this, consider an iteration in which t was not found; let $e = (q, v)$ be the response to the query. We write $S^+ = S \cap N(q, e)$, and $S^- = S \setminus S^+$. By definition, the edge e lies on a shortest q - u path for all $u \in S^+$, so that $d(v, u) = d(q, u) - \omega_e$ for all $u \in S^+$. Furthermore, for all $u \in S^-$, the shortest path from v to u can be no longer than those going through q , so that $d(v, u) \leq d(q, u) + \omega_e$ for all $u \in S^-$. Thus, $\Phi_S(v) \leq \Phi_S(q) - \omega_e \cdot (|S^+| - |S^-|)$. By minimality of $\Phi_S(q)$, it follows that $|S^+| \leq |S^-|$, so $|S^+| \leq \frac{|S|}{2}$. Consequently, the algorithm takes at most $\log n$ queries. ■

Algorithm 1 TARGET SEARCH FOR UNDIRECTED GRAPHS

```
1:  $S \leftarrow V$ .
2: while  $|S| > 1$  do
3:    $q \leftarrow$  a vertex minimizing  $\Phi_S(q)$ .
4:   if  $q$  is the target then
5:     return  $q$ .
6:   else
7:      $e = (q, v) \leftarrow$  response to the query.
8:      $S \leftarrow S \cap N(q, e)$ .
9: return the only vertex in  $S$ .
```

Notice that Theorem 3 implies a quasipolynomial time algorithm for finding an optimal adaptive strategy for a given undirected graph. Because an optimal strategy can use at most $\log n$ queries, an exhaustive search over all possible vertices q to query at each stage, and all possible edges that could be given as responses (of which there are at most as many as the degree of q) will require at most $O(m^{\log n} n^2 \log n)$ time. In particular, the decision problem of determining whether, given an undirected graph G and an integer K , there is an adaptive strategy using at most K queries, cannot be NP-hard unless $\text{NP} \subseteq O(n^{O(\log n)})$. We later show hardness results for this problem based on two hypotheses: ETH and SETH.

A natural question is how much the logarithmic bound of Theorem 3 depends on our assumptions. In Appendix B, we generalize this theorem to strongly connected graphs which are *almost undirected* in the following sense: each edge e with weight ω_e belongs to a cycle of total weight at most $c \cdot \omega_e$, for a constant c .

3 Noisy Model

When $\frac{1}{2} < p < 1$, i.e., each response is incorrect independently with probability $1 - p$, our goal is to find the target with probability at least $1 - \delta$, for a given δ , minimizing the expected number of queries in the worst case. (When $p \leq \frac{1}{2}$, even for the path, it is impossible to get any useful information about the target with any number of queries.⁵)

First, note that for the path (or more generally, trees), a simple idea leads to an algorithm using $c \log n \log \log n$ queries (where c is a constant depending on p): simulate the deterministic algorithm, but repeat each query $c \log \log n$ times and use the majority outcome. A straightforward analysis shows that with a sufficiently large c , this finds the target with high probability. For general graphs, however, this idea fails. The reason is that if there are multiple *correct* answers for the query, then there may be no majority among the answers, and the algorithm cannot infer which answers are correct.

The $c \log n \log \log n$ bound for a path is not information-theoretically optimal. A sequence of papers [18, 25, 5] improved the upper bound to $O(\log n)$ for simple paths. More specifically, Ben-Or and Hassidim [5] give an algorithm that finds the target with probability $1 - \delta$, using $\frac{(1-\delta) \log n}{C(p)} + o(\log n) + O(\log(1/\delta))$ queries, where $C(p) = 1 - H(p) = 1 + p \log p + (1 - p) \log(1 - p)$. As pointed out in [5], this bound is information-theoretically optimal up to the $o(\log n) + O(\log(1/\delta))$ term.

⁵Recall that an “incorrect” query response is adversarial; as the adversary may also respond correctly, an algorithm cannot simply flip all answers.

In this section, we extend this result to general connected undirected graphs. At a high level, the idea of the algorithm is to keep track of each node’s *likelihood* of being the target, based on the responses to queries so far. Generalizing the idea of iteratively querying a median node from Section 2, we now iteratively query a *weighted* median, where the likelihoods are used as weights. Intuitively, this should ensure that the total weight of non-target nodes decreases exponentially faster than the target’s weight. In this sense, the algorithm shares a lot of commonalities with the multiplicative weights update algorithm, an idea that is also prominent in [25, 5], for example.

The high-level idea runs into difficulty when there is one node that accumulates at least half of the total weight. In this case, such a node will be the queried weighted median, and the algorithm cannot ensure a sufficient decrease in the total weight. On the other hand, such a high-likelihood node is an excellent candidate for the target. Therefore, our algorithm *marks* such nodes for “subsequent closer inspection,” and then removes them from the multiplicative weights algorithm by setting their weight to 0.

The key lemma shows that with high probability, within $\Theta(\log n)$ rounds, the target node has been marked. Therefore, the algorithm could now identify the target by querying each of the $O(\log n)$ marked nodes $\Theta(\log \log n)$ times, and then keeping the node that was identified as the target most frequently.

However, because there could be up to $\Theta(\log n)$ marked nodes, this could still take $\Theta(\log n \log \log n)$ rounds. Instead, we run a second phase of the multiplicative weights algorithm, starting with weights only on the nodes that had been previously marked. Akin to the first phase, we can show that with high probability, the target is among the marked nodes in the first $\Theta(\log \log n)$ rounds of the second phase. Among the at most $\Theta(\log \log n)$ resulting candidates, the target can now be identified with high probability by querying each node sufficiently frequently. Because there are only $\Theta(\log \log n)$ nodes remaining, this final round takes time only $O((\log \log n)^2)$.

Given a function $\mu : V \rightarrow \mathbb{R}_{\geq 0}$, for every vertex u , we define $\Phi_\mu(u) = \sum_{v \in V} \mu(v)d(u, v)$ as the weighted sum of distances from u to other vertices. The weighted median is then $\operatorname{argmin}_{u \in V} \Phi_\mu(u)$ (ties broken arbitrarily). For any $S \subseteq V$, let $\Gamma_\mu(S) = \sum_{v \in S} \mu(v)$ be the sum of weights for the vertices in S . The following lemma, proved by exactly the same arguments as we used in Section 2, captures the key property of the weighted median.

Lemma 4 *Let u be a weighted median with respect to μ and $e = (u, v)$ an edge incident on u . Then, $\Gamma_\mu(N(u, e)) \leq \Gamma_\mu(V)/2$.*

We now specify the algorithm formally. Algorithm 2 encapsulates the multiplicative weights procedure, and Algorithm 3 shows how to combine the different pieces. Compared to the high-level outline above, the target bounds of $\Theta(\log n)$ and $\Theta(\log \log n)$ are modified somewhat in Algorithm 3, mostly to account for the case when δ is very small.

To analyze the performance of Algorithm 2, we keep track of the sum of the node weights and show, by induction, that after i iterations, $\Gamma_\mu(V) \leq 2^{-i}$. Consider the i^{th} iteration:

- If there exists a node whose weight is at least half of the total node weight, MULTIPLICATIVE-WEIGHTS sets its weight to 0. Therefore, in this case, the sum of node weights drops to at most half of its previous value.
- If MULTIPLICATIVE-WEIGHTS queries a median q , and is told that q is the target, then q ’s weight decreases by a factor of p , and the other nodes’ weights decrease by a factor of $1 - p$. The new sum of node weights is $p\mu(q) + (1 - p)(\Gamma_\mu(V) - \mu(q)) \leq \Gamma_\mu(V)/2$, because $\mu(q) \leq \Gamma_\mu(V)/2$.

Algorithm 2 MULTIPLICATIVE-WEIGHTS ($S \subseteq V, K$)

- 1: $\mu(v) \leftarrow 1/|S|$ for all vertices $v \in S$
and $\mu(v) \leftarrow 0$ for all $v \in V \setminus S$.
- 2: $M \leftarrow \emptyset$.
- 3: **for** K iterations **do**
- 4: Let q be a weighted median with respect to μ .
- 5: **if** $\mu(q) \geq \frac{1}{2} \cdot \Gamma_\mu(V)$ **then**
- 6: Mark q , by setting $M \leftarrow M \cup \{q\}$.
- 7: $\mu(q) \leftarrow 0$.
- 8: **else**
- 9: Query node q .
- 10: **for all** nodes $v \in S$ **do**
- 11: **if** v is consistent with the response **then**
- 12: $\mu(v) \leftarrow p \cdot \mu(v)$.
- 13: **else**
- 14: $\mu(v) \leftarrow (1 - p) \cdot \mu(v)$.
- 15: **return** M

Algorithm 3 PROBABILISTIC BINARY SEARCH (δ)

- 1: $\delta' \leftarrow \delta/3$.
- 2: Fix $\lambda_1 = \min(\sqrt{\frac{1}{\log \log n}}, \frac{\mathcal{C}(p)}{2 \log(p/(1-p))})$.
- 3: $K_1 \leftarrow \max\{\frac{\log n}{\mathcal{C}(p) - \lambda_1 \log(p/(1-p))} + 1, \frac{\ln(1/\delta')}{2\lambda_1^2}\}$.
- 4: $S_1 \leftarrow \text{MULTIPLICATIVE-WEIGHTS}(V, K_1)$.
- 5: Fix $\lambda_2 = \frac{\mathcal{C}(p)}{2 \log(p/(1-p))}$.
- 6: $K_2 \leftarrow \max\{\frac{\log |S_1|}{\mathcal{C}(p) - \lambda_2 \log(p/(1-p))} + 1, \frac{\ln(1/\delta')}{2\lambda_2^2}\}$.
- 7: $S_2 \leftarrow \text{MULTIPLICATIVE-WEIGHTS}(S_1, K_2)$.
- 8: **for all** $v \in S_2$ **do**
- 9: Query v repeatedly $\frac{2 \ln(|S_2|/\delta')}{(2p-1)^2}$ times.
- 10: **if** v is returned as the target for at least half of these queries **then**
- 11: **return** v .
- 12: **return** failure.

- If MULTIPLICATIVE-WEIGHTS queries a median q , and the response is an edge $e = (q, v)$, the algorithm multiplies the weights of nodes in $N(q, e)$ by p and the weights of all other nodes by $1 - p$. The new total weight is $p\Gamma_\mu(N(q, e)) + (1 - p)(\Gamma_\mu(V) - \Gamma_\mu(N(q, e)))$, which is at most $\Gamma_\mu(V)/2$, because Lemma 4 implies that $\Gamma_\mu(N(q, e)) \leq \Gamma_\mu(V)/2$.

The key lemma for the analysis (Lemma 5) shows that with high probability, the target is in the set returned by the Multiplicative Weights algorithm. Recall that $\mathcal{C}(p)$ is defined as $1 - H(p) = 1 + p \log p + (1 - p) \log(1 - p)$.

Lemma 5 *Let $S \subseteq V$ be a subset of vertices containing the target and $0 < \lambda < \frac{\mathcal{C}(p)}{\log(p/(1-p))}$. If $K \geq \frac{\ln(1/\delta')}{2\lambda^2}$ and $K > \frac{\log |S|}{\mathcal{C}(p) - \lambda \log(p/(1-p))}$, then with probability at least $1 - \delta'$, the target is in the set*

returned by `MULTIPLICATIVE-WEIGHTS(S, K)`.

Proof. By a standard Hoeffding Bound [19], the probability that fewer than $(p - \lambda)K$ queries are answered correctly is at most $e^{-2K\lambda^2}$. By the first bound on K in Lemma 5, this probability is at most δ' .

If we assume that the target is not marked by the end of the `MULTIPLICATIVE-WEIGHTS` algorithm, its final weight is at least $\mu(t) \geq \frac{(p^{p-\lambda}(1-p)^{1-p+\lambda})^K}{|S|}$, with probability at least $1 - \delta'$. On the other hand, we showed that the final sum of node weights is upper-bounded by $\Gamma \leq 2^{-K}$. Using both bounds, we obtain that

$$\begin{aligned} \log(\mu(t)/\Gamma) &\geq K \cdot \log(2p^{p-\lambda}(1-p)^{1-p+\lambda}) - \log |S| \\ &= K \cdot (1 - H(p) - \lambda \log(p/(1-p))) - \log |S| \\ &= K \cdot (\mathcal{C}(p) - \lambda \log(p/(1-p))) - \log |S| > 0, \end{aligned}$$

where the final inequality follows from the second assumed bound on K in the lemma. This implies that $\mu(t) > \Gamma$, a contradiction. \blacksquare

The next lemma shows that the final phase of testing each node finds the target with high probability.

Lemma 6 *Assuming that the target is in S , if each node $v \in S$ is queried at least $K \geq \frac{2 \ln(|S|/\delta')}{(2p-1)^2}$ times, then with probability at least $1 - \delta'$, the true target is the unique node returned as the “target” by at least half of the queries.*

Proof. We prove Lemma 6 using standard tail bounds and a union bound. Let $\lambda = p - \frac{1}{2}$. As in the proof of Lemma 5, for each queried node, a Hoeffding Bound establishes that the probability that at most $(p - \lambda)K = \frac{K}{2}$ queries are correctly answered is at most $e^{-2K\lambda^2}$. Because of the lower bound on K , this probability is upper-bounded by $\delta'/|S|$. By a union bound, the probability that any queried vertex has at least half of its queries answered incorrectly is at most δ' . Barring this event, the target vertex is confirmed as such more than half the time, while none of the non-target vertices are confirmed as target vertices at least half the time. \blacksquare

Combining these lemmas, we prove the following:

Lemma 7 *Given $\delta > 0$, Algorithm 3 finds the target with probability at least $1 - \delta$, using no more than $\frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta))$ queries.*

Proof. Because λ_1 and K_1 , satisfy both bounds of Lemma 5 by definition, Lemma 5 can be applied, implying that with probability at least $1 - \delta'$, the set S_1 contains the target. Note that the maximum in the definition of K_1 is taken over two terms. The first one is $\frac{\log n}{\mathcal{C}(p)} + o(\log n)$ because $\lambda_1 = o(1)$. The second one is $O(\log \log n \cdot \log(1/\delta')) = O((\log \log n)^2) + O(\log^2(1/\delta'))$. Therefore, $|S_1|$ is bounded by $\frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta'))$.

For the second invocation of `MULTIPLICATIVE-WEIGHTS`, the conditions of Lemma 5 are again satisfied by definition, so S_2 will contain the target with probability at least $1 - 2\delta'$. The maximum in the definition of K_2 is again taken over two terms. The first one is $O(\log \log n + \log \log(1/\delta'))$ (by the bound on $|S_1|$) and the second one is $O(\log(1/\delta'))$.

Finally, by Lemma 6, the target will be returned with probability at least $1 - 3\delta' = 1 - \delta$. The final phase again only makes $o(\log n) + O(\log^2(1/\delta))$ queries, giving us the claimed bound on the total number of queries. \blacksquare

To obtain the bound $(1 - \delta)\frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta))$ from Theorem 2, we can reason as follows:

- If $\delta < \frac{1}{\log n}$, then $\frac{\log n}{\mathcal{C}(p)} = \frac{(1-\delta)\log n}{\mathcal{C}(p)} + \frac{\delta\log n}{\mathcal{C}(p)}$, and $\frac{\delta\log n}{\mathcal{C}(p)} = O(1)$ can be absorbed into the $o(\log n)$ term.
- If $\delta \geq 1/\log n$, we can modify the algorithm using a simple trick from [5]: With probability $\delta - \frac{1}{\log n}$, the modified algorithm outputs an arbitrary node without any queries; otherwise, it runs Algorithm 3 with error parameter $\hat{\delta} = \frac{1}{\log n}$. The success probability is then $(1 - \delta + \frac{1}{\log n})(1 - \frac{1}{\log n}) \geq 1 - \delta$, while the number of queries in expectation is

$$(1 - \delta + \frac{1}{\log n}) \cdot (\frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta))) = \\ (1 - \delta) \cdot \frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta)).$$

Hence, we have proved our main theorem:

Theorem 8 *There exists an algorithm with the following property: Given a connected undirected graph and positive $\delta > 0$, the algorithm finds the target with probability at least $1 - \delta$, using no more than $(1 - \delta)\frac{\log n}{\mathcal{C}(p)} + o(\log n) + O(\log^2(1/\delta))$ queries in expectation.*

Notice that except for having $O(\log^2(1/\delta))$ instead of $O(\log(1/\delta))$, this bound matches the one obtained by Ben-Or and Hassidim [5] for the line.

4 Hardness Results

We next turn to the question of determining the optimal number of queries required to find a target, or — equivalently — the decision problem of determining whether K queries are sufficient in the worst case. All our hardness results apply already for the noise-free version, i.e., the special case where $p = 1$; they are proved in Appendix A. For undirected graphs, the result from Section 2 implies a quasipolynomial-time algorithm, which makes the problem unlikely to be NP-hard. However, we show hardness results for this problem based on two well-known conjectures.

Conjecture [ETH]: *The Exponential Time Hypothesis (ETH) [21] states that there exists a positive constant \hat{s} such that 3-CNF-SAT is not solvable in time $2^{s^n} \cdot \text{poly}(n, m)$, for any constant $s < \hat{s}$. Here, n and m are the number of variables and the number of clauses, respectively.*

Conjecture [SETH]: *The Strong Exponential Time Hypothesis (SETH) [21, 10] states that CNF-SAT does not admit any algorithm with running time $2^{(1-\epsilon)n} \cdot \text{poly}(n, m)$ for any constant $\epsilon > 0$.*

Theorem 9 *The problem of deciding, given a connected undirected graph $G = (V, E)$ and integer K , whether there is an adaptive strategy that finds the target in at most K queries*

- (1) *does not admit any algorithm with $m^{o(\log n)}$ running time unless the ETH is false.*
- (2) *does not admit any algorithm with $O(m^{(1-\epsilon)\log n})$ running time for any constant $\epsilon > 0$ unless the SETH is false.*

Our next hardness result is that the decision problem is PSPACE-complete for *directed* graphs. In fact, we will show a more fine-grained characterization for the following “semi-adaptive” version of the problem, even for undirected graphs:

Problem: *Given a connected undirected or strongly connected directed graph G , and integers k and r , the algorithm queries r vertices in each round $1, 2, \dots, k$. In response to the r queries, prior to the next round, it receives one edge out of each queried vertex, lying on a shortest path from that vertex to the target. The decision problem $\text{TARGETSEARCH}(k, r)$ is to ascertain whether the target can be found using k rounds of r queries each.*

When $r = 1$ and k is part of the input, we obtain the original problem. When $k = 1$ (i.e., under the non-adaptive query model), a straightforward reduction from SET COVER establishes NP-hardness. Our complexity-theoretic results for $\text{TARGETSEARCH}(k, r)$ are the following:

Theorem 10 1. *When the graph is directed and k is part of the input, $\text{TARGETSEARCH}(k, 1)$ is PSPACE-complete.*

2. *When $k \geq 3$ is a constant but r is part of the input, $\text{TARGETSEARCH}(k, r)$ is contained in Σ_{2k-1} in the polynomial hierarchy, and Σ_{2k-5} -hard. This holds even for undirected graphs.*

The proof consists of a reduction from the QUANTIFIED BOOLEAN FORMULAS (QBF) problem and its variant with a constant depth of quantifier nesting.

The graphs constructed in the Σ_{2k-5} -hardness proof are undirected, but the PSPACE-hardness proof crucially relies on long directed cycles to build a threat of a large number of required queries, while avoiding an exponential blowup in the reduction.⁶

If node queries can incur non-uniform cost, then the same effect can be obtained for undirected graphs, giving us the following theorem

Theorem 11 *It is PSPACE-complete to determine, given a connected and unweighted undirected graph $G = (V, E)$ with vertex cost function $\mathcal{C} : V \rightarrow \mathbb{Z}_+$ and an integer K , whether there is an adaptive query strategy which always finds the target while paying a total cost of at most K . This hardness holds even when G has diameter at most 13.*

Notice that for any connected and unweighted undirected graph with diameter D , the target can always be found using at most $D - 1$ queries, as follows: start with an arbitrary node; whenever the edge $e = (q, v)$ is revealed in response to a query q , the next vertex to query is v . If the optimal number of queries is upper-bounded by a constant, then the exhaustive search outlined in Section 2 needs only polynomial time to find an optimal querying strategy. Thus, Theorem 11 shows that non-uniform costs significantly change the problem.

5 Conclusion and Open Problems

We proved that using vertex queries (which reveal either that the target is at the queried vertex, or exhibit an edge on a shortest path to the target), a target can always be found in an undirected positively weighted graph using at most $\log n$ queries. When queries give the correct answer only with probability $p > \frac{1}{2}$, we obtain an (essentially information-theoretically optimal) algorithm which

⁶ Indeed, long cycles are the main obstacle to efficient Binary Search: in Appendix B, we show that the positive result from Section 2 can be extended to strongly connected directed graphs which are almost undirected in the sense that each edge appears in a short cycle.

finds the target with probability at least $1 - \delta$ using $(1 - \delta) \log n / (1 - H(p)) + o(\log n) + O(\log^2(1/\delta))$ queries in expectation. We also proved several hardness results for different variants, ranging from hardness under the ETH and SETH to PSPACE-completeness.

Our work raises a number of immediate technical questions for future work, but also some broader directions. Some of the more immediate directions are:

1. In light of our hardness results for finding the exact answer, how hard is it to find an *approximately* optimal strategy? Can this be done in polynomial time?
2. For directed graphs which are almost undirected, close the gap (see Appendix B) between the upper bound of $c \ln(2) \cdot \log n$, and the lower bound of $\frac{c-1}{\log c} \cdot \log n$ on the number of queries required. Clearly, if $c \in \omega(n/\log n)$, the upper bound cannot be tight, and we conjecture that the lower bound may be asymptotically tight.
3. For undirected graphs, is the problem of deciding if an adaptive strategy using at most K queries exists in NP? What about Co-NP?
4. For the semi-adaptive version, close the gap between the Σ_{2k-5} -hardness and the membership in Σ_{2k-1} . We believe that a more involved hardness proof establishes Σ_{2k-3} -hardness, which still leaves a gap with the upper bound.

The semi-adaptive version of the problem raises a number of interesting questions. In preliminary work, we have shown that for the D -dimensional hypercube (with $n = 2^D$ vertices), an algorithm using 2 queries per round can use significantly fewer rounds than one using only a single query per round (namely, $\frac{\log n}{2}$ instead of $\log n$). Surprisingly, moving from 2 to 3 or more queries per round does not give any further significant improvement; the next improvement in the worst-case number of rounds arises when the number of queries per round is $r = \Omega(\log D)$. This fact is based on a combinatorial result of Kleitman and Spencer [26]. This observation raises the question how a larger number of queries per round affects the number of rounds required. In particular, when the worst case over all connected undirected graphs is considered, will a larger number of queries per round always guarantee a multiplicatively smaller number of rounds?

As discussed in the introduction, we can think of a positively edge-weighted graph as a finite metric space. This naturally suggests a generalization to infinite metric spaces. For instance, we may try to solve a puzzle such as: “Which major city in the United States lies straight East of San Diego, and straight Southeast of Seattle?⁷” More generally, for an arbitrary metric space d , the “direction” in which the target lies from the queried point q can be characterized by revealing a point $v \neq q$ such that $d(q, v) + d(v, t) = d(q, t)$.

When the metric space is \mathbb{R}^D with $D \geq 2$, endowed with the L_p norm for $p \in (1, \infty)$, it suffices to query two points q_1, q_2 not collinear with the target: their query responses reveal two lines on which t must lie, and these lines intersect in a unique point. For the L_1 norm in $D \geq 2$ dimensions, on the other hand, an adversary can always provide only answers that get closer to the target in the first dimension, without ever revealing any information about the second. Hence, no adaptive algorithm can provide any guarantees.

For more general metric spaces, e.g., arbitrary surfaces, the question of how many queries are required to locate a point is wide open. In preliminary work, we have shown that in an arbitrary polygon with n corners, a target can always be found using $O(\log n)$ queries. However, our approach does not extend to higher genus.

⁷Dallas, TX.

There are many other fundamental generalizations of the model that are possible, and would be of interest. These include querying an approximate median instead of an exact one, receiving answers that lie on approximately shortest paths, or searching for multiple targets at once. Absent careful definitions, some of these generalizations make the problem trivially impossible to solve, but with suitable definitions, challenging generalizations arise.

6 Acknowledgments

We would like to thank Noga Alon, Sepehr Assadi, Yu Cheng, Shaddin Dughmi, Bobby Kleinberg, Daniel Roß, Shanghua Teng, Adam Wierman, and Avi Wigderson for useful discussions and pointers.

References

- [1] A. Abboud, A. Backurs, and V. Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. 56th IEEE Symp. on Foundations of Computer Science*, 2015.
- [2] J. A. Aslam. *Noise Tolerant Algorithms for Learning and Searching*. PhD thesis, 1995.
- [3] Y. Ben-Asher and E. Farchi. The cost of searching in general trees versus complete binary trees. Technical report, 1997.
- [4] Y. Ben-Asher, E. Farchi, and I. Newman. Optimal search in trees. *SIAM J. on Computing*, 28(6):2090–2102, 1999.
- [5] M. Ben-Or and A. Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, pages 221–230, 2008.
- [6] R. S. Borgstrom and S. R. Kosaraju. Comparison-based search in the presence of errors. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 130–136, 1993.
- [7] M. Braverman, Y. K. Ko, and O. Weinstein. Approximating the best Nash Equilibrium in $n^{o(\log n)}$ -time breaks the Exponential Time Hypothesis. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms*, pages 970–982, 2015.
- [8] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th IEEE Symp. on Foundations of Computer Science*, 2015.
- [9] M. V. Burnashev and K. S. Zigangirov. An interval estimation problem for controlled observations. *Problemy Peredachi Informatsii*, 10:51–61, 1974.
- [10] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Proc. of IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85, 2009.
- [11] R. Carmo, J. Donadelli, Y. Kohayakawa, and E. S. Laber. Searching in random partially ordered sets. *Theoretical Computer Science*, 321(1):41–57, 2004.

- [12] F. Cicalese, T. Jacobs, E. Laber, and M. Molinaro. On the complexity of searching in trees and partially ordered structures. *Theoretical Computer Science*, 412(50):6879–6896, 2011.
- [13] F. Cicalese, T. Jacobs, E. Laber, and C. Valentim. The binary identification problem for weighted trees. *Theoretical Computer Science*, 459:100–112, 2012.
- [14] F. Cicalese, D. Mundici, and U. Vaccaro. Least adaptive optimal search with unreliable tests. *Theoretical Computer Science*, 270(1–2):877–893, 2002.
- [15] D. Dereniowski. Edge ranking and searching in partial orders. *Discrete Applied Mathematics*, 156(13):2493–2500, 2008.
- [16] A. Dhagat, P. Gács, and P. Winkler. On playing “twenty questions” with a liar. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 16–22, 1992.
- [17] R. H. Farrell. Asymptotic behavior of expected sample size in certain one sided tests. *Annals of Mathematical Statistics*, 35(1):36–72, 1964.
- [18] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. on Computing*, 23(5):1001–1018, 1994.
- [19] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
- [20] M. Horstein. Sequential transmission using noiseless feedback. *IEEE Trans. Inf. Theor.*, 9(3):136–143, 1963.
- [21] R. Impagliazzo and R. Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62:367–375, 2001.
- [22] A. V. Iyer, H. D. Ratliff, and G. Vijayan. Optimal node ranking of trees. *Information Processing Letters*, 28(5):225–229, 1988.
- [23] B. Jedynek, P. I. Frazier, and R. Sznitman. Twenty questions with noise: Bayes optimal policies for entropy loss. *Journal of Applied Probability*, 49(1):114–136, 2012.
- [24] C. Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.
- [25] R. M. Karp and R. Kleinberg. Noisy binary search and its applications. In *Proc. 18th ACM-SIAM Symp. on Discrete Algorithms*, pages 881–890, 2007.
- [26] D. J. Kleitman and J. Spencer. Families of k -independent sets. *Discrete Mathematics*, 6, 1973.
- [27] E. S. Laber, R. L. Milidiú, and A. A. Pessoa. On binary searching with non-uniform costs. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, pages 855–864, 2001.
- [28] T. W. Lam and F. L. Yue. Edge ranking of graphs is hard. *Discrete Applied Mathematics*, 85(1):71–86, 1998.
- [29] T. W. Lam and F. L. Yue. Optimal edge ranking of trees in linear time. *Algorithmica*, 30(1):12–33, 2001.
- [30] N. Linial and M. Saks. Searching ordered structures. *Journal of Algorithms*, 6(1):86–103, 1985.

- [31] S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, pages 1096–1105, 2008.
- [32] M. Naghshvar, T. Javidi, and K. Chaudhuri. Bayesian active learning with non-persistent noise. *IEEE Transactions on Information Theory*, 61(7):4080–4098, 2015.
- [33] R. Nowak. Noisy generalized binary search. In *Proc. 23rd Advances in Neural Information Processing Systems*, pages 1366–1374, 2009.
- [34] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *Proc. 47th IEEE Symp. on Foundations of Computer Science*, pages 379–388, 2006.
- [35] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [36] A. Pedrotti. Searching with a constant rate of malicious lies. In *Proceedings of the International Conference on Fun with Algorithms (FUN-98)*, pages 137–147, 1999.
- [37] A. Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.
- [38] A. Pelc. Searching games with errors — fifty years of coping with liars. *Theoretical Computer Science*, 270(1–2):71–109, 2002.
- [39] A. Rényi. On a problem of information theory. *MTA Mat.Kut.Int.Kozl.*, 6 B:505–516, 1961.
- [40] R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.
- [41] A. A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.
- [42] S. M. Ulam. *Adventures of a Mathematician*. 1991.
- [43] R. Waeber, P. I. Frazier, and S. G. Henderson. Bisection search with noisy responses. *SIAM Journal on Control and Optimization*, 51(3):2261–2279, 2013.
- [44] S. Yan, K. Chaudhuri, and T. Javidi. Active learning from noisy and abstention feedback. In *Allerton Conference on Communication, Control and Computing*, 2015.

A Hardness Proofs

Here, we supply the hardness proofs omitted from Section 4. We begin with the proof for Theorems 10 and 11. The proofs are quite similar, so we present them together.

First, recall (see [35, Chapter 17.2] for a more detailed discussion) that Σ_{2k-1} (here, we are interested only in odd classes) can be characterized as all problems that can be expressed as

$$\exists \mathbf{x}_1 \forall \mathbf{y}_1 \exists \mathbf{x}_2 \forall \mathbf{y}_2 \cdots \exists \mathbf{x}_k : F(\mathbf{x}_1, \mathbf{y}_1, \mathbf{x}_2, \mathbf{y}_2, \dots, \mathbf{x}_k),$$

where each $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,r})$, $\mathbf{y}_i = (y_{i,1}, \dots, y_{i,r})$ is a vector of r Boolean variables, and F is a predicate that can be evaluated in polynomial time. A canonical hard problem for Σ_{2k-1} , which we will use for the hardness proof here, is the problem QBF_k , in which we specifically choose $F(\mathbf{x}_1, \mathbf{y}_1, \dots, \mathbf{x}_k) = \bigwedge_{\ell=1}^m C_\ell$ as a CNF formula. Without loss of generality, we may assume that no clause contains both a variable and its negation, since such a clause is always satisfied, and can thus be pruned. When k is part of the input, QBF_k is equivalent to the well-known QBF problem — in that case, the problem is PSPACE-hard even when each $\mathbf{x}_i, \mathbf{y}_i$ is just a single Boolean variable.

QBF_k (and thus also QBF) can be considered as a two-player game in which the players take turns choosing values for the variables. During the i^{th} round, the first player assigns either true or false to each of the r variables in \mathbf{x}_i ; subsequently, the second player chooses Boolean values for the r variables in \mathbf{y}_i . The first player’s objective is to satisfy F while the second player wants F to become false. Using this interpretation, QBF_k can be rephrased as follows: “Is there a winning strategy for the first player?”

Proof of Theorem 10. We consider the adaptive query problem as a two-player game, in which the first player (whom we call the vertex player) queries r vertices at a time, while the second player (the edge player), for each queried vertex q , chooses an outgoing edge from q lying on a shortest path from q to the target. The vertex player wins if after k rounds, he can uniquely identify the target vertex based on the responses, while the edge player wins if after k rounds of r queries each, there is still more than one potential target vertex consistent with all answers.

To prove membership in Σ_{2k-1} (and thus also in PSPACE when k is part of the input), first notice that the vertex player’s choice of r nodes to query can be encoded in $r \log(n)$ bits, i.e., Boolean variables, as can the edge player’s response of r edges. Membership in Σ_{2k} would now be obvious, as we could quantify over all of the edge player’s responses, and given all responses, it is obvious how to decide whether the target is uniquely identified.

To prove the stronger bound of Σ_{2k-1} , consider the following decision problem: given all queries, as well as possibly responses to some of them, can the edge player respond to the remaining queries while keeping at least two candidate targets? This question can be decided as follows. The edge player enumerates all target pairs $\{t_1, t_2\}$. For each query independently, she checks if there is a response that would be consistent with both t_1 and t_2 . If such responses exist, then the edge player can ensure that the vertex player cannot differentiate between t_1 and t_2 ; otherwise, the vertex player will succeed. By exhaustively checking all pairs, the edge player tries all possible winning strategies.

Let $\hat{k} = k - 2$. To prove hardness, we reduce from $\text{QBF}_{2\hat{k}-1}$, i.e., we let \hat{k} be the number of \exists quantifiers in the formula. For the PSPACE-hardness proof, we only need each $\mathbf{x}_i, \mathbf{y}_i$ to be a single variable.

Given an instance of $\text{QBF}_{2\hat{k}-1}$, in which each $\mathbf{x}_i, \mathbf{y}_i$ has (without loss of generality) exactly r Boolean variables ($r = 1$ for the PSPACE-hardness reduction), we construct an unweighted strongly connected graph $G = (V, E)$ with the following pieces, illustrated in Figure 1.

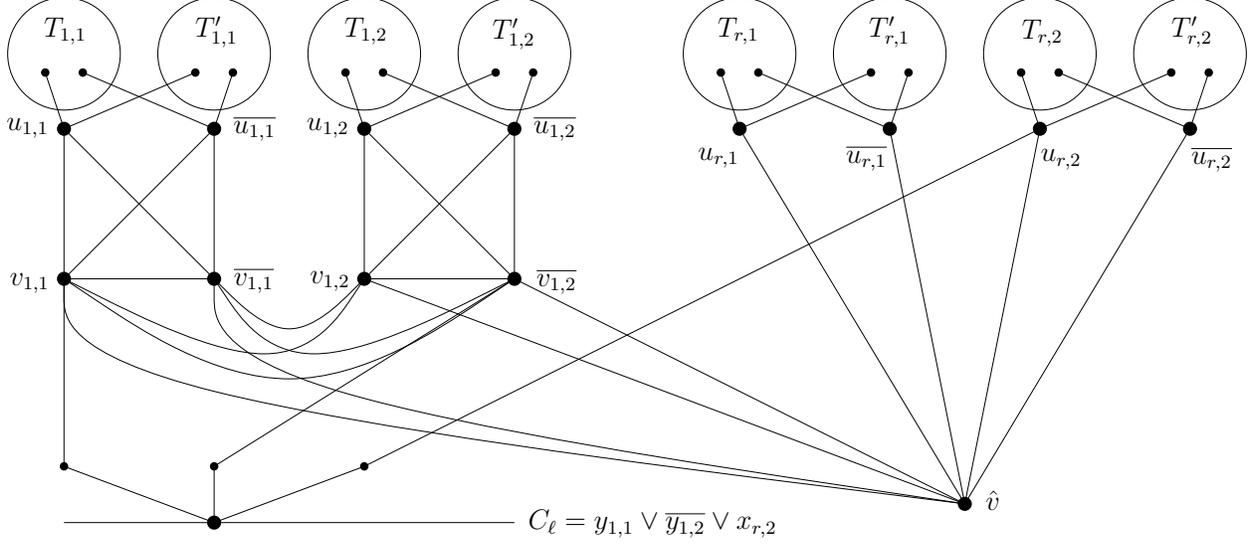


Figure 1: A schematic depiction of the graphs produced by the reduction.

- For each variable $x_{i,j}$, add two *literal vertices of type I*, named $u_{i,j}$ and $\overline{u_{i,j}}$. Similarly, corresponding to each variable $y_{i,j}$, add two *literal vertices of type II*, named $v_{i,j}$ and $\overline{v_{i,j}}$. For every $1 \leq i < \hat{k}$ and $1 \leq j \leq r$, add undirected edges from both $u_{i,j}$ and $\overline{u_{i,j}}$ to both $v_{i,j}$ and $\overline{v_{i,j}}$. Add undirected edges between each pair of literal vertices of type II.

Furthermore, add an extra vertex \hat{v} . Add undirected edges between \hat{v} and each of the literal vertices of type II, as well as between \hat{v} and both $u_{\hat{k},j}$ and $\overline{u_{\hat{k},j}}$ for all $1 \leq j \leq r$. For the PSPACE-hardness reduction, also add directed edges from \hat{v} to both $u_{i,j}$ and $\overline{u_{i,j}}$ for $i < \hat{k}$; however, for the $\Sigma_{2\hat{k}-1}$ -hardness, there is no edge between \hat{v} and $u_{i,j}$ or $\overline{u_{i,j}}$ for $i < \hat{k}$.

- For each $1 \leq i \leq \hat{k}, 1 \leq j \leq r$, add two *critical gadgets* $T_{i,j}$ and $T'_{i,j}$ that will be specified momentarily. Choose two arbitrary distinct vertices in each critical gadget as *critical nodes*. Let $t_{i,j}$ and $\overline{t_{i,j}}$ be critical nodes of $T_{i,j}$, and $t'_{i,j}$ and $\overline{t'_{i,j}}$ be in $T'_{i,j}$. Add undirected edges between $u_{i,j}$ and both $t_{i,j}$ and $t'_{i,j}$, as well as between $\overline{u_{i,j}}$ and both $\overline{t_{i,j}}$ and $\overline{t'_{i,j}}$.

For the PSPACE-hardness reduction, the i^{th} critical gadgets are directed cycles of length $\hat{k} + 3 - i$; for the $\Sigma_{2\hat{k}-1}$ -hardness proof, they are undirected paths of length $2(r+1)^{\hat{k}+2-i} - 1$.

- Corresponding to each clause C_ℓ in F , add two *clause gadgets* P_ℓ and P'_ℓ . Each clause gadget is an undirected path of length $2(r+1)^2 - 1$. (For the PSPACE-hardness reduction, where $r = 1$, the path has length 7.) Let p_ℓ and p'_ℓ be vertex number $2(r+1)$ on the respective paths. For the case $r = 1$, this is the midpoint of the path, and more generally, the leftmost of the set of vertices dividing the path into $r+1$ equal-sized subpaths.
- Whenever a literal $(x_{i,j}, \overline{x_{i,j}}, y_{i,j}, \overline{y_{i,j}})$ appears in clause C_ℓ , add two *intermediate nodes* specific to this pair of a literal and clause. Add undirected edges between the intermediate nodes and the corresponding literal vertex $(u_{i,j}, \overline{u_{i,j}}, v_{i,j}, \overline{v_{i,j}})$ and add an undirected edge between the first intermediate node and p_ℓ , and between the second intermediate node and p'_ℓ . (The intermediate nodes connect the corresponding literal vertex to both p_ℓ and p'_ℓ via paths of length 2.) See Figure 1.

Recall that we set the total number of rounds to be $k = \hat{k} + 2$. The vertex player is allowed to query r vertices in each round. Notice that the graph we construct is strongly connected, because all the critical gadgets and clause gadgets are strongly connected. Moreover, although there are some directed edges in the construction for PSPACE-hardness, all the edges are undirected for the $\Sigma_{2\hat{k}-1}$ -hardness reduction.

Furthermore, while the size of the critical gadgets for the $\Sigma_{2\hat{k}-1}$ -hardness reduction grows exponentially in k , it is polynomial for constant \hat{k} (and therefore, k). The reduction thus always takes polynomial time. We begin with a simple lemma elucidating the role of the critical gadgets and clause gadgets.

- Lemma 12 (Gadgets)**
1. *Conditioned on knowing that the target is in $T_{i,j}$ (or $T'_{i,j}$), and nothing else, $\hat{k} + 2 - i = k - i$ rounds of r queries are necessary and sufficient to find the target in the worst case.*
 2. *Conditioned on knowing that the target is in a specific clause gadget P_ℓ or P'_ℓ , but nothing more, 2 more rounds of queries are necessary. The same number is also sufficient to identify the target uniquely, even when the target may be in an intermediate node adjacent to the clause gadget.*

Proof. We begin with the first part of the lemma: For the directed cycle, until all but one of the vertices have been queried, there are always at least two candidates; querying all but one vertex is also clearly sufficient.

For a path of length M , querying r nodes can at most eliminate those nodes, and results at best in $r + 1$ subpaths of length $(M - r)/(r + 1)$ each. Spacing the query nodes evenly matches this bound. A simple proof by induction now establishes the bound.

The proof of the second part is analogous: The length of the path is chosen to require exactly 2 rounds of r queries. The choice of the vertices p_ℓ and p'_ℓ ensures that if the target was at an intermediate vertex, this fact is revealed in the first round of querying. ■

We claim that the first player in the QBF game has a winning strategy if and only if the vertex player can find any target in G using at most k rounds of queries.

(1) First, assume that there exists a strategy for finding the target in G using no more than k rounds of queries.

Let Q_i be the set of vertices queried by the vertex player in the i^{th} round. We claim the following, for each $i \leq \hat{k}$: if $Q_{i'} \subseteq \{u_{i',j}, \overline{u_{i',j}} \mid j = 1, \dots, r\}$ for each $i' < i$, and all of the second player's responses were toward either $v_{i',j}$ or $\overline{v_{i',j}}$, then $Q_i \subseteq \{u_{i,j}, \overline{u_{i,j}} \mid j = 1, \dots, r\}$, and furthermore, Q_i contains exactly one of $\{u_{i,j}, \overline{u_{i,j}}\}$ for each j . The reason is that under the assumption about prior queries, all critical gadgets $T_{i,j}$ and $T'_{i,j}$ are still in S . Having already used $i - 1$ rounds previously, the vertex player has only $k - (i - 1)$ rounds left, and $k - i$ rounds are necessary in order to find a target in one of these critical gadgets, by the first part of Lemma 12. Since the edge player can also choose which of the $2r$ critical gadgets contains the target, the vertex player has to identify the correct gadget using at most one round, which is only accomplished by choosing one of $u_{i,j}, \overline{u_{i,j}}$ for each j .

We now define the following mapping from the vertex player's strategy to a winning strategy for the first player in the formula game. When the vertex player queries $u_{i,j}$, the first player assigns $x_{i,j} = \text{true}$, whereas when the vertex player queries $\overline{u_{i,j}}$, the first player sets $x_{i,j} = \text{false}$. For $i < \hat{k}$, in response to the first player's setting of $x_{i,j}$, the second player will set $y_{i,j}$ either true or false. If the second player sets $y_{i,j} = \text{true}$, then we have the edge player reveal $\overline{v_{i,j}}$, whereas when

$y_{i,j} = \text{false}$, the edge player reveals $v_{i,j}$ instead. By the previous claim, if $i < \hat{k}$, the vertex player's next queries must be to either $u_{i+1,j}$ or $\overline{u_{i+1,j}}$, meaning that we can next set all of the $x_{i+1,j}$ by the same procedure. For the k^{th} round of queries, we make the edge player reveal the edge toward \hat{v} . We thus obtain a variable assignment to all $x_{i,j}$ and $y_{i,j}$, and it remains to show that it satisfies all clauses C_ℓ .

By assumption, having used $\hat{k} = k - 2$ rounds of queries, the vertex player can always identify the target with the remaining 2 rounds. Note that for each matching pair (P_ℓ, P'_ℓ) of clause gadgets, both P_ℓ and P'_ℓ are entirely in the candidate set, or both have been completely ruled out. This is because all responses were pointing to $v_{i,j}$ or $\overline{v_{i,j}}$, and were thus symmetric for both P_ℓ and P'_ℓ . By the second part of Lemma 12, it would take at least 2 rounds of queries to identify a node in a known clause gadget, and one more to identify the correct gadget. This is too many rounds of queries, so no clause gadgets can be in S , and all clause gadget instances must have been ruled out previously.

This means that for each clause C_ℓ , there must have been $1 \leq i \leq \hat{k}$ such that the responses to the i^{th} query ruled out the target being in either P_ℓ or P'_ℓ . Suppose that the queried set Q_i contained a node $q_{i,j} \in \{u_{i,j}, \overline{u_{i,j}}\}$, and $v_{i,j}$ or $\overline{v_{i,j}}$ (or \hat{v} if $i = \hat{k}$) is the answer. Then, the clause gadgets could have been ruled out in one of two ways:

- $q_{i,j}$ is connected to P_ℓ and P'_ℓ via intermediate nodes, i.e., there is a path of length 2 from $q_{i,j}$ to p_ℓ and also p'_ℓ . These clause gadgets have been ruled out because the answer of the query was not toward one of the intermediate nodes (connected to p_ℓ or p'_ℓ). In this case, by definition, the literal corresponding to $q_{i,j}$ (either $x_{i,j}$ or $\overline{x_{i,j}}$) is in C_ℓ , and because $q_{i,j}$ was queried, the literal is set to true. Thus, C_ℓ is satisfied under the assignment.
- An intermediate node connects $v_{i,j}$ to P_ℓ (and P'_ℓ) and the edge player chose $\overline{v_{i,j}}$, or — symmetrically — an intermediate node connects $\overline{v_{i,j}}$ to P_ℓ (and P'_ℓ) and the edge player chose $v_{i,j}$. Without loss of generality, assume the first case. In that case, by definition, $y_{i,j} = \text{true}$ (because $\overline{v_{i,j}}$ was chosen), and by construction of the graph, $y_{i,j} \in C_\ell$. Again, this ensures that C_ℓ is satisfied under the assignment.

In summary, we have shown that all clauses are satisfied, meaning that the first player in the formula game has won the game.

(2) For the converse direction, assume that the first player has a winning strategy in the formula game. We will use this strategy to construct a winning strategy for the vertex player in the target search game.

We begin by considering a round $i \leq \hat{k}$ in which the vertex player needs to make a choice. Assume that for each round $i' < i$, $Q_{i'} \subseteq \{u_{i',j}, \overline{u_{i',j}} \mid j = 1, \dots, r\}$, and furthermore, $Q_{i'}$ contains exactly one of $\{u_{i',j}, \overline{u_{i',j}}\}$ for each j . Also assume that the edge player responded with either $v_{i',j}$ or $\overline{v_{i',j}}$ for each query of $u_{i',j}$ or $\overline{u_{i',j}}$. (We will see momentarily that these assumptions are warranted.) Interpret an answer pointing to $v_{i',j}$ as setting $y_{i',j} = \text{false}$, and an edge pointing to $\overline{v_{i',j}}$ as setting $y_{i',j} = \text{true}$. Consider the choice for $x_{i,j}$ prescribed by the first player's assumed winning strategy, based on the history so far. The vertex player will query $u_{i,j}$ if $x_{i,j} = \text{true}$, and $\overline{u_{i,j}}$ if $x_{i,j} = \text{false}$. We distinguish several cases, based on the response:

- If one of the queried vertices is the target, then clearly, the vertex player has won.
- If the edge player's response is toward an instance of critical gadgets, then the target is known to lie in that critical gadget. By Lemma 12, there exists a query strategy for T_i (or T'_i) which can find the target using at most $k - i$ rounds of r queries. Together with the i rounds of

queries already used by the vertex player, this gives a successful identification with at most k rounds of queries total.

- If the answer is toward an intermediate node connected to one of the queried nodes, then the target must lie in the corresponding clause gadget, say P_ℓ , or is one of the intermediate nodes connected to this gadget. By Lemma 12, it takes at most 2 more rounds of queries to identify the target; together with the first i rounds of queries, this is a successful identification with at most k rounds of queries.
- This leaves the case when the edge player chooses the edge toward $v_{i,j}$ or $\overline{v_{i,j}}$ (or \hat{v} , if $i = \hat{k}$), justifying the assumption made earlier that for each of the first i rounds of queries, the edge player responds by revealing edges toward either $v_{i,j}$ or $\overline{v_{i,j}}$.

In summary, the fact that the assignment satisfies all C_ℓ implies that the target cannot lie on any clause gadget. The fact that the edge player responded with edges toward $v_{i,j}$ or $\overline{v_{i,j}}$ to the first \hat{k} rounds of queries implies that the target cannot lie on any critical gadgets, either.

The remaining case is the \hat{k}^{th} round of queries, when the edge player's response may contain edges toward \hat{v} . Then, the only candidate nodes remaining after k rounds of queries are (some of) the literal vertices, (some of) the intermediate nodes and \hat{v} . In this case, 2 more rounds are sufficient for both of the reductions, as follows. For the PSPACE-hardness reduction, query \hat{v} , and subsequently query the vertex $u_{i,j}$, $\overline{u_{i,j}}$, $v_{i,j}$ or $\overline{v_{i,j}}$ with which the edge player responds. (Recall that there are edges from \hat{v} to all the literal vertices.) That query either reveals the target, or points to an intermediate vertex which is then known to be the target. For the $\Sigma_{2\hat{k}-1}$ -hardness reduction, \hat{v} is queried in the first extra round. Next, the vertex player simultaneously queries whichever of $v_{i,j}$, $\overline{v_{i,j}}$ the edge player responded with, as well as the node of $u_{i,j}$, $\overline{u_{i,j}}$ that he has not queried yet. This will either reveal the target in one of the queried nodes, or reveal an edge to an intermediate vertex which is then known to be the target.

Finally, notice that the number of rounds was chosen to be $k = \hat{k} + 2$; expressing the hardness result in terms of the number of rounds of the target search game implies the claimed Σ_{2k-5} -hardness for the semi-adaptive version with k rounds. We believe that a somewhat more complex construction and argument improves this bound to Σ_{2k-3} -hardness. ■

Proof of Theorem 11. We only detail the changes in the gadgets of Theorem 10 here. First, we set an upper bound of $K = \hat{k} + 3$ on the cost, instead of the bound $k = \hat{k} + 2$ on the number of rounds. Recall that for critical gadgets T_i and T'_i , we only need the following property: $K - i$ queries are necessary and sufficient in the worst case to find the target in each of them. We therefore let each T_i and T'_i consist of two vertices connected with an undirected edge. For each of these nodes, the query cost is $K - i$. The new critical gadgets still satisfy Lemma 12, which was all that the proof required.

Clause gadgets are replaced with paths of length 15 (instead of paths of length 7), meaning that instead of two queries, three queries are now necessary and sufficient to identify a target in a clause gadget. Also, p_ℓ and p'_ℓ are again the middle points of these paths. Finally, in the new construction, no directed edges are inserted from \hat{v} to the $u_{i,j}$, $\overline{u_{i,j}}$ for $i < \hat{k}$. The only difference is that now, when all critical and clause gadgets have been ruled out after \hat{k} queries, it takes 3 more queries to find the target. The rest of the proof stays the same as before.

It is easy to check that the diameter of the resulting graph is at most 13. ■

We now prove Theorem 9. Hardness results based on ETH and SETH have appeared in several recent works: for example, Braverman et al. [7] proved a quasipolynomial-time lower bound

for approximating the best Nash equilibrium, while Abboud et al. [1] and Bringmann et al. [8] rule out sub-quadratic algorithms for a family of classical string problems (e.g., Longest Common Subsequence).

Proof of Theorem 9. Let $\bigwedge_{\ell=1}^m C_\ell$ be an instance of CNF-SAT with n variables and m clauses (with m polynomial in n). Without loss of generality, assume that $n = k^2$ is a perfect square. (If it is not, we can add $O(\sqrt{n})$ dummy variables to make it a perfect square.) Partition the variables into k batches of k variables each, labeled $x_{j,i}$.

The overall construction idea is similar to the proofs of Theorems 10 and 11, but slightly easier. The (unweighted and undirected) graph looks as follows this time:

- For each batch j ($1 \leq j \leq k$), and each assignment $\mathbf{a} \in \{0, 1\}^k$, construct three vertices: an *assignment vertex* $v_{j,\mathbf{a}}$ and two *intermediate vertices* $u_{j,\mathbf{a}}, u'_{j,\mathbf{a}}$. Add edges between $v_{j,\mathbf{a}}$ and $u_{j,\mathbf{a}}$, and between $v_{j,\mathbf{a}}$ and $u'_{j,\mathbf{a}}$. Add two extra nodes \hat{v}, \hat{v}' , connected via an edge. Moreover, connect \hat{v} with all assignment vertices $v_{j,\mathbf{a}}$.
- For each batch j ($1 \leq j \leq k$), add two critical gadgets T_j and T'_j , each a simple path of length $2^{k-j+3} - 1$. Let t_j and t'_j be the middle points of T_j and T'_j , respectively. Connect t_j to the intermediate nodes $u_{j,\mathbf{a}}$ for all \mathbf{a} , and t'_j to $u'_{j,\mathbf{a}}$ for all \mathbf{a} . Hence, all assignment vertices are connected to the corresponding critical gadgets via paths of length two.
- Corresponding to each clause C_ℓ in the formula, add a clause gadget P_ℓ , which is a simple path of length 7. For each assignment vertex $v_{j,\mathbf{a}}$, if \mathbf{a} satisfies C_ℓ , add a new intermediate node $u''_{j,\mathbf{a},\ell}$, and connect it to both $v_{j,\mathbf{a}}$ and the middle node of P_ℓ .

The overall outline of the proof is similar to (but simpler than) the one of Theorems 10 and 11. The key idea is again that to have any chance of finding a target in the critical gadgets, an adaptive strategy must pick exactly one assignment vertex from each batch; otherwise, a target in a critical gadget could not be identified. This allows us to establish a one-to-one correspondence between adaptive strategies and variable assignments. Revealing an edge to a critical or clause gadget would give the adaptive strategy an easy winning option, so one can show that w.l.o.g., all responses are to \hat{v} . This rules out all clause gadgets for clauses satisfied by the \mathbf{a} for the queried vertex $v_{j,\mathbf{a}}$.

In order to succeed in the final two rounds with \hat{v}, \hat{v}' , a number of unqueried $v_{j,\mathbf{a}}$ and many $u''_{j,\mathbf{a},\ell}$ still remaining, an algorithm must have eliminated all of the clause gadgets from consideration, which is accomplished only when all clauses are satisfied. (Conversely, if all critical and clause gadgets have been eliminated, the algorithm can next query \hat{v} and the $v_{j,\mathbf{a}}$ that is returned as the response.) Hence, a satisfying variable assignment exists if and only if $k + 2$ queries are sufficient, as captured by the following lemma:

Lemma 13 *There exists an adaptive strategy to find the target in the constructed graph within at most $k + 2$ queries if and only if the CNF formula is satisfiable.*

The constructed graph has $N = O(mk2^k)$ vertices and $M = O(mk2^k)$ edges; thus $\log N = k + o(k)$. Assume that some algorithm \mathcal{A} decides whether there exists any adaptive strategy to find the target with $k + 2$ queries. We would obtain the following complexity-theoretic consequences:

1. If the formula is a 3-CNF-SAT formula, and the running time of \mathcal{A} is $M^{o(\log N)} = M^{o(k)}$, then the reduction would give us an algorithm for 3-CNF-SAT with running time $M^{o(k)} = 2^{o(n)}$, which contradicts the ETH.

2. For general CNF-SAT instances, if the running time of \mathcal{A} is $O(M^{(1-\epsilon)\log N})$, then the above reduction would solve CNF-SAT with running time

$$O((m\sqrt{n}2^{\sqrt{n}})^{(1-\epsilon)\sqrt{n}}) = O(2^{(1-\epsilon/2)n}),$$

contradicting the SETH. ■

B Almost Undirected Graphs

We consider the generalization of the problem to (strongly connected) directed graphs. The example of a directed cycle shows that one can, in general, not hope to find a target using a sublinear number of queries. Thus, in order to achieve positive results, additional assumptions must be placed on the graph structure. Indeed, we saw in Section 4 that deciding, for general strongly connected graphs, whether a given number of queries is sufficient is PSPACE-complete.

We show that if the graph is “almost undirected,” then the positive result of Theorem 3 degrades gracefully. Specifically, we assume that each edge e with weight ω_e is part of a cycle of total weight at most $c \cdot \omega_e$. Notice that for unweighted graphs, this means that each edge e is part of a cycle of at most c edges, and specifically for $c = 2$, the graph is undirected.⁸

Theorem 14 *Algorithm 1 has the following property: if G is a strongly connected, positively weighted graph, in which each edge e belongs to a cycle of total weight at most $c \cdot \omega_e$, then the algorithm finds the target using at most*

$$\frac{1}{\ln c - \ln(c-1)} \cdot \ln n \leq c \ln(2) \cdot \log n$$

queries in the worst case.

Proof. In Algorithm 1, the potential is now defined with respect to directed distances. To analyze its performance, assume that the algorithm, in some iteration, queried the node q , and received as a response an edge $e = (q, v)$. We define the sets S^+ and S^- as in the proof of Theorem 3. As before, $d(v, u) = d(q, u) - \omega_e$ for all vertices $u \in S^+$. On the other hand, there exists a path of total weight at most $(c-1) \cdot \omega_e$ from v to q (since e appears in a cycle of length at most $c \cdot \omega_e$). Thus, for any vertex $u \in S^-$, $d(v, u) \leq d(q, u) + (c-1) \cdot \omega_e$. Therefore,

$$\Phi_S(v) \leq \Phi_S(q) - \omega_e \cdot (|S^+| - (c-1) \cdot |S^-|).$$

Since $\Phi_S(q)$ is minimal, $|S^+| \leq (c-1) \cdot |S^-|$, so $|S^+| \leq \frac{c-1}{c} \cdot |S|$. Thus, after at most $\log_{c/(c-1)}(n) = \frac{\ln n}{\ln c - \ln(c-1)}$ queries, the target must be identified. Finally,

$$\ln(c) - \ln(c-1) = \int_{c-1}^c \frac{dx}{x} \geq 1/c,$$

implying that $\frac{\ln n}{\ln c - \ln(c-1)} \leq c \cdot \ln n = c \ln(2) \cdot \log n$. ■

⁸However, for weighted graphs with $c = 2$, G will not necessarily be undirected.

The upper bound of Theorem 14 is nearly matched, up to a factor of $O(\log c)$, by the following lower bound.

Proposition 15 *For any integers N and $c \geq 2$, there exists an unweighted and strongly connected graph G of $n \geq N$ vertices, such that each edge in G belongs to a cycle of length c , and at least $\frac{c-1}{\log c} \cdot \log n$ queries are required to identify a target in G .*

Proof. We construct a family of unweighted strongly connected directed graphs $H_{c,k}$, $k = 0, 1, \dots$ inductively. $H_{c,0}$ is a single vertex. For any $k \geq 1$, $H_{c,k}$ is obtained from c disjoint copies of $H_{c,k-1}$. For each of the c copies $i = 1, \dots, c$, let v_i be an arbitrary vertex in that copy. Now add a directed cycle of length c through the vertices v_i . By construction, each edge is part of a cycle of length c ; and by induction, $H_{c,k}$ is strongly connected for all k .

We will prove by induction that any strategy requires at least $(c-1) \cdot k$ queries in the worst case to identify the target in $H_{c,k}$. Because $n = c^k$, this proves a lower bound of $\frac{c-1}{\log c} \cdot \log n$ on the number of queries in an n -vertex graph.

The base case $k = 0$ of the induction is trivial. For the induction step, let G_1, G_2, \dots, G_c be the c copies of $H_{c,k-1}$ that were combined to form $H_{c,k}$. For $i < k$, define v'_i to be v_{i+1} and let v'_k be v_1 . Consider a query of a node $q \in G_i$. By construction, v'_i lies on any path from q to any vertex not in G_i . In other words, if an algorithm receives an edge (q, v) lying on a shortest path from q to v'_i , it might learn that the target is not in G_i , but cannot infer which G_j , $j \neq i$ the target is in.

The adversary's strategy is now simple: for the first $c-1$ queries, to each query $q \in G_i$, the adversary will give an edge toward v'_i , i.e., the first edge of a shortest path from q to v'_i . At this point, there is at least one G_i such that no vertex in G_i has been queried. The adversary now picks one such G_i (arbitrarily, if there are multiple), and commits the target to G_i . Then, he continues to answer queries to vertices in G_j , $j \neq i$ in the same way as before. Queries to vertices in G_i are answered with the adversarial strategy for $H_{c,k-1}$.

As a result, after $c-1$ queries, there will always remain at least one entirely unqueried copy of $H_{c,k-1}$; the algorithm has no information about the location of the target vertex in this copy, and by induction hypothesis, it will take at least $(c-1) \cdot (k-1)$ queries to find the target. Adding the $c-1$ queries to reach this point gives us a total of at least $(c-1) \cdot k$ queries to find the target in $H_{c,k}$. ■

C More Informative Queries on Directed Graphs

Instead of restricting cycle lengths in G , an alternative way to obtain positive results for directed graphs is to assume that the responses to queries are more informative. This approach is motivated by the directed cycle, where the answer to a query reveals no additional information to the algorithm. If the algorithm could learn not only the outgoing edge but also the distance to the target, then a single query would suffice on the cycle. Indeed, we show that in general, this information is enough to always find the target using at most $\log n$ queries. We remark that learning *only* the distance to the target would not be enough to guarantee even a sublinear number of queries: in an unweighted undirected star, when the target is a leaf, such an answer will reveal no information except whether the queried node is the target.

Formally, we define more informative responses to queries as follows. Let $N(u, e, \ell) = \{v \in N(u, e) \mid d(u, v) = \ell\}$. In response to querying node q , the algorithm will be given an edge e and distance ℓ such that $t \in N(q, e, \ell)$.

Theorem 16 *Assume that each query reveals the distance from the queried node q to the target t as well as an edge $e = (q, v)$ on a shortest q - t path. Then, there is an efficient algorithm which, for each directed, strongly connected and positively weighted graph G , finds the target using at most $\log n$ queries.⁹*

Proof. The algorithm is nearly identical to Algorithm 1. It also queries a vertex q minimizing a potential function of the set S of remaining candidate vertices at each iteration. However, the potential function takes a different form.

For each distance ℓ , let $\Gamma_\ell(v)$ be the set of vertices at distance at most ℓ from v . Define $\Psi_S(v)$ as the minimum distance ℓ such that strictly more than half of the vertices of S are in $\Gamma_\ell(v)$. When receiving an answer of (e, ℓ) in response to a vertex query q , the algorithm updates S to $S \cap N(q, e, \ell)$. Except for using $\Psi_S(v)$ in place of $\Phi_S(v)$ in Line 3, receiving the distance ℓ in addition to the edge e in Line 7, and updating $S \leftarrow S \cap N(q, e, \ell)$ in Line 8, the algorithm is identical to Algorithm 1.

Similar to the analysis of Algorithm 1, we show that the size of S decreases by at least a factor of 2 in each iteration. Consider one iteration in which a vertex q is queried, and the response is an edge $e = (q, v)$ and distance ℓ . We partition S into the remaining candidates $S^+ = S \cap N(q, e, \ell)$, and the eliminated vertices $S^- = S \setminus S^+$.

Assume for contradiction that $|S^+| > |S|/2$. Because more than half of the vertices of S are thus at distance *exactly* ℓ from q , we obtain that $\Psi_S(q) = \ell$. And because v lies on a shortest q - u path for any $u \in S^+$, the distance from v to each $u \in S^+$ is strictly less than ℓ . Thus, more than $\frac{1}{2}|S|$ vertices of S are at distance less than ℓ from v , implying that $\Psi_S(v) < \ell$. But this contradicts the choice of q as minimizing $\Psi_S(v)$. ■

D Edge Queries on Trees

As mentioned in the introduction, a version of the problem that has been studied more frequently [30, 29, 3, 4, 34, 31], is one in which the algorithm is allowed to query the *edges* of a graph (most often: a tree). For the case of trees, by querying an edge $e = (u, v)$, the algorithm learns whether the target is in the subtree rooted at u or the one rooted at v .

Ben Asher and Farchi [3] proved, for a tree with maximum degree Δ , an upper bound of $\log_{\Delta/(\Delta-1)}(n) = \Theta(\Delta \log n)$ and a lower bound of $\frac{\Delta-1}{\log \Delta} \log n$ on the number of queries required, which — contrary to the authors' claim — leaves a small gap of $\Theta(\log \Delta)$. In this section, we prove that the lower bound is practically tight:

Theorem 17 *For any tree T with maximum degree $\Delta > 1$, there is an adaptive algorithm for finding the target using at most $1 + \frac{\Delta-1}{\log(\Delta+1)-1} \log n$ edge queries.*

Proof. The algorithm is quite similar to the algorithm for vertex queries in Section 2, in that it repeatedly queries a median node v . However, since a vertex in the given tree cannot be queried directly, the algorithm instead finds a median node v and queries the edge (v, u) , where u is the neighbor of v maximizing the size of its rooted subtree.

More precisely, the algorithm is as follows: Given a tree T , the algorithm initializes $T' = T$. In each iteration, it considers a separator node v (a vertex such that each subtree (in fact, connected component) of $T' \setminus \{v\}$ contains at most $|T'|/2$ vertices), breaking ties in favor of the separator node from the previous iteration. That is, if the last chosen separator is still a valid separator for

⁹This upper bound is tight even for complete binary trees.

the current tree T' , the algorithm uses it for the next iteration; otherwise, it picks a new separator node arbitrarily. Among all neighbors of v , let u be one maximizing the size of the subtree T_u of T' rooted at u . The algorithm then queries the edge (v, u) , and depending on whether the target is in the subtree rooted at v or at u , updates T' to either $T' \setminus T_u$ or T_u . When only a single vertex remains, the algorithm returns it. More formally, the algorithm is given as Algorithm 4.

Algorithm 4 TARGET SEARCH USING EDGE QUERIES (T)

```

1:  $T' \leftarrow T$ .
2:  $v \leftarrow$  a separator node of  $T'$ : each subtree of  $T' \setminus \{v\}$  contains at most  $|T'|/2$  vertices.
3: while  $|T'| > 1$  do
4:   if  $v$  is not in  $T'$  any more or it is not a separator for  $T'$ . then
5:      $v \leftarrow$  a separator for  $T'$ .
6:   Let  $u_1, u_2, \dots$  be the neighbors of  $v$  in  $T'$ , and  $T_1, T_2, \dots$  the subtrees of  $T'$  rooted at  $u_1, u_2, \dots$ 
7:    $i \leftarrow \underset{j}{\operatorname{argmax}}\{|T_j|\}$ .
8:   Query the edge  $(v, u_i)$ .
9:   if the target is in  $T_i$  then
10:     $T' \leftarrow T_i$ .
11:  else
12:     $T' \leftarrow T \setminus T_i$ .
13: return the unique vertex in  $T'$ .

```

First off, recall that the existence of a separator node was proved by Jordan [24]. For the purpose of analysis, we divide the execution of the algorithm into phases: maximal intervals during which the same vertex v is used as a separator. Notice that there may be multiple phases during which the same vertex is chosen — however, no two such phases can be adjacent by maximality.

Consider one phase and its corresponding vertex v ; suppose that v was used k times, and had degree d in T' at the beginning of the phase. Let u_1, u_2, \dots, u_d be the neighbors of v in T' , with subtrees T_1, T_2, \dots, T_d . Without loss of generality, assume that $|T_1| \geq |T_2| \geq \dots \geq |T_d|$, so that during those k queries, the edges $(v, u_1), (v, u_2), \dots, (v, u_k)$ were queried. If the phase is the last phase, and at some point, the tree only contains two vertices (and one edge), we treat the last step, determining the target with one more query, separately. We now distinguish three cases, based on how the phase ended:

1. The k^{th} query revealed that the target is in T_k , so that v was permanently removed from T' (along with all T_i for $i \neq k$). By the ordering of sizes, we have $|T_i| \geq |T_k|$ for all $i < k$. Furthermore, because v is a separator at the k^{th} query of the phase, $1 + \sum_{i=k+1}^d |T_i| \geq |T_k|$. Thus, $|T'| \geq (k+1)|T_k|$ where $|T'|$ is the size of the tree at the beginning of the phase; in other words, using k queries, the size of the remaining tree was reduced by a factor at least $k+1$.
2. The k^{th} query did not place the target in T_k , but the removal of T_1, \dots, T_k resulted in v not being a separator any more. By the ordering of sizes, we have $|T_i| \geq |T_{k+1}|$ for all $i \leq k$. Let $r = 1 + \sum_{i \geq k+1} |T_i|$ be the size of the remaining tree after the removal of T_1, \dots, T_k . Because v is not a separator any more, $r < 2|T_{k+1}|$, while $|T'| \geq k|T_{k+1}| + r$, where $|T'|$ is the size of the tree at the beginning of the phase. Thus, using k queries, the size of the remaining tree was reduced by a factor at least $\frac{k|T_{k+1}| + r}{r} \geq \frac{k+2}{2}$.

3. T' consisted of a single node (the algorithm is done), or two nodes with one edge (the algorithm is done with one more query).

Thus, in the first two cases, using k queries, the size of the remaining tree decreases by a factor at least $\frac{k+2}{2}$. Now, for each phase j , let k_j be the length of the phase. The total number of queries is $\sum_j k_j$, and the constraint imposed by the size decrease is that $\prod_j \frac{k_j+2}{2} \leq n$. By writing $\frac{k_j+2}{2} = e^{x_j}$, we obtain a convex objective function $\sum_j (2e^{x_j} - 2)$ subject to an upper bound $\sum_j x_j \leq \ln n$, which is maximized by making each x_j (and thus k_j) as large as possible or equal to 0. In other words, we obtain an upper bound by setting $k_j = \Delta - 1$ for all j . Notice that the only case in which the algorithm queries all d edges incident on a node is when there are exactly two nodes remaining before the final query; this case is treated separately.

The number of phases is then $\log_{(\Delta+1)/2} n = \frac{\log n}{\log(\Delta+1)-1}$. Thus, the total number of queries, counting the possible final query, is at most $1 + \frac{\Delta-1}{\log(\Delta+1)-1} \cdot \log n$, completing the proof. ■