Tree Search With Predictions

Bob Dong
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
bdong9@jh.edu

Michael Dinitz*
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
mdinitz@cs.jhu.edu

Abstract

"Algorithms with predictions", or "learning-augmented algorithms", has proved to be an extremely useful paradigm for combining machine learning with traditional algorithms. One of the textbook settings for this is searching a sorted array. Without a prediction, classical binary search takes $O(\log n)$ queries, while with a prediction we can use "doubling binary search" to find the target key using $O(\log \eta)$ queries, where η is the error of the prediction measured as the absolute value of the difference between the true location and the predicted location. Since an array is just a path graph, in this paper we ask whether similar bounds can be achieved for search on even slightly more general graphs: trees. We show first that the high-level answer is "no": there is no search algorithm that uses $O(\log \eta)$ queries, where η is now the graph distance between the predicted location and the true location. However, as our main result, we show that such bounds can be achieved on trees which are "path-like" in that they have low *pathwidth*. In particular, we prove that there is a search algorithm which uses at most $O(k \log \eta)$ queries, where k is the pathwidth of the tree. We also prove a lower bound showing that our algorithm has existentially optimal query complexity. Finally, we show experimentally, on real-life inputs, that our algorithm has query complexity which is notably better than the simple non-prediction-based algorithm.

1 Introduction

Traditional algorithms tend to optimize for the worst case, leading to extremely impressive worst-case performance but somewhat lacking performance in the usual case. On the other hand, approaches based on machine learning often have extremely impressive performance in the usual case (at least if the training data are close to the true data) while suffering from extremely poor worst-case behavior (e.g., if the training data comes from a completely different distribution than we see in our real application). An important framework that attempts to get the best of both is the "algorithms with predictions" framework, also called "learning-augmented algorithms" or "algorithms with machine-learned advice". In this framework we are given a problem-dependent "prediction" (possibly learned by a machine learning system), and want to do well if the prediction is accurate but also have good worst-case bounds if the prediction is extremely inaccurate. See Section 1.2 for more discussion.

A paradigmatic example of this framework is searching a sorted array. This problem is extremely simple and has an extremely simple solution, so it is a standard warm-up and motivation for this area; it is in fact the very first example considered in the survey of the area by Mitzenmacher and Vassilvitskii [2021]. In the basic search problem we are given an array A of n elements, each of which is from some totally ordered universe U, where A is in sorted order with respect to the ordering of U. We are also given a target element t, and the goal is to find which index of A contains t, i.e.,

*Funded in part by NSF award 2228995.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

find the value of i such that $A[i] = t$. Given t, let $\alpha(t)$ denote this value of i. The textbook solution to this is binary search, which finds $\alpha(t)$ in $O(\log n)$ queries.

But what if we are also given a *prediction* of the correct location, i.e., we are also given an index $\hat{\alpha}(t)$? Then it turns out that the folklore "doubling binary search" algorithm (originally proposed in an infinite search context by Bentley and Yao [1976]) achieves query complexity $O(\log \eta)$, where $\eta = |\alpha(t) - \hat{\alpha}(t)|$ is the error of the prediction. Note that $0 \leq \eta \leq n$, so this is never (asymptotically) worse than the non-prediction $O(\log n)$ bound and, if η is small, can be notably better. This type of bound, which degrades in a controlled way with the prediction error, is often called a "smoothness" guarantee, and is the gold standard for algorithms with predictions. This is in contrast, for example, to algorithms which only give tradeoffs between "consistency" (performance when the prediction error $\eta = 0$) and "robustness" (performance in the worst case when η is unbounded or as large as possible).

This framework has become extremely popular and is surprisingly powerful; see Section 1.2 for more discussion of related work. There is even significant work building directly from this binary search example by considering more complicated types of predictions [Dinitz et al., 2024]. However, one natural generalization of searching a sorted array has not yet been explored: searching objects more complicated than arrays.

To motivate this, note that an equivalent formulation of the search problem on a sorted array is a search problem on *paths*. In particular, suppose that we are given a path graph $P = (V, E)$ and are trying to find a particular target node $t \in V$, and are given a prediction $s \in V$. When we query some point $x \in V$, we are told which neighbor of x is on the unique path from x to t, i.e., we are told which direction on the path t is from x. Our goal is to find (query) t with as few queries as possible. It is trivial to see that this is *precisely* the problem of searching a sorted array. And interpreted in the language of graphs, we get that the doubling binary search algorithm has query complexity $O(\log \operatorname{dist}(s, t))$, where $\operatorname{dist}(s, t)$ denotes the graph distance between our prediction s and the true target node t.

But now by thinking of an array as a path graph, there is a natural generalization to other types of graphs. A particularly obvious generalization is to *trees*, since in trees it is always true that for any node x there is a unique neighbor of x that is on the (unique) path from x to t. So on trees we have the exact same problem, which we now define formally:

Definition 1.1. *In the* Search on Trees *problem we are given a tree $T = (V, E)$, and there is an unknown target node $t \in V$. In the prediction setting we are also given a prediction $s \in V$. We can query any node $x \in V$, and that query will return the unique neighbor of x that is on the path from x to t. The goal is to find (query) t using as few queries as possible.*

Without predictions, there is an obvious generalization of traditional binary search to trees, where instead of querying the median of the remaining interval we query the centroid of the remaining subtree. This obviously has query complexity $O(\log n)$, like in the path/array setting [Onak and Parys, 2006]. But what about the prediction setting? Is there an algorithm that, similar to the array setting, only requires $O(\log \operatorname{dist}(s, t))$ queries? If not, what query complexity can be achieved? Are there at least simple classes of trees where $O(\log \operatorname{dist}(s, t))$ queries is possible?

1.1 Our Results

In this paper we answer these questions. We first show that trees are fundamentally different than paths, in that no algorithm can achieve query complexity of $O(\log \operatorname{dist}(s, t))$. However, as our main result we show that trees which are "path-like" *do* allow for this type of query complexity. In particular, we design such an algorithm for trees with *bounded pathwidth*, a well-studied definition of how path-like a graph is. We give theoretical bounds on its query complexity, and also show experimentally on real life data that our theory matches reality: our algorithm significantly outperforms the traditional centroid-based algorithm when our predictions are reasonably accurate on low-pathwidth trees. Finally, we prove a matching lower bound which shows that our dependence on the pathwidth and on $\operatorname{dist}(s, t)$ is existentially optimal (no algorithm can have better dependence on all trees).

1.1.1 Initial Lower Bound

We begin with the bad news: when we generalize from paths to trees, it is no longer possible to get $O(\log \text{dist}(s,t))$ queries.

Theorem 1.2. *There is no algorithm (deterministic or randomized) for the Search on Trees problem that has expected query complexity $O(\log \text{dist}(s,t))$ for every prediction $s \in V$ and target $t \in V$.*

This theorem turns out to be quite simple to prove. Informally, consider the complete binary tree with n leaves. Then given a prediction s which is a leaf, at least $\Omega(n)$ other leaves are at least $\Omega(\log n)$ away from s. Distinguishing which is the true target must take at least $\Omega(\log n)$ queries for some target through a standard information-theoretic argument. So in this case the number of needed queries is at least $\Omega(\text{dist}(s,t))$, exponentially worse than our goal of $O(\log \text{dist}(s,t))$ and essentially matching the trivial algorithm of just following the query path. It turns out that this theorem is also a corollary of our stronger lower bound (Theorem 1.4), so rather than the direct proof sketched above we provide the proof as a corollary in Appendix A.

1.1.2 Bounded pathwidth: upper bound

Fortunately, we show that Theorem 1.2 is not the end of the story: many trees actually *do* admit algorithms with low query complexity. In particular, we show that trees with bounded *pathwidth* admit such algorithms. We define pathwidth formally in Section 2, but it is a standard way of quantifying how close a graph is to a path, in essentially the same way that treewidth measures how close a graph is to a tree. Paths (and slight extensions to paths like caterpillars) have pathwidth 1, while on the other extreme the complete binary tree has pathwidth $\Theta(\log n)$. Every other tree has pathwidth somewhere between these two extremes. Note that the pathwidth is well-defined for any graph, but we will only apply it to trees. This is not particularly unusual; there is significant previous work (which we draw on) studying the special properties of trees with low pathwidth (we will particularly use results from Suderman [2004]).

More formally, as our main result we prove the following theorem in Section 3.

Theorem 1.3. *There is a deterministic algorithm for the Search on Trees problem which, when given a tree $T = (V, E)$ of pathwidth k and a prediction $s \in V$, finds the unknown target $t \in V$ using at most $O(k \log \text{dist}(s,t))$ queries.*

To get some intuition for our algorithm, consider the special case of a *caterpillar* tree. Formally, a caterpillar is a tree in which there is a special path (usually called the *spine*) such that every node not in the spine is at distance 1 from the spine. So a caterpillar is *almost* a path, and indeed like a path has pathwidth 1. We claim that there is an obvious algorithm for solving the Search on Trees problem in a caterpillar. Given a prediction s, let s' be the closest node to s on the spine (so either s itself or the neighbor of s that is on the spine). Now run the classic doubling binary search algorithm for a path on the spine, starting from s'. If t is on the spine then this clearly takes at most $O(\log \text{dist}(s',t)) = O(\log \text{dist}(s,t))$ queries. If t is not on the spine, then in $O(\log \text{dist}(s',t) - 1)$ queries we will have queried the neighbor t' of t that is on the spine, which will then point us directly to t. So in either case, we use at most $O(\log \text{dist}(s,t))$ queries.

Clearly this can be extended to other related graphs. For example, we can clearly handle the (less standard) definition of a caterpillar where instead of every spine node being adjacent to a set of leaves, each spine node is adjacent to a set of arbitrarily long paths (i.e., deleting the spine results in a collection of disjoint paths). In these types of caterpillars we simply start from s' being the closest node on the spine to s (not necessarily a neighbor of s), and then when we find t' we run doubling binary search again on the path which contains t. In other words, the key feature that we need is a "spine" path on which we can run doubling binary search, and then we can recursively search whatever (hopefully simpler) tree the target node is located in off of the spine. It turns out that trees of bounded pathwidth are known to have a path with precisely these properties: it was shown by Suderman [2004] that any tree of pathwidth k has what they call a "main path" which, when deleted, leaves us with a disjoint collection of subtrees all of which have pathwidth at most $k-1$. So we can treat this main path as a spine, doing doubling binary search on it to figure out which subtree contains the target t, and then recursively search the resulting $k-1$-pathwidth subtree. Once properly formalized, it is not hard to see that we do at most k doubling binary searches, each of which takes at most $O(\log \text{dist}(s,t))$ queries, for a total query complexity of $O(k \log \text{dist}(s,t))$ as claimed.

Robustness. It is common in the algorithms with predictions literature to focus on *robustness*: if the prediction is arbitrarily bad, then we would like our algorithm to still be no worse (asymptotically) than the best algorithm without predictions. In many problems it is highly nontrivial to get both robustness and consistency (good performance when the prediction is exactly accurate). But we note that in the Search on Trees problem, we can get robustness "for free" by simply running the centroid algorithm in parallel with our algorithm. That is, we can run each algorithm separately, and stop whenever the first one finishes. This gives us a query complexity of $O(\min(\log n, k \log \mathrm{dist}(s,t)))$. So for the remainder of the paper we do not discuss robustness.

Running Time. We focus on query complexity rather than running time, following the lead of Mitzenmacher and Vassilvitskii [2021] and Dinitz et al. [2024]. There are multiple justifications for this. First, queries might be much more expensive than computation; think of a case where each query itself requires either a significant amount of computation, or something like a physical experiment (when we do search in, for example, a scientific discovery context). Second, our search algorithms can be computed *offline*: given the tree T, we can compute ahead of time the policy that we will use for any given prediction s. This takes time, of course, but can be done before seeing any queries. Then when we see queries, following the policy is as simple as following a search tree, and our query time is equal to our query complexity. So we can essentially trade increased "preprocessing time" for decreased "query time".

But even with these justifications, one might be skeptical of our algorithm if its running time were astronomical (e.g., exponential). Fortunately, our algorithm turns out to have polynomial running time, allowing us to run relatively large-scale experiments (see Section 4). The most time-intensive step is computing the hierarchy of spines, so in Appendix C we discuss explicitly how to do this step efficiently.

1.1.3 Pathwidth lower bound

A natural followup question to Theorem 1.3 is whether there are matching lower bounds. Most notably: do we actually need to have query complexity $\Omega(k \log \mathrm{dist}(s,t))$? Note that Theorem 1.2 does not give such a bound, since the complete binary tree has pathwidth $\Theta(\log n)$ and requires $\Theta(\log n)$ queries on pairs at distance $\Theta(\log n)$, and so it only implies lower bounds of $\Omega(k)$ (purely as a function of k) or $\Omega\left(\frac{k}{\log k} \log \mathrm{dist}(s,t)\right)$ (as a function of both k and $\log \mathrm{dist}(s,t)$).

We improve this lower bound by showing through a more complicated class of trees that our upper bound is tight: no algorithm can always have query complexity $o(k \log \mathrm{dist}(s,t))$, even if we allow randomized algorithms and expected query complexity.

Theorem 1.4. *There is no algorithm (deterministic or randomized) for the Search on Trees problem that has expected query complexity $o(k \log dist(s,t))$ for every tree $T = (V, E)$ of pathwidth k, prediction $s \in V$, and target $t \in V$.*

The details of this construction and proof can be found in Appendix A. While our construction is more complicated than the simple complete binary tree, it is intuitively just a "stretched" version of such a tree. The reason that the complete binary tree does not give such a lower bound is that each "spine" in the recursive call is just a single node (the root of the subtree), so we only need one query at every level rather than $\log \mathrm{dist}(s,t)$. To get our desired lower bound, we simply have to "stretch" each spine at each level to be longer. Carefully balancing parameters leads to Theorem 1.4.

1.2 Related Work

Algorithms with Predictions. The algorithms with predictions setting has become quite popular, so there is significant work in this framework for a variety of problems and models. We refer the interested reader to the early survey of Mitzenmacher and Vassilvitskii [2021] as well as the invaluable website which keeps track of the area [Lindermayr and Megow, 2022]. It is usually considered to have been initiated by the seminal work of Lykouris and Vassilvitskii [2021], who formally defined the setting and introduced the popular notions of robustness and consistency. Since then, it has been studied in settings as diverse as online algorithms (ski rental [Purohit et al., 2018], scheduling [Lattanzi et al., 2020], knapsack [Im et al., 2021], set cover [Bamas et al., 2020], and more), speeding up combinatorial algorithms [Dinitz et al., 2021], dynamic algorithms [van den Brand et al., 2024], mechanism design [Agrawal et al., 2024], and many more.

Most related to our paper is the survey of Mitzenmacher and Vassilvitskii [2021], which first introduced the binary search problem in the predictions context. Also related is the work of Dinitz et al. [2024], which generalized the binary search problem in a different way: rather than to more general graphs (like us), they stay in the path but allow for *distributional* predictions. Extending our results to distributional predictions is an interesting problem that we leave to future work.

Search in trees and graphs. As mentioned, there has been significant work extending binary search on paths to trees and even more general graphs in the non-predictions context. This was kicked off by Onak and Parys [2006], who among other results proved that the centroid algorithm (discussed earlier) has $O(\log n)$ query complexity. A further important generalization was to the "Search Trees on Trees" (STT) problem, where we are given a tree (like in our settings) and are given a distribution over the nodes in that tree, and are asked to compute an optimal search strategy (equivalently, an optimal search *tree*) for that tree and distribution. Note that this is like a "distributional prediction", except the assumption is that the given distribution is perfectly accurate. For example, Berendsohn et al. [2023] showed that the centroid algorithm (mentioned earlier) not only has $O(\log n)$ query complexity, but given a distribution over the nodes the natural weighted generalization is a 2-approximation of the *optimal* search strategy and can be computed extremely quickly. There has also been work on generalizing to graphs beyond trees [Emamjomeh-Zadeh et al., 2016].

Practical Motivations. Although our study of problem 1.1 is theoretical, the problem has a wide range of practical applications in data science. Several examples are discussed in Ben-Asher et al. [1999]. For instance, when large "tree-like" data structures are transmitted over noisy channels, algorithms for *Search on Trees* can be used to identify faulty nodes. In this setting, a learning system could predict the location of a fault to guide the search. Other applications include software testing, where search problems arise on partially ordered sets, and information retrieval.

2 Preliminaries and Notation

Given a tree T, we denote its vertex set by $V(T)$ and its edge set by $E(T)$. For vertices $u, v \in V(T)$, let $\mathrm{dist}_T(u, v)$ denote the length of the unique simple path between u and v in T. If P is a path in T and $u, v \in V(P)$, then $\mathrm{dist}_P(u, v)$ denotes the number of edges on the subpath of P between u and v.

We first formalize the oracle model used throughout the paper.

Definition 2.1 (Direction oracle)**.** *Let $T = (V(T), E(T))$ be a tree, and let $t \in V(T)$ be a hidden target vertex. The direction oracle for t is the map*

$$\mathrm{dir}_t : V(T) \to V(T) \cup \{\textsf{here}\}$$

defined as follows. If $v = t$, then $\mathrm{dir}_t(v) =$ here. Otherwise, $\mathrm{dir}_t(v)$ is the unique neighbor u of v that lies on the simple path from v to t in T. An oracle query at v returns $\mathrm{dir}_t(v)$.

In the search problem, the algorithm is given the tree T and a prediction $s \in V(T)$ for the target, but it does not know the target t. The algorithm may adaptively query vertices of T through the direction oracle dir_t. The algorithm succeeds when it outputs t; equivalently, it may stop once it queries a vertex v with $\mathrm{dir}_t(v) = \textsf{here}$. The query complexity of an algorithm is the number of oracle queries it makes. When the target is clear from context, we write dir instead of dir_t.

We will use the following standard path-search primitive.

Fact 2.2 (Exponential search on a path)**.** *Let P be a simple path, let $a \in V(P)$ be a starting vertex, and let $x \in V(P)$ be an unknown target vertex. Suppose that each query to a vertex $v \in V(P)$ returns* here *if $v = x$, and otherwise indicates which of the two directions along P contains x. Then there exists an adaptive algorithm, namely exponential (doubling binary) search, that identifies x using $O\big(\log(\mathrm{dist}_P(a, x) + 1)\big)$ oracle queries [Bentley and Yao, 1976].*

Next we recall the standard definition of pathwidth [Robertson and Seymour, 1983]:

Definition 2.3 (Path decomposition and pathwidth)**.** *Let $G = (V(G), E(G))$ be a graph. A* path decomposition *of G is a sequence of vertex sets*

$$\mathcal{B} = (B_1, B_2, \ldots, B_m), \qquad B_i \subseteq V(G),$$

called bags, satisfying the following conditions:

1. $\bigcup_{i=1}^{m} B_i = V(G)$;

2. *for every edge* $uv \in E(G)$*, there exists an index* i *such that* $\{u, v\} \subseteq B_i$;

3. *for every vertex* $v \in V(G)$*, the set* $\{i : v \in B_i\}$ *is an interval in* $\{1, \ldots, m\}$.

The width of $\mathcal{B}$ *is* $\max_i |B_i| - 1$. *The* pathwidth *of* G*, denoted* $\mathrm{pw}(G)$*, is the minimum width of a path decomposition of* G.

For our purposes, we use the following definition of pathwidth specific for trees which was shown to be equivalent (for trees) by [Kinnersley, 1992].

Definition 2.4 (Recursive characterization for trees)**.** *Let* T *be a tree. Then* $\mathrm{pw}(T) = 0$ *if and only if* T *consists of a single vertex. For* $k \geq 1$*, a tree* T *has pathwidth at most* k *if and only if, for every vertex* $x \in V(T)$*, at most two connected components of* $T \setminus \{x\}$ *have pathwidth at least* k.

Equivalently, $\mathrm{pw}(T) > k$ if and only if there exists a vertex $x \in V(T)$ such that $T \setminus \{x\}$ has at least three connected components of pathwidth at least k.

We now define the spine structure used by the algorithm. Informally, a spine is a path whose removal lowers the pathwidth of every remaining component.

Definition 2.5 (k-spine of a tree)**.** *Let* T *be a tree with* $\mathrm{pw}(T) = k$*. A* k-spine *of* T *is a path* $P \subseteq V(T)$ *such that every connected component of* $T \setminus P$ *has pathwidth at most* $k - 1$.

This is the same object as the *main path* of Suderman [2004], who proved that every tree admits such a path. Hence every tree admits the following recursive decomposition.

Definition 2.6 (k-spine decomposition)**.** *Let* T *be a tree with* $\mathrm{pw}(T) = k$*. A* k-spine decomposition *is formed by recursively removing spines: first remove a* k*-spine* K_T *of* T*, then recursively remove a* $\mathrm{pw}(C)$*-spine from each connected component* C *of* $T[V(T) \setminus K_T]$*, stopping at singleton components. The removed spines form vertex-disjoint paths whose union is* $V(T)$.

A k-spine decomposition need not be unique. Throughout the paper, the algorithm uses an arbitrary fixed k-spine decomposition; the guarantees in Section 3.2 hold for any such choice.

3 Main Algorithm and Analysis

3.1 The Algorithm

We now describe the search algorithm. Fix an arbitrary k-spine decomposition of T. The algorithm maintains a current component T_{cur} and a current start vertex $s_{\mathrm{cur}} \in V(T_{\mathrm{cur}})$. Each phase operates on the spine of the current component. Let P be the spine of T_{cur} specified by the fixed decomposition. The algorithm begins the phase by projecting s_{cur} onto P; that is, it chooses the closest vertex $a \in P$ to s_{cur}.

The algorithm then performs exponential search along P starting from a. The oracle answers on P have a one-dimensional interpretation. Let $x \in P$ be the unique vertex where the path from the target to P meets the spine; if the target lies on P, then x is the target itself. For every queried vertex $v \in P$ with $v \neq x$, the oracle returns the neighbor of v on P in the direction of x. Thus, away from x, the search proceeds exactly as one-dimensional exponential search on a path. When the search reaches x, one of two things happens. If the target lies on P, then the oracle returns here and the algorithm terminates. Otherwise, the oracle returns a neighbor $y \notin P$, and the target lies in the unique connected component of $T_{\mathrm{cur}} \setminus P$ containing y.

In the latter case, the algorithm descends into that off-spine component: it sets T_{cur} to the component of $T_{\mathrm{cur}} \setminus P$ containing y and sets $s_{\mathrm{cur}} := y$. Thus each phase either finds the target on the current spine or identifies the unique lower-pathwidth component containing the target. Since removing a spine reduces the pathwidth of every remaining component by at least one, there are at most k nontrivial phases.

Algorithm 1 gives the full procedure.

Algorithm 1 k-Spine Exponential Search

Require: Tree T, fixed k-spine decomposition, prediction s, direction oracle dir

$T_{\mathrm{cur}} \leftarrow T$, $s_{\mathrm{cur}} \leftarrow s$
while T_{cur} is not a singleton **do**
  Let P be the spine of T_{cur} in the decomposition
  Let a be the closest vertex of P to s_{cur}
  Run exponential search on P from a until either here is returned or the oracle points off P
  if the target is found on P **then**
    return the target
  else
    Let $x \in P$ be the vertex where the oracle points off the spine
    Let $y \notin P$ be the oracle answer at x
    Let C be the component of $T_{\mathrm{cur}} \setminus P$ containing y
    $T_{\mathrm{cur}} \leftarrow C$, $s_{\mathrm{cur}} \leftarrow y$
  end if
end while
return the unique vertex of T_{cur}

3.2 Analysis

We now prove that Algorithm 1 satisfies Theorem 1.3. Namely, the algorithm correctly outputs the target vertex and uses at most $O(k \log \mathrm{dist}(s,t))$ oracle queries, where $k = \mathrm{pw}(T)$. We prove correctness in Section 3.2.1 and query complexity in Section 3.2.2.

3.2.1 Correctness

We first prove that Algorithm 1 always returns the hidden target.

Lemma 3.1. *Algorithm 1 returns the target vertex.*

Proof. Let t denote the hidden target. We prove the following invariant: at the beginning of every iteration of the while loop, $t \in V(T_{\mathrm{cur}})$. The invariant holds initially because $T_{\mathrm{cur}} = T$.

Now consider an arbitrary iteration and assume that $t \in V(T_{\mathrm{cur}})$. Let P be the spine of T_{cur}. Since T_{cur} is a tree and P is connected, there is a unique vertex $x \in P$ closest to t. Equivalently, x is the unique vertex of P at which the path from t to P meets the spine. If $t \in P$, then $x = t$. Otherwise, x is the unique vertex of P whose off-spine component contains t.

We claim that exponential search on P, started from the anchor a, correctly identifies x. For any queried vertex $v \in P$, the oracle returns the neighbor of v on the unique v-to-t path, unless $v = t$, in which case it returns here. Thus every vertex of P on either side of x points along the spine toward x. At x, the oracle either returns here, if $x = t$, or returns the unique neighbor $y \notin P$ on the x-to-t path. Therefore the oracle answers along P have exactly the one-dimensional structure needed by exponential search: the search either finds t on P, or identifies the unique vertex $x \in P$ where the path to t leaves the spine.

If the target is found on P, the algorithm returns t, so the output is correct. Otherwise, the oracle answer at x is a vertex $y \notin P$ on the unique x-to-t path. Hence t lies in the connected component C of $T_{\mathrm{cur}} \setminus P$ containing y. The algorithm sets

$$T_{\mathrm{cur}} \leftarrow C \qquad \text{and} \qquad s_{\mathrm{cur}} \leftarrow y,$$

so the invariant is preserved.

It remains to show that the algorithm terminates. Whenever the algorithm does not return during an iteration, it descends into a component of $T_{\mathrm{cur}} \setminus P$. By the definition of the recursive spine decomposition, every such component has pathwidth strictly smaller than that of T_{cur}. Therefore each descent strictly decreases the pathwidth of the current component. Since pathwidth is a nonnegative integer, after finitely many descents the algorithm reaches a component of pathwidth 0, which is a singleton. By the invariant, that single vertex must be t, and the final line of the algorithm returns it. □

3.2.2 Query Complexity

We now bound the number of oracle queries made by Algorithm 1.

Theorem 3.2. *Algorithm 1 makes $O(k \log D)$ oracle queries, where $k = \mathrm{pw}(T)$ and $D = \mathrm{dist}_T(s, t)$.*

Proof. Consider an arbitrary phase of the algorithm. Let T_{cur} be the current component, let s_{cur} be the current start vertex, and let P be the spine of T_{cur}. Let a be the closest vertex of P to s_{cur}.

By Lemma 3.1, the target t lies in T_{cur} at the beginning of the phase. Let $x \in P$ be the unique vertex of P closest to t. Equivalently, x is the vertex of P at which the path from t to P meets the spine; if $t \in P$, then $x = t$.

The cost of the phase is the cost of exponential search on the path P, starting from a, until it identifies x or finds the target. By the standard analysis of exponential search on a path, this takes

$$O\big(\log(\mathrm{dist}_P(a, x))\big)$$

queries.

We next relate $\mathrm{dist}_P(a, x)$ to the original prediction error D. Since a is the projection of s_{cur} onto P and x is the projection of t onto P, the unique s_{cur}-to-t path in the tree contains the subpath of P from a to x. Hence

$$\mathrm{dist}_P(a, x) \le \mathrm{dist}_T(s_{\mathrm{cur}}, t).$$

Moreover, s_{cur} never moves farther from the target. Initially $s_{\mathrm{cur}} = s$. Whenever the algorithm updates s_{cur}, it sets it to a vertex y on the current s_{cur}-to-t path. Therefore

$$\mathrm{dist}_T(s_{\mathrm{cur}}, t) \le \mathrm{dist}_T(s, t) = D.$$

Thus every phase uses at most $O\big(\log(D)\big)$ queries.

It remains to bound the number of phases. Whenever the algorithm does not terminate during a phase, it descends into a component of $T_{\mathrm{cur}} \setminus P$. After this update, all later queries are made inside this component, so the algorithm never returns to the removed spine P. Since P is the spine of T_{cur} in the fixed spine decomposition, every such component has pathwidth strictly smaller than $\mathrm{pw}(T_{\mathrm{cur}})$. Thus the pathwidth of the current component decreases by at least one after every descent. Since the initial pathwidth is k, there are at most k nontrivial spine-search phases.

Multiplying the per-phase bound by the number of phases gives $O\big(k \log D\big)$ queries in total. □

3.2.3 Proof of the Main Upper Bound

We can now combine correctness and query complexity.

Theorem 1.3. *There is a deterministic algorithm for the Search on Trees problem which, when given a tree $T = (V, E)$ of pathwidth k and a prediction $s \in V$, finds the unknown target $t \in V$ using at most $O(k \log \mathrm{dist}(s, t))$ queries.*

Proof. The result follows immediately from Lemma 3.1 and Theorem 3.2. □

4 Experiments

We empirically evaluate the oracle-query complexity of k-spine search on tree instances derived from real-world networks. Since our theoretical guarantees are stated in terms of oracle queries, the main performance measure is the average number of oracle queries. Given a tree $T = (V, E)$, a prediction $\hat{p} \in V$, a target $p \in V$, and pathwidth $\mathrm{pw}(T) = k$, we compare three algorithms: centroid search, k-spine search, and naive trace. Centroid search is prediction-agnostic and repeatedly queries a centroid of the current feasible subtree, giving worst-case query complexity $O(\log |V(T)|)$. k-spine search is Algorithm 1, implemented using a fixed k-spine decomposition. Naive trace starts at $\hat{p}$ and follows the oracle direction until reaching p, using exactly $\mathrm{dist}_T(\hat{p}, p) + 1$ queries.

Implementation details, including a constant-factor initialization heuristic, are deferred to Appendix B.

The relevant prediction error is $d := \mathrm{dist}_T(\hat{p}, p)$. The analysis of k-spine search suggests a leading dependence of the form $k \log d$, while centroid search has worst-case query complexity $O(\log |V(T)|)$. Equating these terms gives the coarse reference scale

$$k \log d \approx \log |V(T)|, \qquad \text{or equivalently} \qquad d \approx |V(T)|^{1/k}.$$

This scale should not be interpreted as an exact prediction of the empirical crossing point. In the experiments, centroid search is run on the actual tree instances and sampled prediction–target pairs, and its average query cost can be smaller than the worst-case upper bound because of the structure of the tree and the sampled target distribution. Thus the main empirical question is whether k-spine search beats both the actual centroid baseline and the naive trace baseline over a meaningful range of prediction errors.

4.1 Real-world tree instances

The input graphs are taken from the Network Repository [Rossi and Ahmed, 2015]: the Luxembourg road network, the sc-msdoor scientific-computing graph, and two Orkut social-network instances. We convert each graph to a DFS spanning tree rooted at a low-degree peripheral vertex; for directed graphs, we first take the underlying undirected largest connected component. This root is used only for tree construction, not as the prediction. For each tree, we evaluate at fixed prediction error d by uniformly sampling ordered pairs $(\hat{p}, p)$ with $\mathrm{dist}_T(\hat{p}, p) = d$. We run all algorithms on the same pairs and report average oracle queries conditioned on d. Additional sampling details appear in Appendix B.1.

Table 1 summarizes the instances and empirical crossover behavior. The column $|V(T)|^{1/k}$ reports the coarse worst-case comparison scale above. The column "k-spine wins" records the sampled distance interval on which k-spine search has lower average query complexity than both baselines. The empirical crossing is the estimated distance at which the k-spine and centroid curves meet.

Dataset	$\lvert V(T)\rvert$	k	$\lvert V(T)\rvert^{1/k}$	Graph diameter	Mean pairwise graph distance	k-spine wins	Crossing
Luxembourg road network	114,599	5	10.28	1,337	486.85	$d = 10$–22	23.0
sc-msdoor	404,785	6	8.60	149	64.08	$d = 10$–43	43.6
soc-orkut-dir	3,072,441	5	19.84	10	4.21	$d = 10$–53	53.7
soc-orkut	2,997,166	5	19.74	10	4.27	$d = 10$–67	68.0

Table 1: Summary of the real-world DFS tree instances. Prediction–target pairs are uniformly sampled among ordered pairs at fixed tree distance d. The scale $|V(T)|^{1/k}$ is a coarse worst-case reference obtained by comparing $k \log d$ with $\log |V(T)|$. The win range is the sampled interval on which k-spine search uses fewer average oracle queries than both naive trace and centroid search.

Figure 1 shows the pre-crossing regime. The shaded region marks the sampled distances for which k-spine search has lower average oracle-query complexity than both naive trace and centroid search. Figure 2 shows the full-range results over the sampled distance range. These plots use a logarithmic x-axis to display behavior across both small and large prediction errors. They illustrate the large-scale qualitative behavior: naive trace grows with the prediction error, while centroid search and k-spine search remain substantially flatter.

Across all four datasets, k-spine search exhibits a clear intermediate-distance advantage. For very small d, naive trace is difficult to beat: it has essentially no setup cost and uses exactly $\mathrm{dist}_T(\hat{p}, p) + 1$ queries, while our implementation of k-spine search incurs a small initial overhead before beginning the spine search. For very large d, centroid search becomes competitive because its cost is essentially independent of the prediction. Between these regimes, k-spine search uses the prediction while avoiding the linear growth of naive trace, and it achieves the lowest average query complexity over a nontrivial interval in every instance. The initial overhead is small in all experiments and does not affect the asymptotic guarantees.

The empirical crossings are later than the coarse $|V(T)|^{1/k}$ reference scale in all four instances: 23.0 versus 10.28 on Luxembourg, 43.6 versus 8.60 on sc-msdoor, and $53.7, 68.0$ versus roughly 20 on the two Orkut instances. These crossings should not be read as precise validations of a worst-case threshold, since centroid search is evaluated on the actual trees and significantly outperforms its

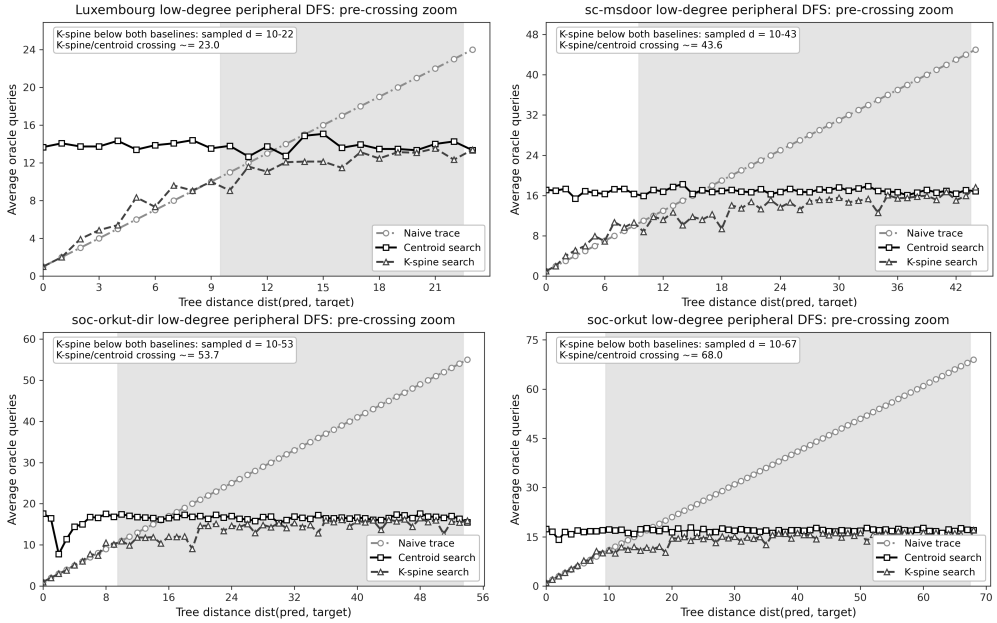

Figure 1: Zoomed pre-crossing results. The shaded region marks distances where k-spine search uses fewer average oracle queries than both baselines.

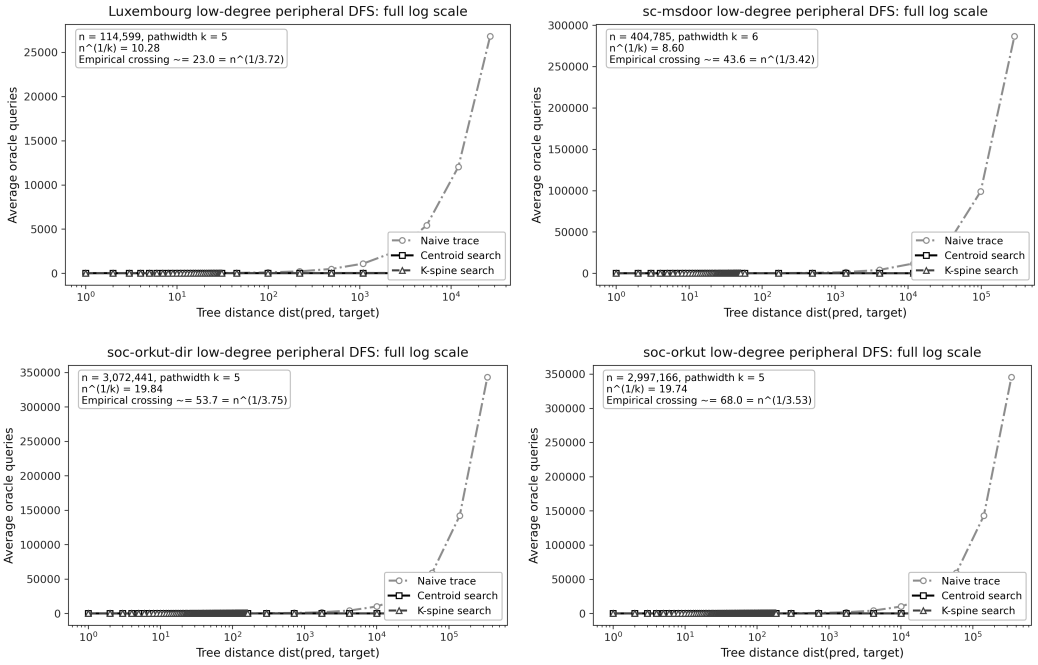

Figure 2: Full-range results on real-world DFS tree instances. Top row: Luxembourg road network and sc-msdoor. Bottom row: soc-orkut-dir and soc-orkut. Naive trace grows with the prediction error, while centroid search and k-spine search remain substantially flatter on this scale.

worst-case bound in these experiments. Rather, the result is stronger than merely beating a theoretical upper bound: k-spine search remains below the actual centroid baseline over a substantial range of prediction errors, while also outperforming naive trace once the error is no longer very small.

References

Priyank Agrawal, Eric Balkanski, Vasilis Gkatzelis, Tingting Ou, and Xizhi Tan. Learning-augmented mechanism design: Leveraging predictions for facility location. *Math. Oper. Res.*, 49(4):2626–2651, 2024. doi: 10.1287/MOOR.2022.0225. URL https://doi.org/10.1287/moor.2022.0225.

Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html.

Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM Journal on Computing*, 28(6):2090–2102, 1999. doi: 10.1137/S009753979731858X. URL https://doi.org/10.1137/S009753979731858X.

Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(76)90071-5. URL https://www.sciencedirect.com/science/article/pii/0020019076900715.

Benjamin Aram Berendsohn, Ishay Golinsky, Haim Kaplan, and László Kozma. Fast Approximation of Search Trees on Trees with Centroid Trees. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-278-5. doi: 10.4230/LIPIcs.ICALP.2023.19. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2023.19.

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10393–10406, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html.

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, Aidin Niaparast, and Sergei Vassilvitskii. Binary search with distributional predictions. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/a4b293979b8b521e9222d30c40246911-Abstract-Conference.html.

J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994. ISSN 0890-5401. doi: https://doi.org/10.1006/inco.1994.1064. URL https://www.sciencedirect.com/science/article/pii/S0890540184710649.

Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 519–532, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341325. doi: 10.1145/2897518.2897656. URL https://doi.org/10.1145/2897518.2897656.

Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 2733–2743, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/161c5c5ad51fcc884157890511b3c8b0-Abstract.html.

Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(92)90234-M. URL https://www.sciencedirect.com/science/article/pii/002001909290234M.

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. doi: 10.1137/1.9781611975994.114. URL https://doi.org/10.1137/1.9781611975994.114.

Alexander Lindermayr and Nicole Megow. Algorithms with predictions. https://algorithms-with-predictions.github.io/, 2022.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM (JACM)*, 68(4):1–25, 2021.

Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*, page 646–662. Cambridge University Press, 2021. doi: 10.1017/9781108637435.037.

Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 379–388, 2006. doi: 10.1109/FOCS.2006.32.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

Neil Robertson and P.D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. ISSN 0095-8956. doi: https://doi.org/10.1016/0095-8956(83)90079-5. URL `https://www.sciencedirect.com/science/article/pii/0095895683900795`.

Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL `https://networkrepository.com`.

Matthew Suderman. Pathwidth and layered drawings of trees. *International Journal of Computational Geometry & Applications*, 14(03):203–225, 2004. doi: 10.1142/S0218195904001433. URL `https://doi.org/10.1142/S0218195904001433`.

Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In *SODA*, pages 3534–3557. SIAM, 2024.

Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977. doi: 10.1109/SFCS.1977.24.

A Lower Bounds

We now show that the dependence on the pathwidth parameter k in Theorem 1.3 is necessary. We construct a family of trees of pathwidth k and maximum degree at most 3 such that every deterministic or zero-error randomized algorithm has some target requiring $\Omega(k \log \operatorname{dist}(s,t))$ queries.

A.1 Hard Instance Construction

Fix integers $k \geq 1$ and $\ell \geq \max\{4, k^2\}$. We recursively define a rooted tree $T_{h,\ell}$ for each $0 \leq h \leq k$. The parameter h denotes the remaining height of the construction, so level 0 is the bottom level.

For the base case, let $T_{0,\ell}$ consist of a single vertex r_0. For $h \geq 1$, construct $T_{h,\ell}$ as follows. Start with a path

$$P_h = (p_1^h, p_2^h, \ldots, p_\ell^h).$$

For each $i \in \{1, \ldots, \ell\}$, attach a disjoint copy $T_{h-1,\ell}^{(i)}$ of $T_{h-1,\ell}$ by adding an edge from p_i^h to the root of $T_{h-1,\ell}^{(i)}$. The root of $T_{h,\ell}$ is defined to be

$$r_h := p_1^h.$$

By construction, every vertex of $T_{h,\ell}$ has degree at most 3. The recursive structure of $T_{h,\ell}$ is illustrated in Figure 3.

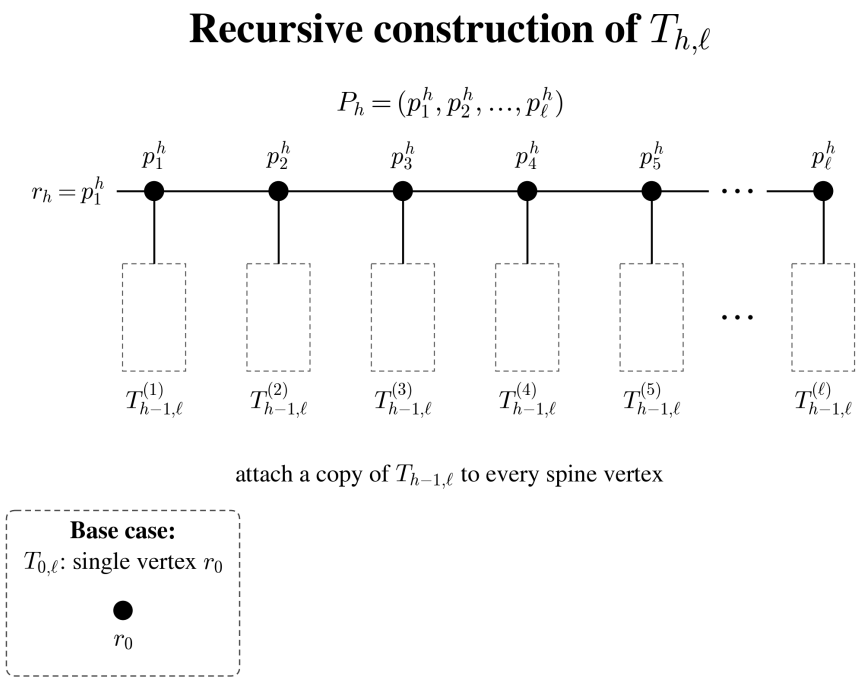

Figure 3: Recursive construction of $T_{h,\ell}$. The tree consists of a spine $P_h = (p_1^h, \ldots, p_\ell^h)$, with a disjoint copy of $T_{h-1,\ell}$ attached to every spine vertex. The base case $T_{0,\ell}$ is a single vertex r_0.

Next we define the bottom-level target set of the final tree $T_{k,\ell}$. Let

$$I := \{\lceil \ell/2 \rceil, \lceil \ell/2 \rceil + 1, \ldots, \ell\}.$$

For a sequence

$$\alpha = (i_k, i_{k-1}, \ldots, i_1) \in I^k,$$

let $v_\alpha \in V(T_{k,\ell})$ be the vertex obtained by descending through the recursive construction according to α. Concretely, starting from the root of $T_{k,\ell}$, we enter the copy of $T_{k-1,\ell}$ attached to the vertex $p_{i_k}^k$.

Inside that copy, we then enter the copy of $T_{k-2,\ell}$ attached to its vertex $p^{k-1}_{i_{k-1}}$, and we continue in this way until reaching a copy of $T_{0,\ell}$. Since $T_{0,\ell}$ consists of a single vertex, this procedure identifies a unique bottom-level vertex v_α.

We define

$$B_{k,\ell} := \{v_\alpha : \alpha \in I^k\}.$$

Thus $B_{k,\ell}$ is the set of admissible targets, all of which lie at level 0 of the construction. Moreover,

$$|B_{k,\ell}| = |I|^k \geq (\ell/2)^k.$$

Figure 4 illustrates how a sequence $\alpha \in I^k$ determines a unique bottom-level target.

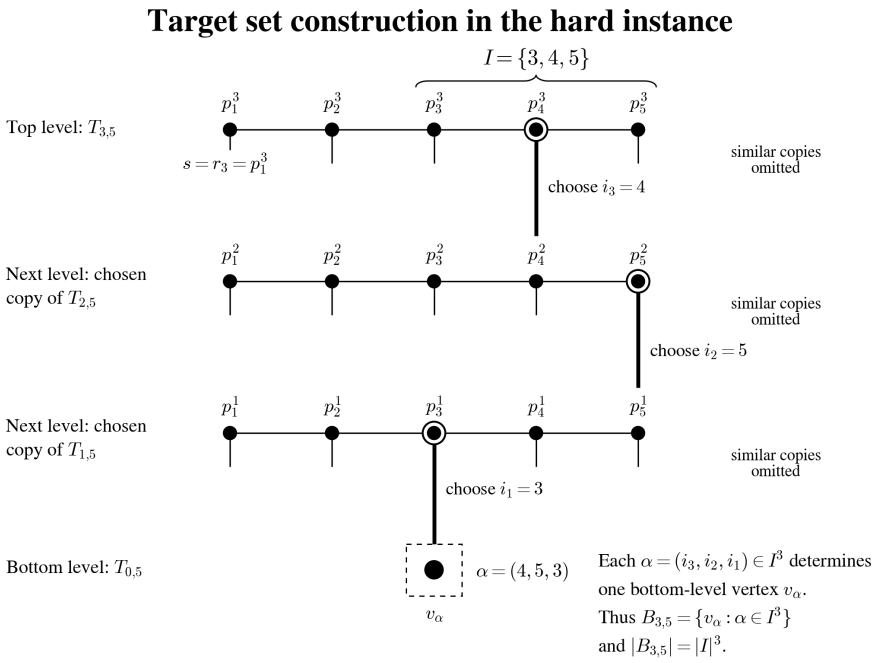

Figure 4: Illustration of the target-set construction for the hard instance, shown here for $k = 3$ and $\ell = 5$. At each recursive level, one chooses an attachment index in $I = \{3, 4, 5\}$. A sequence $\alpha = (i_3, i_2, i_1) \in I^3$ determines a unique bottom-level vertex v_α. In general, each $\alpha \in I^k$ determines one target in $B_{k,\ell}$, so $|B_{k,\ell}| = |I|^k$.

The hard instance is the tree $T_{k,\ell}$ with prediction fixed to be

$$s := r_k = p_1^k,$$

the left endpoint of the top-level spine. The unknown target is promised to lie in the bottom-level target set $B_{k,\ell}$.

Intuitively, $T_{h,\ell}$ consists of a spine on ℓ vertices, with a copy of $T_{h-1,\ell}$ attached to every spine vertex. A target is specified by making one attachment choice from I at each of the k recursive levels. After k such choices, the process reaches a bottom-level copy of $T_{0,\ell}$, which consists of a single vertex. Hence there are $\Theta(\ell)$ choices at each of k levels, and therefore

$$|B_{k,\ell}| = |I|^k = \Theta(\ell^k).$$

We first verify that this construction has the desired pathwidth.

Lemma A.1 (Pathwidth of the construction)**.** *For every $h \geq 0$, the tree $T_{h,\ell}$ has pathwidth exactly h.*

Proof. We prove the claim by induction on h. The case $h = 0$ is immediate, since $T_{0,\ell}$ is a single vertex.

First, we prove the upper bound. Assume inductively that $\mathrm{pw}(T_{h-1,\ell}) = h-1$. For each attached copy $T^{(i)}_{h-1,\ell}$, take a path decomposition of width $h-1$ and add the attachment vertex p^h_i to every bag. This increases the width by at most one, so the resulting bags have width at most h. Moreover, these bags cover the attachment edge from p^h_i to the root of $T^{(i)}_{h-1,\ell}$.

Let $\mathcal{D}_i$ denote this modified path decomposition of the ith attached copy. Concatenate the decompositions in the order

$$\mathcal{D}_1,\ \{p^h_1, p^h_2\},\ \mathcal{D}_2,\ \{p^h_2, p^h_3\},\ \ldots,\ \{p^h_{\ell-1}, p^h_\ell\},\ \mathcal{D}_\ell.$$

This is a valid path decomposition of $T_{h,\ell}$. Every edge inside an attached copy is covered by the corresponding $\mathcal{D}_i$, every attachment edge is covered because p^h_i was added to all bags of $\mathcal{D}_i$, and every edge of P_h is covered by one of the two-vertex bags. The contiguity condition also holds: each vertex inside an attached copy appears only inside its copy's decomposition, while each spine vertex p^h_i appears only in $\mathcal{D}_i$ and in the adjacent path-edge bags. Therefore,

$$\mathrm{pw}(T_{h,\ell}) \le h.$$

It remains to prove the lower bound. Choose an internal vertex p^h_i of P_h, with $2 \le i \le \ell - 1$. Removing p^h_i creates three connected components: one containing the left side of P_h, one containing the right side of P_h, and the attached copy $T^{(i)}_{h-1,\ell}$. Each of these three components contains a copy of $T_{h-1,\ell}$, and hence each has pathwidth at least $h-1$ by the induction hypothesis and monotonicity of pathwidth under taking subgraphs.

We use the standard three-component obstruction for pathwidth on trees: if deleting a vertex of a tree leaves at least three connected components of pathwidth at least r, then the tree has pathwidth at least $r+1$. This follows from the recursive characterization of pathwidth on trees, equivalently from the vertex-separation characterization [Kinnersley, 1992, Ellis et al., 1994]. Applying this with $r = h-1$ gives

$$\mathrm{pw}(T_{h,\ell}) \ge h.$$

Combining the upper and lower bounds gives $\mathrm{pw}(T_{h,\ell}) = h$. □

We next relate ℓ to the distance between the prediction and every possible target.

Lemma A.2 (Prediction error in the construction)**.** *For every target $t \in B_{k,\ell}$,*

$$\mathrm{dist}_{T_{k,\ell}}(s,t) = \Theta(k\ell).$$

Consequently, since $\ell \ge k^2$,

$$\log \mathrm{dist}_{T_{k,\ell}}(s,t) = \Theta(\log \ell).$$

Proof. By the definition of $B_{k,\ell}$, every target $t \in B_{k,\ell}$ is a bottom-level vertex indexed by a sequence

$$\alpha = (i_k, i_{k-1}, \ldots, i_1) \in I^k.$$

At level j, the path from the root of the current copy of $T_{j,\ell}$ to the next lower-level copy moves along the current spine from p^j_1 to $p^j_{i_j}$ and then takes the attachment edge into the chosen copy of $T_{j-1,\ell}$. The distance along the spine is $i_j - 1$, and the attachment edge contributes one additional edge, so level j contributes exactly i_j edges. Therefore,

$$\mathrm{dist}_{T_{k,\ell}}(s,t) = \sum_{j=1}^{k} i_j.$$

Since each $i_j \in I$, we have $\ell/2 \le i_j \le \ell$. Hence

$$\frac{k\ell}{2} \le \mathrm{dist}_{T_{k,\ell}}(s,t) \le k\ell.$$

Thus $\mathrm{dist}_{T_{k,\ell}}(s,t) = \Theta(k\ell)$. Since $\ell \ge k^2$, we have $k \le \sqrt{\ell}$, and therefore

$$\log \ell \le \log(k\ell) \le \log(\ell^{3/2}) = \frac{3}{2}\log \ell.$$

Thus $\log(k\ell) = \Theta(\log \ell)$, which proves the second claim. □

We now prove a distributional decision-tree lower bound for the uniform distribution over the bottom-level targets.

Lemma A.3 (Distributional decision-tree lower bound)**.** *Let μ be the uniform distribution over $B_{k,\ell}$. For every deterministic correct search algorithm A,*

$$\mathbb{E}_{t\sim\mu}[Q_A(t)] = \Omega(k \log \ell),$$

where $Q_A(t)$ denotes the number of oracle queries made by A when the target is t.

Proof. Let

$$N := |B_{k,\ell}|.$$

Restricted to targets in $B_{k,\ell}$, the deterministic algorithm A induces a decision tree. Since $T_{k,\ell}$ has maximum degree at most 3, each oracle query has at most four possible outcomes: the answer here or one of at most three neighboring vertices. Hence this decision tree has branching factor at most 4.

For each target $t \in B_{k,\ell}$, let d_t be the depth of the leaf reached by A on target t. Since A is correct on every target, distinct targets in $B_{k,\ell}$ must reach distinct leaves; otherwise the same transcript would force the same output on two different targets. Moreover,

$$Q_A(t) = d_t.$$

We lower bound the average leaf depth by a simple counting argument. The case $N < 16$ is absorbed into the constant in the $\Omega(\cdot)$ notation, so assume $N \geq 16$. Since the decision tree has branching factor at most 4, the number of leaves of depth at most q is at most

$$\sum_{i=0}^{q} 4^i = \frac{4^{q+1}-1}{3} \leq 4^{q+1}.$$

Choose

$$q := \lfloor \log_4 N \rfloor - 2.$$

Then

$$4^{q+1} \leq \frac{N}{4}.$$

Thus at most $N/4$ targets in $B_{k,\ell}$ can reach leaves of depth at most q. Consequently, at least $3N/4$ targets have leaf depth greater than q.

Since μ is the uniform distribution on $B_{k,\ell}$, we obtain

$$\mathbb{E}_{t\sim\mu}[Q_A(t)] = \frac{1}{N} \sum_{t\in B_{k,\ell}} d_t \geq \frac{3}{4} q = \Omega(\log N).$$

Using $N = |B_{k,\ell}| = |I|^k \geq (\ell/2)^k$, this gives

$$\mathbb{E}_{t\sim\mu}[Q_A(t)] = \Omega(\log |B_{k,\ell}|) = \Omega(k \log \ell),$$

where the last equality uses $\ell \geq 4$. □

We are now ready to prove the lower bound.

Theorem 1.4. *There is no algorithm (deterministic or randomized) for the Search on Trees problem that has expected query complexity $o(k \log dist(s,t))$ for every tree $T = (V, E)$ of pathwidth k, prediction $s \in V$, and target $t \in V$.*

Proof. Fix $k \geq 1$ and $\ell \geq \max\{4, k^2\}$, and consider the hard instance $T_{k,\ell}$ with prediction $s = r_k$. By Lemma A.1, this tree has pathwidth k.

For deterministic algorithms, Lemma A.3 implies that under the uniform distribution on $B_{k,\ell}$, the average query cost is $\Omega(k \log \ell)$. Therefore, for every deterministic correct search algorithm A, there exists some target $t \in B_{k,\ell}$ on which A makes $\Omega(k \log \ell)$ queries.

For randomized algorithms, we assume the algorithm must always output the correct target, and its query complexity is the expected number of oracle queries over its internal randomness. By Yao's minimax principle [Yao, 1977],

$$\min_{\mathcal{R}} \max_{t\in B_{k,\ell}} \mathbb{E}_\rho\big[Q_{\mathcal{R}_\rho}(t)\big] \geq \min_{A} \mathbb{E}_{t\sim\mu}\big[Q_A(t)\big],$$

where $\mathcal{R}$ ranges over zero-error randomized algorithms, ρ denotes the internal randomness of $\mathcal{R}$, and A ranges over deterministic correct search algorithms. By Lemma A.3, the right-hand side is $\Omega(k \log \ell)$. Hence, for every zero-error randomized algorithm $\mathcal{R}$, there exists a target $t \in B_{k,\ell}$ such that

$$\mathbb{E}_\rho\big[Q_{\mathcal{R}_\rho}(t)\big] = \Omega(k \log \ell),$$

with the tree $T_{k,\ell}$ and prediction $s = r_k$ fixed.

Finally, by Lemma A.2, every target $t \in B_{k,\ell}$ satisfies

$$\log \mathrm{dist}_{T_{k,\ell}}(s,t) = \Theta(\log \ell).$$

Thus the hard target guaranteed above satisfies

$$\mathbb{E}_\rho\big[Q_{\mathcal{R}_\rho}(t)\big] = \Omega\big(k \log \mathrm{dist}_{T_{k,\ell}}(s,t)\big).$$

The deterministic statement follows identically without the expectation over ρ. Hence no deterministic or zero-error randomized algorithm can guarantee $o(k \log \mathrm{dist}(s,t))$ expected query complexity on every tree of pathwidth k, prediction s, and target t. □

Finally, we show that Theorem 1.4 implies the statement of Theorem 1.2.

Theorem 1.2. *There is no algorithm (deterministic or randomized) for the Search on Trees problem that has expected query complexity $O(\log \mathrm{dist}(s,t))$ for every prediction $s \in V$ and target $t \in V$.*

Proof. Suppose, for contradiction, that there exists an algorithm for Search on Trees with expected query complexity

$$O(\log \mathrm{dist}(s,t))$$

for every tree T, every prediction $s \in V(T)$, and every target $t \in V(T)$. In particular, this guarantee would hold on the hard instances $T_{k,\ell}$ from Section A.1, for every choice of k.

Choose any non-constant sequence of pathwidths $k \to \infty$. On the corresponding hard instances, the assumed guarantee gives

$$O(\log \mathrm{dist}(s,t)) = o\big(k \log \mathrm{dist}(s,t)\big).$$

This contradicts Theorem 1.4, which states that no deterministic or zero-error randomized algorithm can guarantee $o(k \log \mathrm{dist}(s,t))$ expected query complexity on all trees of pathwidth k.

Therefore, no universal $O(\log \mathrm{dist}(s,t))$ expected-query guarantee is possible for Search on Trees. □

B Experiments

We use one implementation optimization that improves constants but does not affect the asymptotic guarantees. Instead of always starting from the top-level spine, the algorithm first queries the prediction $\hat{p}$. If the oracle returns here, the algorithm terminates. Otherwise, the oracle answer identifies the neighbor of $\hat{p}$ on the unique path to the target, and hence certifies which side of $\hat{p}$ contains the target.

The implementation then starts the spine-search procedure from the lowest decomposition component consistent with this certified direction, rather than from the root of the decomposition. Intuitively, this skips high-level components that cannot contain the target. In our implementation, this component is found by searching upward through the decomposition hierarchy. This optimization can reduce unnecessary high-level spine searches in practice, but it does not affect correctness: the algorithm only skips components certified not to contain the target. It also does not change the asymptotic query-complexity guarantees of Algorithm 1.

The optimization introduces a small initial overhead because the prediction is queried before the recursive spine procedure begins. This explains why, at very small prediction error, naive trace can be slightly better: naive trace has almost no setup cost and uses exactly $\mathrm{dist}_T(\hat{p}, p) + 1$ oracle queries. The overhead is small in all experiments and is dominated once the prediction error leaves this very local regime.

B.1 Experimental Details

For each resulting tree $T = (V, E)$, we evaluate performance at fixed tree distance. For a distance d, define

$$\mathcal{P}_d := \{(\hat{p}, p) \in V(T) \times V(T) : \mathrm{dist}_T(\hat{p}, p) = d\}.$$

We uniformly sample ordered pairs $(\hat{p}, p)$ from $\mathcal{P}_d$, treat $\hat{p}$ as the prediction and p as the target, and run all three algorithms on the same sampled pairs. Thus the x-axis in the plots is the prediction error d, and the y-axis is the average number of oracle queries conditioned on that error.

C Computing k-Spine

We briefly describe the preprocessing step used to compute the recursive spine decomposition. This step is not central to our contribution: our main results concern the number of oracle queries once such a decomposition is available. We therefore use a standard exact tree pathwidth routine as a preprocessing subroutine, rather than optimizing this part of the implementation.

Our implementation is based on the classical characterization of tree pathwidth via vertex separation, together with the EST labeling procedure for trees [Ellis et al., 1994]. Since vertex separation is equivalent to pathwidth, the EST labels allow us to compute the exact pathwidth of each tree component. For each connected component in the recursive decomposition, we root the component at a chosen vertex and compute EST labels bottom-up. These labels identify the bottleneck pathwidth levels in the rooted subtrees and provide enough information to recover a canonical backbone path. We use this backbone as the current spine. Removing this spine leaves connected components of strictly smaller pathwidth, and we recursively apply the same procedure to each remaining component.

Equivalently, one can view this as an implementation of the following straightforward exact procedure. For a connected subtree T', compute its pathwidth level $k' = \mathrm{pw}(T')$. Then find a path P such that every connected component of $T' \setminus P$ has pathwidth at most $k' - 1$. The path P becomes the spine at the current node of the decomposition tree, and the algorithm recurses on the components of $T' \setminus P$.

The preprocessing runs in polynomial time. At each recursive level, the current components are disjoint subtrees of the original tree, so the total number of vertices processed at that level is at most n. Moreover, the recursion depth is at most $\mathrm{pw}(T)$, since the pathwidth level decreases by at least one after each spine removal. For trees, $\mathrm{pw}(T) = O(\log |V|)$, so there are at most logarithmically many recursive levels. Since the underlying EST vertex-separation routine runs in polynomial time on trees, the full preprocessing procedure is polynomial-time as well.

We emphasize that this preprocessing is used only to construct the decomposition before running the search algorithm. The focus of the paper is the query complexity of prediction-based search on the resulting decomposition, not the optimization of the decomposition routine itself.

D Experimental Setup

All experiments are conducted in Python 3.13.9 on a 16-inch MacBook Pro equipped with an Apple M2 Pro chip, consisting of a 12-core CPU with 8 performance cores and 4 efficiency cores, 16 GB of unified memory, and 512 GB of storage, running macOS Tahoe version 26.4.1. We emphasize that our primary metric is oracle-query complexity, which is hardware-independent; the hardware specification is reported only for reproducibility of the implementation-level experiments.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and introduction state our results clearly.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: While we do not have a separate "Limitations" section, our contributions are precise mathematical claims, so it is trivial to see to which settings they apply and to which settings they do not.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We have precise theorem statements and rigorous proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: This paper has a limited experimental evaluation section, which references the datasets we used and the algorithms that we implemented. Since we study the query complexity rather than the running time, implementation details should not matter.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: This paper has a limited experimental evaluation section, which references the datasets we used and the algorithms that we implemented. Since we study the query complexity rather than the running time, implementation details should not matter, so we are not releasing the code itself.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Our experimental section provides all of the necessary details.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Our experiments are meant to show that the theory is predictive of practice and that, in some cases, the practice is even better. We do not include rigorous experiments, since this paper is primarily theoretical, and so do not test for statistical significance.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Setup is stated.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We are compliant with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper is primarily theoretical, detailing a new search algorithm in trees using predictions. While this may be useful in a variety of practical settings, there are not direct paths to negative applications.

    Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Not applicable to our paper.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Any code/data used is cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We are not releasing any new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We have no such experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No such studies.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: There was no usage of LLMs for any of the results in this paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.