Graph Searching with Predictions

Siddhartha Banerjee
Operations Research and Information Engineering, Cornell University, USA

Vincent Cohen-Addad
Google Research, Zurich, Switzerland

Anupam Gupta
Computer Science, Carnegie Mellon University, Pittsburgh, USA

Zhouzi Li
IIIS, Tsinghua University, Beijing, China

Abstract

Consider an agent exploring an unknown graph in search of some goal state. As it walks around the graph, it learns the nodes and their neighbors. The agent only knows where the goal state is when it reaches it. How do we reach this goal while moving only a small distance? This problem seems hopeless, even on trees of bounded degree, unless we give the agent some help. This setting with "help" often arises in exploring large search spaces (e.g., huge game trees) where we assume access to some score/quality function for each node, which we use to guide us towards the goal. In our case, we assume the help comes in the form of *distance predictions*: each node v provides a prediction $f(v)$ of its distance to the goal vertex. Naturally if these predictions are correct, we can reach the goal along a shortest path. What if the predictions are unreliable and some of them are erroneous? Can we get an algorithm whose performance relates to the error of the predictions?

In this work, we consider the problem on trees and give deterministic algorithms whose total movement cost is only $O(OPT + \Delta \cdot ERR)$, where OPT is the distance from the start to the goal vertex, Δ the maximum degree, and the ERR is the total number of vertices whose predictions are erroneous. We show this guarantee is optimal. We then consider a "planning" version of the problem where the graph and predictions are known at the beginning, so the agent can use this global information to devise a search strategy of low cost. For this planning version, we go beyond trees and give an algorithms which gets good performance on (weighted) graphs with bounded doubling dimension.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Algorithms with predictions, network algorithms, graph search

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.52

Funding The authors gratefully acknowledge funding received from the following sources:
Siddhartha Banerjee: NSF: ECCS-1847393, CNS-1955997, ARO: W911NF-19-0217
Anupam Gupta: NSF awards CCF-1955785, CCF-2006953, and CCF-2224718

Acknowledgements Part of this work was done when SB and AG were visitors to the Data-Driven Decision Making program at the Simons Institute for Theoretical Computing in Berkeley.

1 Introduction

Consider an agent (say a robot) traversing an environment modeled as an undirected graph $G = (V, E)$. It starts off at some *root* vertex r, and commences looking for a goal vertex g. However, the location of this goal is initially unknown to the agent, who gets to know it only when it visits vertex g. So the agent starts exploring from r, visits various vertices $r = v_0, v_1, \cdots, v_t, \cdots$ in G one by one, until it reaches g. The cost it incurs at timestep t is the distance it travels to get from v_{t-1} to v_t. How can the agent minimize the total cost?

14th Innovations in Theoretical Computer Science Conference (ITCS 2023).
Editor: Yael Tauman Kalai; Article No. 52; pp. 52:1–52:25
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This framework is very general, capturing not only problems in robotic exploration, but also general questions related to game tree search: how to reach a goal state with the least effort?

Since this is a question about optimization under uncertainty, we use the notion of *competitive analysis*: we relate the cost incurred by the algorithm on an instance to the optimal cost incurred in hindsight. The latter is just the distance $D := d(r, g)$ between the start and goal vertices. Sadly, a little thought tells us that this problem has very pessimistic guarantees in the absence of any further constraints. For example, even if the graph is known to be a complete binary tree and the goal is known to be at some distance D from the root, the adversary can force any algorithm to incur an expected cost of $\Omega(2^D)$. Therefore the competitiveness is unbounded as D gets large. This is why previous works in online algorithms enforced topological constraints on the graph, such as restricting the graph to be a path, or k paths meeting at the root, or a grid [3].

But in many cases (such as in game-tree search) we want to solve this problem for broader classes of graphs—say for complete binary trees (which were the bad example above), or even more general settings. The redeeming feature in these settings is that we are not searching blindly: the nodes of the graph come with estimates of their quality, which we can use to search effectively. What are good algorithms in such settings? What can we prove about them?

In this paper we formalize these questions via the idea of *distance predictions*: each node v gives a prediction $f(v)$ of its distance $d_G(v, g)$ to the goal state. If these predictions are all correct, we can just "walk downhill"—i.e., starting with v_0 being the start node, we can move at each timestep t to a neighbor v_t of v_{t-1} with $f(v_t) = f(v_{t-1}) - 1$. This reaches the goal along a shortest path. However, getting perfect predictions seems unreasonable, so we ask:

> *What if a few of the predictions are incorrect?* Can we achieve an "input-sensitive" or "smooth" or "robust" bound, where we incur a cost of $d(g, r)+$ some function of the prediction error?

We consider two versions of the problem:

The Exploration Problem. In this setting the graph G is initially unknown to the agent: it only knows the vertex $v_0 = r$, its neighbors ∂v_0, and the predictions on all these nodes. In general, at the beginning of time $t \geq 1$, it knows the vertices $V_{t-1} = \{v_0, v_1, \cdots, v_{t-1}\}$ visited in the past, all their neighboring vertices ∂V_{t-1}, and the predictions for all the vertices in $V_{t-1} \cup \partial V_{t-1}$. The agent must use this information to move to some unvisited neighbor (which is now called v_t), paying a cost of $d(v_{t-1}, v_t)$. It then observes the edges incident to v_t, along with the predictions for nodes newly observed.

The Planning Problem. This is a simpler version of the problem where the agent starts off knowing the entire graph G, as well as the predictions at all its nodes. It just does not know which node is the goal, and hence it must traverse the graph in some order.

The cost in both cases is the total distance traveled by the agent until it reaches the goal, and the competitive ratio is the ratio of this quantity to the shortest path distance $d(r, g)$ from the root to the goal.

1.1 Our Results

Our first main result is for the (more challenging) exploration problem, for the case of trees.

► **Theorem 1** (Exploration). *The (deterministic)* TreeX *algorithm solves the graph exploration problem on trees in the presence of predictions: on any (unweighted) tree with maximum* *degree Δ, for any constant $\delta > 0$, the algorithm incurs a cost of*

$$d(r, g)(1 + \delta) + O(\Delta \cdot |\mathcal{E}|/\delta),$$

where $\mathcal{E} := \{v \in V \mid f(v) \neq d(v, g)\}$ is the set of vertices that give erroneous predictions.

One application of the above theorem is for the layered graph traversal problem (see §1.3 for a complete definition).

▶ **Corollary 2** (Robustness and Consistency for the Layered Graph Traversal problem.)**.** *There exists an algorithm that achieves the following guarantees for the layered graph traversal problem in the presence of predictions: given an instance with maximum degree Δ and width k, for any constant $\delta > 0$, the algorithm incurs an expected cost of at most* $\min(O(k^2 \log \Delta)\, OPT, OPT + O(\Delta|\mathcal{E}|))$*.*

The proof of the above corollary is immediate: Since the input is a tree (with some additional structure that we do not require) that is revealed online, we can use the algorithm from Theorem 1. Hence, given an instance I of layered graph traversal (with predictions), we can use the algorithm from Theorem 1 in combination with the [8], thereby being both *consistent* and *robust*: if the predictions are of high quality, then our algorithm ensures that the cost will be nearly optimal; otherwise if the predictions are useless, [8]'s algorithm gives an upper bound in the worst case.

Moreover, we show that the guarantee obtained in Theorem 1 is the best possible, up to constants.

▶ **Theorem 3** (Exploration Lower Bound)**.** *Any algorithm (even randomized) for the graph exploration problem with predictions must incur a cost of at least* $\max(d(r, g), \Omega(\Delta \cdot |\mathcal{E}|))$*.*

Proof. The lower bound of $d(r, g)$ is immediate. For the second term, consider the setting where the root r has Δ disjoint paths of length D leaving it, and the goal is guaranteed to be at one of the leaves. Suppose we are given the "null" prediction, where each vertex predicts $f(v) = D + \ell(v)$ (where $\ell(v)$ is the distance of the vertex from the root, which we henceforth refer to as the *level* of the vertex). The erroneous vertices are the D vertices along the r-g path. Since the predictions do not give any signal at all (they can be generated by the algorithm itself), this is a problem of guessing which of the leaves is the goal, and any algorithm, even randomized, must travel $\Omega(\Delta \cdot D) = \Omega(\Delta \cdot |\mathcal{E}|)$ before reaching the goal. ◀

Our next set of results are for the planning problem, where we know the graph and the predictions up-front, and must come up with a strategy with this global information.

▶ **Theorem 4** (Planning)**.** *For the planning version of the graph exploration problem, there is an algorithm that incurs cost at most*

(i) $d(r, g) + O(\Delta \cdot |\mathcal{E}|)$ *if the graph is a tree, where Δ is the maximal degree.*

(ii) $d(r, g) + 2^{O(\alpha)} \cdot O(|\mathcal{E}|^2)$ *where α is the doubling dimension of G.*

Again, $\mathcal{E}$ is the set of nodes with incorrect predictions.

Note that result (i) is very similar to that of Theorem 1 (for the harder exploration problem): the differences are that we do not lose any constant in the distance $d(r, g)$ term, and also that the algorithm used here (for the planning problem) is simpler. Moreover, the lower bound from Theorem 3 continues to hold in the planning setting, since the knowledge of the graph and the predictions does not help the algorithm; hence result (i) is tight.

We do not yet know an analog of result (ii) for the exploration problem: extending Theorem 1 to general graphs, even those with bounded doubling metrics remains a tantalizing open problem. Moreover, we currently do not have a lower bound matching result (ii); indeed, we conjecture that a cost of $d(r, g) + 2^{O(\alpha)} \cdot |\mathcal{E}|$ should be achievable. We leave these as questions for future investigation.

1.2 Our Techniques

To get some intuition for the problem, consider the case where given a tree and a guarantee that the goal is at distance D from the start node r. Suppose each node v gives the "null" prediction of $f(v) = D + d(r, v)$. In case the tree is a complete binary tree, then these predictions carry no information and we would have to essentially explore all nodes within distance D. But note that the predictions for about half of these nodes are incorrect, so these erroneous nodes can pay for this exploration. But now consider a "lopsided" example, with a binary tree on one side of the root, and a path on the other (Figure 1). Suppose the goal is at distance D along the path. In this case, only the path nodes are incorrect, and we only have $O(D + |\mathcal{E}|) = O(D)$ budget for the exploration. In particular, we must explore more aggressively along the path, and balance the exploration on both sides of the root. But such gadgets can be anywhere in the tree, and the predictions can be far more devious than the null-prediction, so we need to generalize this idea.

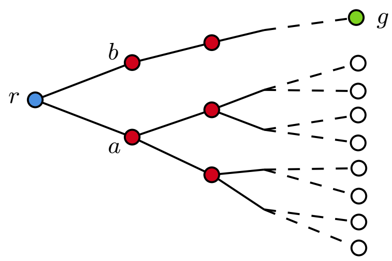


Figure 1 *The subtree rooted on r's child a is a complete binary tree, while the subtree rooted on b is a path to the goal g. "Null" predictions $f(v) = D + d(r, v)$ at every v have a total error D (only nodes on the path from r to g have errors on predictions).*

We start off with a special case which we call the *known-distance* case. This is almost the same as the general problem, but with the additional guarantee that the prediction of the root is correct. Equivalently, we are given the distance $D := d(r, g)$ of the goal vertex from the root/starting node r. For this setting, we can get the following very sharp result:

▶ **Theorem 5** (Known-Distance Case). *The* TreeX-KnownDist *algorithm solves the graph exploration problem in the known-distance case, incurring a cost of at most $d(r, g) + O(\Delta|\mathcal{E}|)$.*

Hence in the zero-error case, or in low-error cases where $|\mathcal{E}| \ll D$, the algorithm loses very little compared to the optimal-in-hindsight strategy, which just walks from the root to the goal vertex, and incurs a cost of D. This algorithm is inspired by the "lopsided" example above: it not only balances the exploration on different subtrees, but also at multiple levels. To generalize this idea from predictions, we introduce the concepts of *anchor* and *loads* (see §2). At a high level, for each node we consider the subtrees rooted at its children, and identify subset of nodes in each of these subtrees which are erroneous depending on which subtree contains the goal g. We ensure that these sets have near-equal sizes, so that no matter which of these subtrees contains the goal, one of them can pay for the others. This requires some delicacy, since we need to ensure this property throughout the tree. The details appear in §3.

Having proved Theorem 5, we use the algorithm to then solve the problem where the prediction for the root vertex may itself be erroneous. Given Theorem 5 and Algorithm 1, we can reduced the problem to finding some node v such that $d(v, g)$ is known; moreover this v must not be very far from the start node r. The idea is conceptually simple: as we explore the graph, if most predictions are correct we can use these predictions to find such a v, otherwise these incorrect predictions give us more budget to continue exploring. Implementing this idea (and particularly, doing this deterministically) requires us to figure out how to "triangulate" with errors, which we do in §4.

Finally, we give the ideas behind the algorithms for the *planning version* of the problem. The main idea is to define the implied-error function $\varphi(v) := |\{u \mid f(u) \neq d(u, v)\}|$, which measures the error if the goal is sitting at node v. Since we know all the predictions and the tree structure in this version of the problem, and moreover $\phi(g) = |\mathcal{E}|$, it is natural to search the graph greedily in increasing order of the implied-error. However, naively doing this may induce a large movement cost, so we bucket nodes with similar implied-error together, and then show that the total cost incurred in exploring all nodes with $\varphi(v) \approx 2^i$ is itself close to 2^i (times a factor that depends on the degree or the doubling dimension). It remains an interesting open problem to extend this algorithm to broader classes of graphs. The details here appear in §5.

1.3 Related Work

Graph Searching. Graph searching is a fundamental problem, and there are too many variants to comprehensively discuss: we point to the works closest to ours. Baeza-Yates, Culberson, and Rawlins [3] considered the exploration problem without predictions on the line (where it is also called the "cow-path" problem), on k-spiders (i.e., where k semi-infinite lines meet at the root) and in the plane: they showed tight bounds of 9 on the deterministic competitive ratio of the line, $1 + 2k^k/(k-1)^{k-1}$ for k-spiders, among other results. Their lower bounds (given for "monotone-increasing strategies") were generalized by Jaillet and Stafford [23]; [24] point out that the results for k-spiders were obtained by Gal [18] before [3] (see also [1]). Kao et al. [29, 28] give tight bounds for both deterministic and randomized algorithms, even with multiple agents.

The *layered graph traversal* problem [42] is very closely related to our model. A tree is revealed over time. At each timestep, some of the leaves of the current tree *die*, and others have some number of children. The agent is required to sit at one of the current (living) leaves, so if the node the agent previously sat is no longer a leaf or is dead, the agent is forced to move. The game ends when the goal state is revealed and objective is to minimize the total movement cost. The *width* k of the problem is the largest number of leaves alive at any time (observe that we do not parameterize our algorithm with this parameter). This problem is essentially the cow-path problem for the case of $w = 2$, but is substantially more difficult for larger widths. Indeed, the deterministic bounds lie between $\Omega(2^k)$ [17] and $O(k2^k)$ [9]. Ramesh [44] showed that the randomized version of this problem has a competitive ratio at least $\Omega(k^2/(\log k)^{1+\varepsilon})$ for any $\varepsilon > 0$; moreover, his $O(k^{13})$-competitive algorithm was improved to a nearly-tight bound of $O(k^2 \log \Delta)$ in recent exciting result by Bubeck, Coester, and Rabani [8].

Kalyanasundaram and Pruhs [26] study the exploration problem (which they call the *searching* problem) in the geometric setting of k polygonal obstacles with bounded aspect ratio in the plane. Some of their results extend to the *mapping* problem, where they must determine the locations of all obstacles. Deng and Papadimitriou [12] study the mapping problem, where the goal is to traverse *all edges* of an unknown directed graph: they give an algorithm with cost $2|E|$ for Eulerian graphs (whereas $OPT = |E|$), and cost $\exp(O(d \log d))|E|$ for graphs with imbalance at most d. Deng, Kameda, and Papadimitriou [11] give an algorithm to map two-dimensional rectilinear, polygonal environments with a bounded number of obstacles.

Kalyanasundaram and Pruhs [27] consider a different version of the mapping problem, where the goal is to visit all vertices of an unknown graph using a tour of least cost. They give an algorithm that is $O(1)$-competitive on planar graphs. Megow et al. [37] extend their algorithm to graphs with bounded genus, and also show limitations of the algorithm from [27].

Blum, Raghavan and Schieber [6] study the *point-to-point navigation* problem of finding a minimum-length path between two known locations s and t in a rectilinear environment; the obstacles are unknown axis-parallel rectangles. Their $O(\sqrt{n})$-competitiveness is best possible given the lower bound in [42]. [30] give lower bounds for randomized algorithms in this setting.

Our work is related in spirit to graph search algorithms like A^*-search which use *score functions* to choose the next leaf to explore. One line of work giving provably good algorithms is that of Goldberg and Harrelson [19] on computing shortest paths without exploring the entire graph. Another line of work related in spirit to ours is that of Karp, Saks, and Wigderson [31] on branch-and-bound (see also [32]).

Noisy Binary Search. A very closely related line of work is that of computing under noisy queries [16]. In this widely-used model, the agent can query nodes: each node "points" to a neighbor that is closer to the goal, though some of these answers may be incorrect. Some of these works include [41, 40, 15, 10, 13, 7]. Apart from the difference in the information model (these works imagine knowing the entire graph) and the nature of hints ("gradient" information pointing to a better node, instead of information about the quality/score of the node), these works often assume the errors are independent, or adversarial with bounded noise rate. Most of these works allow random-access to nodes and seek to minimize the *number* of queries instead of the distance traveled, though an exception is the work of [7]. As far as we can see, the connections between our models is only in spirit. Moreover, we show in §7.3 that results of the kind we prove are impossible if the predictions don't give us distance information but instead just edge "gradients".

Algorithms with Predictions. Our work is related to the exciting line of research on algorithms with predictions, such as in ad-allocation [35], auction pricing [36], page migration [22], flow allocation [34], scheduling [43, 33, 39], frequency estimation [21], speed scaling [4], Bloom filters [38], bipartite matching and secretary problems [2, 14], and online linear optimization [5].

2 Problem Setup and Definitions

We consider an underlying graph $G = (V, E)$ with a known root node r and an unknown *goal* node g. (For most of this paper, we consider the unweighted setting where all edge have unit length; §5.3 and §7.2 discuss cases where edge lengths are positive integers.) Each node has degree at most Δ. Let $d(u, v)$ denote the distance between nodes u, v for any $u, v \in V$, and let $D := d(r, g)$ be the optimal distance from r to the goal node g.

Let us formally define the problem setup. An agent initially starts at root r, and wants to visit goal g while traversing the minimum number of edges. In each timestep $t \in \{1, 2, \ldots\}$, the agent moves from some node v_{t-1} to node v_t. We denote the *visited vertices* at the start of round t by V_{t-1} (with $V_0 = \{r\}$), and use ∂V_{t-1} to denote the *neighboring vertices*—those not in V_{t-1} but having at least one neighbor in V_{t-1}. Their union $V_{t-1} \cup \partial V_{t-1}$ is the set of *observed vertices* at the end of time $t - 1$. Each time t the agent *visits* a new node v_t such

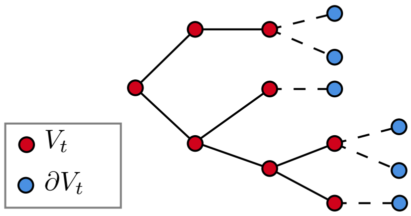


Figure 2 *The observed vertices $V_t \cup \partial V_t$ (and extended subtree $\overline{T}^t := T[V_t \cup \partial V_t]$) at some intermediate stage of the algorithm. Visited nodes V_t are shown in red, and their un-visited neighbors ∂V_t in blue.*

that $V_t := V_{t-1} \cup \{v_t\}$, and it pays the movement cost $d(v_{t-1}, v_t)$, where $v_0 = r$. Finally, when $v_t = g$ and the agent has reached the goal, the algorithm stops. The identity of the goal vertex is known when—and only when—the agent visits it, and we let τ^* denote this timestep. Our aim is to design an algorithm that reaches the goal state with minimum total movement cost:

$$\sum_{t=1}^{\tau^*} d^{t-1}(v_{t-1}, v_t).$$

Within the above setting, we consider two problems:

- In the *planning* problem, the agent knows the graph G (though not the goal g), and in addition, is given a *prediction* $f(v) \in \mathbb{Z}$ for each $v \in V$ of its distance to the goal g; it can then use this information to plan its search trajectory.
- In the *exploration* problem, the graph G and the predictions $f(v) \in \mathbb{Z}$ are initially unknown to the agent, and these are revealed only via exploration; in particular, upon visiting a node for the first time, the agent becomes aware of previously unobserved nodes in v's neighborhood. Thus, at the end of timestep t, the agent knows the set of visited vertices V_t, neighboring vertices ∂V_t, and the predictions $f(v)$ for each observed vertex $v \in V_t \cup \partial V_t$.

In both cases, we define $\mathcal{E} := \{v \in V \mid f(v) \neq d(g, v)\}$ to be the set of *erroneous nodes*. Extending this notation, for the exploration problem, we define $\mathcal{E}^t := \mathcal{E} \cap V_t$ as the erroneous nodes visited by time t.

3 Exploring with a Known Target Distance

Recall that our algorithm for the exploration problem on trees proceeds via the *known-distance* version of the problem: in addition to seeing the predictions at the various nodes as we explore the tree, we are promised that the *distance from the starting node/root r to the goal state g is is exactly some value D*, i.e., $d(r, g) = D$. The main result of this section is Theorem 5, and we restate a rigorous version here.

▶ **Theorem 6.** *If $D = d(r, g)$, the algorithm* TreeX-KnownDist$(r, D, +\infty)$ *finds the goal node g incurring a cost of at most $d(r, g) + O(\Delta|\mathcal{E}|)$.*

Algorithm TreeX-KnownDist is stated in Algorithm 1. For better understanding of it, we first give some key definitions.

3.1 Definitions: Anchors, Degeneracy, and Criticality

For an unweighted tree T, we define the *level* of node v with respect to the root r to be $\ell(v) := d(r, v)$, and so *level* L denotes the set of nodes v such that $d(r, v) = \ell(v) = L$. Since the tree is rooted, there are clearly defined notions of parent and child, ancestor and descendent. Each node is both an ancestor and a descendant of itself. For any node v, let T_v denote the *subtree* rooted at v. Extending this notation, we define the *visited subtree* $T^t := T[V_t]$, and the *extended subtree* $\overline{T}^t := T[V_t \cup \partial V_t]$, and let T_v^t and $\overline{T}_v^t$ be the subtrees of T^t and $\overline{T}^t$ rooted at v.

▶ **Definition 7** (Active and Degenerate nodes). *In the exploration setting, at the end of timestep t, a node $v \in V_t \cup \partial V_t$ is* active *if $T_v^t \neq \overline{T}_v^t$, i.e., there are observed descendants of v (including itself) that have not been visited.*
An active node $v \in V_t \cup \partial V_t$ is degenerate *at the end of timestep t if it has a unique child node in $\overline{T}^t$ that is active.*

In other words, all nodes which have un-visited descendants (including those in the frontier ∂V_t) are active. Active nodes are further partitioned into degenerate nodes that have exactly one child subtree that has not been fully visited, and active nodes that have at least two active children. See Figure 3 for an illustration.

A crucial definition for our algorithms is that of *anchor* nodes:

▶ **Definition 8** (Anchor). *For node $u \in T$, define its* anchor *$\tau(u)$ to be its ancestor in level $\alpha(u) := \frac{1}{2}(D + \ell(u) - f(u))$. If the value $\alpha(u)$ is negative, or is not an integer, or node u itself belongs at level smaller than $\alpha(u)$, we say that u has no anchor and that $\tau(u) = \perp$.*

Figure 3 demonstrates the location of an anchor node $\tau(u)$ for given node u; it also illustrates the following claim, which forms the main rationale behind the definition:

▷ Claim 9. If the prediction for some node u is correct, then its anchor $\tau(u)$ is the least common ancestor (in terms of level ℓ) of u and the goal g. Consequently, if a node u has no anchor, or if its anchor does not lie on the path P^* from r to g, then $u \in \mathcal{E}$.

For any node $v \in T$, define its children be $\chi_i(v)$ for $i = 1, 2, \ldots, \Delta_v$, where $\Delta_v \leq \Delta$ is the number of children for v. Note that the order is arbitrary but prescribed and fixed throughout the algorithm. For any time t, node v, and $i \in [\Delta_v]$, define the visited portion of the subtree rooted at the i^{th} child as $C_i^t(v) := T_{\chi_i(v)}^t$.

▶ **Definition 10** (Loads σ_i and σ). *For any time t, node v, and index $i \in [\Delta_v]$, define*

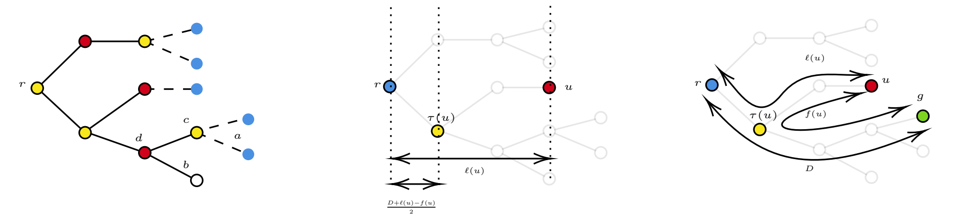
$$\sigma_i^t(v) := |\{u \in C_i^t(v) \mid \tau(u) = v\}|.$$

In other words, $\sigma_i^t(v)$ is the number of nodes in $C_i^t(v)$ that have v as their anchor. Define $\sigma^t(v) = \sum_{i=1}^{\Delta_v} \sigma_i^t(v)$ to be the total number of nodes in $T_v^t \setminus \{v\}$ which have v as their anchor.

▶ **Definition 11** (Critical Node). *For any time t, active and non-degenerate node v, and index $j \in [\Delta_v]$, let $q_j := \arg\min_{i \neq j}\{\sigma_i^t(v) \mid \chi_i(v)$ is active at time $t\}$. Call v a* critical node *with respect to j at time t if it satisfies*

(i) $\sigma_j^t(v) \geq 2\sigma_{q_j}^t(v)$, *namely, the number of nodes of $C_j^t(v)$ that have v as their anchor is at least twice larger than the number of nodes of $C_{q_j}^t(v)$ that have v as their anchor; and*

(ii) $2\sigma_j^t(v) \geq |C_j^t(v)|$, *namely, at least half of the nodes of $C_j^t(v)$ have v as their anchor.*



Active and degenerate nodes Anchor node Illustrating Claim 9

Figure 3 *The first figure from the left illustrates active and degenerate nodes. Nodes such as a (shaded in blue) are in ∂V_t while the rest are visited nodes in V_t. Unshaded node b is inactive (since it has no un-visited descendant), while all other shaded nodes (blue, yellow and red) are active. Among the active nodes, nodes such as c (shaded in yellow) are non-degenerate nodes as they have at least two active children. Finally nodes such as d (shaded in red) are degenerate as they have exactly one active child.*
The second and third figures give an example of anchor node $\tau(u)$ (in yellow) at level $\frac{1}{2}(D+\ell(u)-f(u))$ for given node u (in red) at level $\ell(u)$. The rightmost figure (with root r and goal g also indicated) illustrates Claim 9, showing that when u's prediction $f(u)$ is correct, then its anchor is the least common ancestor of u and goal g (since $D + \ell(u) - f(u)$ is equal to twice the distance of $\tau(u)$ from r).

3.2 The TreeX-KnownDist Algorithm

Equipped with the definitions in §3.1, at a high level, the main idea of the algorithm is to balance the *loads* (as defined in Definition 10) of all the nodes v. Note that if the goal $g \in C_i(v)$, then the nodes $u \in C_i(v)$ that have v as their anchor (i.e., $\tau(u) = v$) have erroneous predictions; hence balancing the loads automatically balances the cost and the budget. To balance the loads, we use the definition of a *critical* node (see Definition 11): whenever a node v becomes critical, the algorithm goes back and explores another subtree of v, thereby maintaining the balance.

More precisely, our algorithm TreeX-KnownDist does the following: at each time step t, it checks whether there is a node that is critical. If there is no such node, the algorithm performs one more step of the current DFS, giving priority to the unexplored child of v_t with smallest prediction. On the other hand, if there is a critical node v, then this v must be the anchor $\tau(v_t)$. In this case the algorithm pauses the current DFS, returns to the anchor $\tau(v_t)$ and resumes the DFS in $\tau(v_t)$'s child subtree having the smallest load (say $C_q(\tau(v_t))$). This DFS may have been paused at some time $t' < t$, and hence is continued starting at node $v_{t'}$. The variable $\text{mem}(v)$ saves the vertex that the algorithm left the subtree rooted on v last time. For example, in this case $\text{mem}(\chi_q(\tau(v_t))) = v_{t'}$. If no such time t' exists, the algorithm starts a new DFS from some child of $\tau(v_t)$ whose subtree has the smallest load (in this case, $\text{mem}(\chi_q(\tau(v_t))) = \perp$). The pseudocode appears as Algorithm 1.

A few observations: (a) While $D = d(r, g)$ does not appear explicitly in the algorithm, it is used in the definition of *anchors* (recall Definition 8). Even when $d(r, g)$, the predicted distance at the root, is not the true distance to the goal (as may happen in Section 4), given any input D in Algorithm 1, we will still define $\tau(v)$ to be v's ancestor at level $\alpha(u) := \frac{1}{2}(D + \ell(u) - f(u))$. (b) The new node v_t is always on the *frontier*: i.e., the nodes which are either leaves of T or have unvisited children. Moreover, (c) the memory value $\text{mem}(v) = \perp$ if and only if $v \notin V_t$, else $\text{mem}(v)$ is on the frontier in the subtree below v.

Algorithm 1 TreeX-KnownDist(r, D, B)

$v_0 \leftarrow r$, $t \leftarrow 0$
mem$(r) \leftarrow r$ and mem$(v) \leftarrow \bot$ for all $v \neq r$
while *$v_t \neq g$ and $|V_t| < B$* **do**
  if *$\tau(v_t) \neq \bot$ and $\tau(v_t)$ is active and not degenerate and $\tau(v_t)$ is critical w.r.t. the index of the subtree containing v_t at time t* **then**
    $q \leftarrow$ the child index q s.t. $\tau(v_t)$ is critical w.r.t. q
    if mem$(\chi_q(\tau(v_t))) = \bot$ **then** $v_{t+1} \leftarrow \chi_q(\tau(v_t)$ **else** $u \leftarrow$ mem$(\chi_q(\tau(v_t))$
  else
    $u \leftarrow v_t$
  while *v_{t+1} undefined and u has no child* **do**
    $w \leftarrow u$'s closest active ancestor
    $q \leftarrow \arg\min_i\{\sigma_i^t(w) \mid \chi_i(w)$ active $\}$
    if mem$(\chi_q(w)) = \bot$ **then** $v_{t+1} \leftarrow \chi_q(w)$ **else** $u \leftarrow$ mem$(\chi_q(w))$
  if *v_{t+1} undefined* **then** $v_{t+1} \leftarrow u$'s child with smallest prediction
  foreach *ancestor u of v_{t+1}* **do** mem$(u) \leftarrow v_{t+1}$
  $t \leftarrow t + 1$

3.3 Analysis for the TreeX-KnownDist Algorithm

The proof of Theorem 6 proceeds in two steps. The first step is to show that the total amount of "extra" exploration, i.e., the number of nodes that do not lie on the r-g path, is $O(\Delta \cdot |\mathcal{E}|)$. Formally, let P^* denote the r-g path; for the rest of this section, suppose $g \in C_1(v)$ for all $v \in P^*$. Define the *extra exploration* to be the number of nodes visited in the subtrees hanging off this path:

$$\text{ExtraExp}(t) := \sum_{v \in P^*} \sum_{i \neq 1} |C_i^t(v)|.$$

▶ **Lemma 12** (Bounded Extra Exploration)**.** *For all times t^*,* $\text{ExtraExp}(t^*) \leq 7\Delta \cdot |\mathcal{E}^{t^*}|$.

Next, we need to control the total distance traveled, which is the second step of our analysis:

▶ **Lemma 13** (Bounded Cost)**.** *For all times t^*,*

$$\sum_{t \leq t^*} d(v_{t-1}, v_t) \leq d(r, v_{t^*}) + 10\,\text{ExtraExp}(t^*) + 16|\mathcal{E}^{t^*}|.$$

Using the lemmas above (setting t^* to be the time τ^* when we reach the goal) proves Theorem 5. In the following sections, we now prove Lemmas 12 and 13.

3.4 Bounding the Extra Exploration

▶ **Lemma 14.** *For any node $v \in T^t$, define $x^t(v)$ as follows:*

(i) *if $g \notin T_v$, then $x^t(v) := \sigma^t(v)$.*
(ii) *if $g \in T_v \setminus \{v\}$, let $g \in T_{\chi_j(v)}$. Define $y_1^t(v) := \sigma_j^t(v)$, $y_2^t(v) := \sum_{i \neq j}(|C_i^t(v)| - \sigma_i^t(v))$ and $x^t(v) := y_1^t(v) + y_2^t(v)$.*

Then $\sum_{v \in T^t} x^t(v) \leq 2|\mathcal{E}^t|$.

Proof. Let P^* be the r-g path in T. If $g \notin T_v$ (i.e., $v \notin P^*$), then by Claim 9 all the nodes with v as anchor belong to $\mathcal{E}$. Else suppose $g \in T_v$ (i.e., $v \in P^*$), and suppose $g \in T_{\chi_j(v)}$. Now all nodes u in $C_j(v)$ having anchor v belong to $\mathcal{E}$, since the least common ancestor of u and g can be no higher than $\chi_j(v)$. This means

$$\sum_{v \in T^t \setminus P^*} x^t(v) + \sum_{v \in P^*} y_1^t(v) \leq \sum_{v \in T^t} |\{u \in \mathcal{E} \mid \tau(u) = v\}| \leq |\mathcal{E}^t|.$$

Finally, suppose $g \in T_v$ (i.e., $v \in P^*$) and $g \in T_{\chi_j(v)}$. Now for any node $u \in T_{\chi_i(v)}$ for $i \neq j$, the least common ancestor of u and g is v. Hence nodes in $T_{\chi_i(v)}$ for $i \neq j$ whose anchor is not v must be wrongly predicted. Denote the set of such nodes by $Y_2^t(v)$. Note that $|Y_2^t(v)| = y_2^t(v)$, and $Y_2^t(v)$ for each $v \in P^*$ are disjoint. Hence we have

$$\sum_{v \in P^*} y_2^t(v) \leq \sum_{v \in P^*} |Y_2^t(v)| \leq |\mathcal{E}^t|.$$

Summing the two inequalities we get the proof. ◀

▶ **Lemma 15.** *For any node $v \in T$ and any index $i \in \{1, 2, \ldots, \Delta_v\}$ such that $\sigma_i^t(v) > \min_q\{\sigma_q^t(v) \mid \chi_q(v)$ is active at time $t\}$. If $v_t \in T_{\chi_j(v)}$ for some $j \neq i$ then $v_{t+1} \notin T_{\chi_i(v)}$.*

Proof. The proof is by contradiction. Assume there is such a time t, and let $w := \arg\min_q\{\sigma_q^t(v) \mid \chi_q(v)$ is active at time $t\}$. Since $v_{t+1} \in T_{\chi_i(v)}$, the subtree under node $\chi_i(v)$ was not fully visited at time r and hence $\chi_i(v)$ was active. By the definition of w and the condition on i in the lemma statement, we have $\sigma_i^t(v) > \sigma_w^t(v)$. Now Algorithm 1 will ensure that v_{t+1} either remains in $T_{\chi_j(v)}$ or moves into $T_{\chi_w(v)}$. ◀

▶ **Lemma 16.** *For any node v on the r-g path P^*, recall the assumption that $g \in C_1(v)$. For any time t and any $i \neq 1$, at least one of the following statements must hold:*

- **(i)** $\sigma_i^t(v) \leq 2\sigma_1^t(v)$.
- **(ii)** $2\sigma_i^t(v) \leq |C_i^t(v)|$.
- **(iii)** $\sigma_i^t(v) = |C_i^t(v)| = 1, \sigma_1^t(v) = 0$.

Proof. For sake of a contradiction, assume there exists t, i such that at time t none of the three statements are true, and this is the first such time. If $|C_i^t(v)| = 1$, then the falsity of second statement gives $\sigma_i^t(v) > 1/2\,|C_i^t(v)| = 1/2$, and so $\sigma_i^t(v) = 1$. Then the first statement being false implies $\sigma_1^t(v) < 1/2$, which means the third statement must hold.

Henceforth let us assume $|C_i^t(v)| \geq 2$. Let $t' < t$ be the latest time such $v_{t'} \in C_i(v)$ and $\tau(v_{t'}) = v$. Because the second statement is false, $\sigma_i^t(v) > 1/2\,|C_i^t(v)| \geq 1$, and so such a time t' exists.

Since t' is the latest time satisfying the condition, we have $\sigma_i^t(v) \leq \sigma_i^{t'}(v) + 1$. Moreover, the number of nodes in $C_i^t(v)$ whose anchor is not v does not decrease, hence $|C_i^t(v)| - \sigma_i^t(v) \geq |C_i^{t'}(v)| - \sigma_i^{t'}(v)$. Also, the number of nodes in $C_1^t(v)$ whose anchor is v does not decrease, hence $\sigma_1^t(v) \geq \sigma_1^{t'}(v)$.

Thus we can get

$$\begin{aligned} &\sigma_i^{t'}(v) - 2\sigma_1^{t'}(v) \geq \sigma_i^t(v) - 2\sigma_1^t(v) - 1 \geq 0 \\ &2\sigma_i^{t'}(v) - |C_i^{t'}(v)| \geq 2\sigma_i^t(v) - |C_i^t(v)| - 1 \geq 0 \end{aligned} \tag{1}$$

Now if $C_i^{t'}(v)$ is completely visited, then obviously $v_{t'+1} \notin C_i(v)$. Otherwise, $C_i^{t'}(v)$ is active. Also because $g \in C_1(v)$, hence $C_1(v)$ cannot be completely visited unless the algorithm ends, which means v is not degenerate and $C_1^{t'}(v)$ is still active. Furthermore, we have inequalities (1), hence v must be critical w.r.t. the subtree containing $v_{t'}$ (because taking $q = 1$ we get the two inequalities for critical hold, although $\sigma_1^{t'}(v)$ may not be the smallest one). Hence at time $t' + 1$ the algorithm will go to a node which is not in $C_i(v)$.

If $v_t \notin C_i^t(v)$: Note that one of the three statements holds for t'. If one of the first two statements is true to t', then the same statement is also true to t because $\sigma_i^t(v) = \sigma_i^{t'}(v)$, $|C_i^t(v)| = |C_i^{t'}(v)|$ and $\sigma_1^t(v) \geq \sigma_1^{t'}(v)$. Otherwise we have $\sigma_i^t(v) = \sigma_i^{t'}(v) = |C_i^t(v)| = |C_i^{t'}(v)| = 1$. Then if $\sigma_1^t(v) = 0$, then the third statement is true to t; if $\sigma_1^t(v) \geq 1$, then the first statement is true to t.

Otherwise $v_t \in C_i^t(v)$: By Lemma 15, there must exist a time $t > t'' > t'$ such that $\sigma_1^{t''}(v) \geq \sigma_i^{t''}(v)$ (otherwise the algorithm will never enter $C_i(v)$ since $C_1(v)$ is always active). Hence by the analysis before, we have $\sigma_1^{t''}(v) \geq \sigma_i^{t'}(v) \geq 1$. Because t' is defined as the latest time before t when $v_t \in C_i(v)$, we have $\sigma_i^{t''}(v) = \sigma_i^{t'}(v)$. Hence $\sigma_i^t(v) \leq \sigma_i^{t'}(v) + 1 \leq 2\sigma_i^{t''}(v) \leq 2\sigma_1^{t''}(v) \leq 2\sigma_1^t(v)$, which is the first statement in this lemma. ◀

▶ **Lemma 17.** *For any node v on the r-g path P^*, and any time t,*

(i) *if $f(\chi_i(v)) = d(\chi_i(v), g)$ for all $i \in [\Delta_v]$ then $\sum_{i\neq 1} |C_i^t(v)| \leq 3\Delta x^t(v)$,*

(ii) *else $\sum_{i\neq 1} |C_i^t(v)| \leq 3\Delta x^t(v) + \Delta$.*

Proof. For the first case: if $f(\chi_i(v)) = d(\chi_i(v), g)$ for all i, then $f(\chi_1(v))$ is the smallest label among all $f(\chi_i(v))$ since the predictions are all correct. Hence by the algorithm, the first node reached among $\{\chi_i(v)\}$ must be $\chi_1(v)$, which means the third statement in Lemma 16 cannot hold. By Lemma 16, for any i, t, $\sigma_i^t(v) \leq 2\sigma_1^t(v)$ or $2\sigma_i^t(v) \leq |C_i^t(v)|$.

If $\sigma_i^t(v) \leq 2\sigma_1^t(v)$: $|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) \geq \sigma_1^t(v) \geq \sigma_i^t(v)/2$; If $2\sigma_i^t(v) \leq |C_i^t(v)|$: $|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) \geq |C_i^t(v)| - \sigma_i^t(v) \geq \sigma_i^t(v)$. Either of them can lead to a conclusion that

$$|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) \geq \sigma_i^t(v)/2.$$

Denote $x_i^t(v) := |C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v)$. Then by $\sigma_1^t(v) \geq 0$ and the inequality above, we have $|C_i^t(v)| \leq x_i^t(v) + \sigma_i^t(v) \leq 3x_i^t(v)$.

Hence $\sum_{i\neq 1} |C_i^t(v)| \leq 3\sum_{i\neq 1} x_i^t(v) = 3\sum_{i\neq 1}(|C_i^t(v)| - \sigma_i^t(v) + (\Delta-1)\sigma_1^t(v)) \leq 3\Delta(\sigma_1^t(v) + \sum_{i\neq 1} |C_i^t(v)| - \sigma_i^t(v)) = 3\Delta x^t(v)$. Here the last equality is because of Lemma 14.

Second, consider other cases. By Lemma 16, $\sigma_i^t(v) \leq 2\sigma_1^t(v) + 1$ or $2\sigma_i^t(v) \leq |C_i^t(v)| + 1$.

If $\sigma_i^t(v) \leq 2\sigma_1^t(v) + 1$: $|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) + 1/2 \geq \sigma_1^t(v) + 1/2 \geq \sigma_i^t(v)/2$; If $2\sigma_i^t(v) \leq |C_i^t(v)| + 1$: $|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) + 1/2 \geq |C_i^t(v)| - \sigma_i^t(v) + 1/2 \geq \sigma_i^t(v)$. Either of them can lead to a conclusion that

$$|C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v) + 1/2 \geq \sigma_i^t(v)/2.$$

Denote $x_i^t(v) := |C_i^t(v)| - \sigma_i^t(v) + \sigma_1^t(v)$, then $|C_i^t(v)| \leq x_i^t(v) + \sigma_i^t(v) \leq 3x_i^t(v) + 1$.

Consequently $\sum_{i\neq 1} |C_i^t(v)| \leq \sum_{i\neq i}(3x_i^t(v) + 1) = 3\Delta x^t(v) + \Delta$, where the last equality is because of Lemma 14. ◀

We can finally bound the extra exploration.

Proof of Lemma 12. Divide the set of nodes on P^* into two sets A, B: A contains the nodes all of whose children are correctly labeled, and B contains the other nodes. Then

$$\begin{aligned}\mathrm{ExtraExp}(t^*) &= \sum_{v\in A}\sum_{i\neq 1}|C_i^{t^*}(v)| + \sum_{v\in B}\sum_{i\neq 1}|C_i^{t^*}(v)| && (2)\\ &\overset{(\star)}{\leq} \sum_{v\in A} 3\Delta x^{t^*}(v) + \sum_{v\in B}(3\Delta x^{t^*}(v)+\Delta) && (3)\\ &= 3\Delta \sum_{v\in P^*} x^{t^*}(v) + \Delta|B| \overset{(\star\star)}{\leq} 6\Delta|\mathcal{E}^{t^*}| + \Delta|\mathcal{E}^{t^*}| = 7\Delta|\mathcal{E}^{t^*}|. && (4)\end{aligned}$$

The inequality $(\star)$ uses Lemma 17, and $(\star\star)$ uses Lemma 14. This proves Lemma 12. ◀

3.5 Bounding the Movement Cost

In this subsection, we bound the total movement cost (and not just the number of visited nodes), thereby proving Lemma 13.

First, we partition the edge traversals made by the algorithm into *downwards* (from a parent to a child) and *upwards* (from a child to its parent) traversals, and denote the cost incurred by the downwards and upwards traversals until time t by M_d^t and M_u^t respectively. We start at the root and hence get $M_d^t = M_u^t + d(r, v_t)$; since we care about the time t^* when we reach the goal state g, we have

$$M^{t^*} = M_u^{t^*} + M_d^{t^*} = 2M_u^{t^*} + d(r, v_t). \tag{5}$$

It now suffices to bound the upwards movement $M_u^{t^*}$. For any edge (u, v) with v being the parent and u the child, we further partition the upwards traversals along this edge into two types:

(i) upward traversals when the **if** statement is true at time t for a node v_s (which lies at or below u) and we move the traversal to another subtree of $\tau(v_s)$ (which lies at or above v), and

(ii) the unique upward traversal when we have completely visited the subtree under the edge.

The second type of traversal happens only once, and it never happens for the edges on the r-g path P^* (since those edges contain the goal state under it, which is not visited until the very end). Hence the second type of traversals can be charged to the extra exploration $\mathrm{ExtraExp}(t^*)$. It remains to now bound the first type of upwards traversals, which we refer to as *callback* traversals.

We further partition the callback traversals based on the identity of the anchor which was critical at that timestep: let $M_u^t(v)$ denote the callback traversal cost at those times s when $v = \tau(v_s)$. Hence the total cost of callback traversals is $\sum_{v\in T^{t^*}} M_u^{t^*}(v)$, and

$$M^{t^*} = d(r, v_t) + 2\Bigg(\mathrm{ExtraExp}(t^*) + \sum_{v\in T^{t^*}} M_u^{t^*}(v)\Bigg). \tag{6}$$

We now control each term of the latter sum.

▶ **Lemma 18.** *For any time t and any node $v \in T^t$, $M_u^t(v) \leq 4\sigma^t(v)$.*

Proof. For node v and index j, let S be the set of times $s \leq t$ for which $v_s \in C_j^s(v)$ and the **if** condition is satisfied with $\tau(v_s) = v$ (i.e, $\tau(v_s) = v$, v is active and not degenerate and v is critical w.r.t. the subtree containing v_s at time s). The cost of the upwards movement at this time is $d(v_s, v) \leq |C_j^s(v)| \leq 2\sigma_j^{t_i}(v)$; the latter inequality is true by criticality.

Lemma 15 ensures that we only enter $C_j(v)$ from a node outside it at some time s when $j \in \arg\min_q\{\sigma_q^s(v)\}$. Hence, if $S = \{t_1, \ldots, t_m\}$ then for each i there must exist a time s_i satisfying $t_i < s_i < t_{i+1}$ such that $\min_q\{\sigma_q^{s_i}(v)\} = \sigma_j^{s_i}(v)$. Consequently,

$$\sigma_j^{t_{i+1}} \geq 2\min_q\{\sigma_q^{t_{i+1}}(v)\} \geq 2\min_q\{\sigma_q^{s_i}(v)\} = 2\sigma_j^{s_i}(v) \geq 2\sigma_j^{t_i}(v).$$

Hence, for each $t_i \in S$,

$$\sum_{i=1}^{m} d(v_{t_i}, v) \leq \sum_{i=1}^{m} 2\sigma_j^{t_i}(v) \leq 4\sigma_j^{t_m}(v) \leq 4\sigma_j^{t}(v). \tag{7}$$

This is the contribution due to a single subtree $T_{\chi_j(v)}$; summing over all subtrees gives a bound of $4\sigma^t(v)$, as claimed. ◀

Proof of Lemma 13. The equation (6) bounds the total movement cost M^{t^*} until time t^* in terms of D, the extra exploration, and the "callback" (upwards) traversals $\sum_v M_u^{t^*}(v)$. Lemma 18 above bounds each term $M_u^{t^*}(v)$ by $4\sigma^{t^*}(v)$. To bound this last summation,

- For each $v \notin P^*$, $\sigma^{t^*}(v) = x^{t^*}(v)$ by Lemma 14.
- For each $v \in P^*$, recall our assumption that $g \in C_1(v)$, so

$$\begin{aligned}\sum_{v\in P^*} \sigma^{t^*}(v) &= \sum_{v\in P^*}\left(\sigma_1^{t^*}(v) + \sum_{i\neq 1}\sigma_i^{t^*}(v)\right)\\ &\leq \sum_{v\in P^*} x^{t^*}(v) + \sum_{v\in P^*}\sum_{i\neq 1}|C_i^{t^*}(v)| = \sum_{v\in P^*} x^{t^*}(v) + \text{ExtraExp}(t^*),\end{aligned}$$

  where $\sigma_1^{t^*}(v) \leq x^{t^*}(v)$ is directly given by definition in Lemma 14.

Summing over all v (using Lemma 14), and substituting into (6) gives the claim. ◀

4 The General Tree Exploration Algorithm

We now build on the ideas from known-distance case to give our algorithm for the case where the true target distance $d(g, r)$ is not known in advance, and we have to work merely with the predictions. Recall the guarantee we want to prove:

▶ **Theorem 1** (Exploration). *The (deterministic)* TreeX *algorithm solves the graph exploration problem on trees in the presence of predictions: on any (unweighted) tree with maximum degree Δ, for any constant $\delta > 0$, the algorithm incurs a cost of*

$$d(r, g)(1 + \delta) + O(\Delta \cdot |\mathcal{E}|/\delta),$$

where $\mathcal{E} := \{v \in V \mid f(v) \neq d(v, g)\}$ is the set of vertices that give erroneous predictions.

Note that Algorithm TreeX-KnownDist requires knowing D exactly in computing *anchors*; an approximation to D does not suffice. Because of this, a simple black-box use of Algorithm TreeX-KnownDist using a *"guess-and-double"* strategy does not seem to work. The main idea behind our algorithm is clean: we explore increasing portions of the tree. If most of the predictions we see have been correct, we show how to find a node whose prediction must be correct. Now running Algorithm 1 rooted at this node can solve the problem. On the other hand, if most of predictions that we have seen are incorrect, this gives us enough budget to explore further.

4.1 Definitions

▶ **Definition 19** (Subtree $\Gamma(u,v)$). *Given a tree T, node v and its neighbor u, let $\Gamma(u,v)$ denote the set of nodes w such that the path from w to v contains u.*

▶ **Lemma 20** (Tree Separator). *Given a tree T with maximum degree Δ and $|T| = n > 2\Delta$ nodes, there exists a node v and two neighbors a, b such that $|\Gamma(a,v)| > \frac{|T|}{2\Delta}$ and $|\Gamma(b,v)| > \frac{|T|}{2\Delta}$. Moreover, such v, a, b can be found in linear time.*

Proof. Let v be a *centroid* of tree T, i.e., a vertex such that deleting v from T breaks it into a forest containing subtrees of size at most $n/2$ [25]. Each such subtree corresponds to some neighbor of v. Let a, b be the neighbors corresponding to the two largest subtrees. Then $|\Gamma(a,v)| \geq \frac{n-1}{\Delta} > \frac{n}{2\Delta}$. Moreover the second largest subtree may contain $\frac{n-|\Gamma(a,v)|-1}{\Delta-1} \geq \frac{n/2-1}{\Delta-1} > \frac{n}{2\Delta}$ when $\Delta < n/2$. ◀

▶ **Definition 21** (Vote $\gamma(u,c)$ and Dominating vote $\gamma(S,c)$). *Given a center c, let the* vote *of any node $u \in T$ be $\gamma(u,c) := f(u) - d(u,c)$. For any set of nodes S, define the* dominating vote *to be $\gamma(S,c) := x$ if $\gamma(u,c) = x$ for at least half of the nodes $u \in S$. If such majority value x does not exist, define $\gamma(S,c) := -1$.*

4.2 The TreeX Algorithm

Given these definitions, we can now give the algorithm. Recall that Theorem 6 says that Algorithm 1 finds g in $d(r_\rho, g) + c_1\Delta \cdot |\mathcal{E}|$ steps, for some constant $c_1 \geq 1$. We proceed in rounds: in round ρ we run Algorithm 1 and visit approximately $\Delta \cdot (c_1+\beta)^\rho$ vertices, where $\beta \geq 1$ is a parameter to be chosen later. Now we focus on two disjoint and "centrally located" subtrees of size $\approx (c_1+\beta)^\rho$ within the visited nodes. Either the majority of these nodes have correct predictions, in which case we use their information to identify one correct node. Else a majority of them are incorrect, in which case we have enough budget to go on to the next round. A formal description appears in Algorithm 2.

Algorithm 2 TreeX(r, β)

```
r_0 ← r, D_0 ← f(v), ρ ← 0
while goal g not found do
    B_ρ ← (c_1 + β)^ρ · (2Δ + 1)
    if B_ρ < D_ρ/β then
        run TreeX-KnownDist(r_ρ, D_ρ, B_ρ)
    else
        run TreeX-KnownDist(r_ρ, D_ρ, D_ρ + c_1 B_ρ)
    T^{ρ+1} ← tree induced by nodes that have ever been visited so far
    r_{ρ+1}, a_{ρ+1}, b_{ρ+1} ← centroid for T^ρ and its two neighbors promised by Lemma 20
    let D_{a,ρ+1} ← γ(Γ(a_{ρ+1}, r_{ρ+1}), r_{ρ+1}) and D_{b,ρ+1} ← γ(Γ(b_{ρ+1}, r_{ρ+1}), r_{ρ+1})
    define new distance estimate D_{ρ+1} ← max{D_{a,ρ+1}, D_{b,ρ+1}}
    move to vertex r_{ρ+1}
    ρ ← ρ + 1
```

4.3 Analysis of the TreeX Algorithm

▶ **Lemma 22.** *If the goal is not visited before round* ρ *when* $B_\rho \geq 4|\mathcal{E}|(2\Delta+1)$*, we have* $D_\rho = d(r_\rho, g)$.

Proof. First, if $|\mathcal{E}| = 0$, then the conclusion holds obviously. So next we assume $|\mathcal{E}| > 0$. The execution of Algorithm 1 in round $\rho - 1$ visits at least $B_{\rho-1} = (c_1+\beta)^{(\rho-1)} \cdot (2\Delta+1)$ distinct nodes. Using the assumption on B_ρ, we have

$$|T^\rho| \geq 4|\mathcal{E}| \cdot (2\Delta+1) > 4\Delta|\mathcal{E}| > 2\Delta.$$

Lemma 20 now implies that both the subtrees $\Gamma(a_\rho, r_\rho)$ and $\Gamma(b_\rho, r_\rho)$ contain more than $\frac{1}{2\Delta}|T^\rho| > 2|\mathcal{E}|$ nodes. Since at most $|\mathcal{E}|$ nodes are erroneous, more than half of the nodes in each of $\Gamma(a_\rho, r_\rho)$ and $\Gamma(b_\rho, r_\rho)$ have correct predictions.

Finally, observe that if $g \notin \Gamma(a_\rho, r_\rho)$, then for any correct node x in $\Gamma(a_\rho, r_\rho)$ we have $f(x) = d(x, g) = d(x, r_\rho) + d(r_\rho, g)$, and hence its vote $\gamma(x, r_\rho) = d(r_\rho, g)$. Since a majority of nodes in $\Gamma(a_\rho, r_\rho)$ are correct, we get

$$D_{a,\rho} = \gamma(\Gamma(a_\rho, r_\rho), r_\rho) = d(r_\rho, g). \tag{8}$$

On the other hand, if $g \in \Gamma(a_\rho, r_\rho)$, then for any correct node x in $\Gamma(a_\rho, r_\rho)$ we have $f(x) = d(x, g) \leq d(x, a_\rho) + d(a_\rho, g) < d(x, r_\rho) + d(r_\rho, g)$. Thus its vote, and hence the vote of a strict majority of nodes in the subtree $\Gamma(a_\rho, r_\rho)$ have

$$D_{a,\rho} < d(r_\rho, g). \tag{9}$$

If no value is in a strict majority, recall that we define $D_{a,\rho} = -1$, which also satisfies (9). The same arguments hold for the subtree $\Gamma(b_\rho, r_\rho)$ as well. Since the goal g belongs to at most one of these subtrees, we have that $D_\rho = \max(D_{a,\rho}, D_{b,\rho}) = d(r_\rho, g)$, as claimed. ◀

▶ **Lemma 23.** *For any round* ρ*,* $d(r_\rho, r) \leq O(B_\rho)$*. Moreover, for any round* ρ *such that* $B_\rho \geq 4|\mathcal{E}|(2\Delta+1)$*,* $d(r_\rho, r) \leq O(B_{\rho-1}) + O(\beta|\mathcal{E}|\Delta)$.

Proof. Since r_ρ is at distance at most $(c_1+c_3)B_{\rho-1} = B_\rho$ from $r_{\rho-1}$, an inductive argument shows that its distance from $r_0 = r$ is at most $(B_0 + \cdots + B_\rho) = O(B_\rho)$.

Moreover, when $B_\rho \geq 4|\mathcal{E}|(2\Delta+1)$, we have $d(r_\rho, g) = D_\rho$ by Lemma 22. Hence if $B_\rho \geq D_\rho/\beta$, the algorithm finds the goal in this round by Theorem 6. Therefore, for any rounds ρ when $B_\rho \geq 4|\mathcal{E}|(2\Delta+1)$ except the last round, the number of nodes visited by Algorithm 1 is at most B_ρ, hence we have $d(r_{\rho+1}, r) \leq d(r_\rho, r) + B_\rho$. We denote ρ' to be the first round ρ' such that $B_{\rho'} \geq 4|\mathcal{E}|(2\Delta+1)$. Thus by induction we have

$$d(r_\rho, r) \leq \sum_{i=\rho'}^{\rho-1} B_i + d(r_{\rho'}, r) \leq O(B_{\rho-1}) + O(B_{\rho'}) \leq O(B_{\rho-1}) + O(\beta|\mathcal{E}|\Delta).$$ ◀

Proof of Theorem 1. Firstly, for the rounds ρ when $B_\rho < 4|\mathcal{E}|(2\Delta+1)$: in each round, Algorithm 1 at most visits $(c_1+\beta)B_\rho = B_{\rho+1}$ nodes, the cost incurred is at most $19B_{\rho+1}$, by Lemma 13. Moreover, the distance from the ending node to $r_{\rho+1}$ is a further $O(B_{\rho+1})$ by Lemma 23. Therefore, since the bounds B_ρ increase geometrically, the cost summed over all rounds until round ρ is $O(B_{\rho+1}) = O(\beta|\mathcal{E}|\Delta)$.

Secondly, for any rounds ρ when $B_\rho \geq 4|\mathcal{E}|(2\Delta+1)$ except the last round, by Lemma 22 and Theorem 6, the number of nodes visited by Algorithm 1 is at most B_ρ (the reasoning is the same as that in Lemma 23). Hence the cost incurred is at most $19B_\rho$. Moreover, by

Lemma 23 the distance from the ending node to $r_{\rho+1}$ is at most $O(B_\rho) + O(\beta\Delta|\mathcal{E}|)$, which means the total cost in round ρ is at most $O(B_\rho) + O(\beta\Delta|\mathcal{E}|)$.

Moreover, if we denote round ρ' to be the first round such that $B_{\rho'} \geq 4|\mathcal{E}|(2\Delta+1)$, then we have, for any round $\rho > \rho'$, $B_\rho > \beta\Delta|\mathcal{E}|$. Hence the cost in round ρ is $O(B_\rho)$.

Finally, consider the last round ρ^*. We only need to consider the case when $B_{\rho^*} \geq 4|\mathcal{E}|(2\Delta+1)$, otherwise the cost has been included in the first case. By Theorem 6, the cost incurred in this round is at most $D_{\rho^*} + c_1\Delta|\mathcal{E}| \leq d(r,g) + d(r_{\rho^*}, r) + c_1\Delta|\mathcal{E}|$. So summing the bounds above, the total cost is at most

$$O(\beta\Delta|\mathcal{E}|) + O(B_{\rho'}) + O(\beta\Delta|\mathcal{E}|) + \sum_{i=\rho'+1}^{\rho^*-1} O(B_i) + d(r,g) + d(r_{\rho^*}, r) + c_1\Delta|\mathcal{E}|$$

$$\leq d(r,g) + O(B_{\rho^*-1}) + O(\beta\Delta|\mathcal{E}|) \leq d(r,g) + O(d(r,g)/\beta) + O(\beta\Delta|\mathcal{E}|)$$

Here the final inequality uses that

$$B_{\rho^*-1} \leq D_{\rho^*-1}/\beta \leq (d(r,g) + O(\beta B_{\rho^*-1}))/\beta \leq (d(r,g) + O(B_{\rho^*-1}))/\beta.$$

Setting $\beta = O(1/\delta)$ gives the proof. ◀

5 The Planning Problem

In this section we consider the planning version of the problem when the entire graph G (with unit edge lengths, except for §5.3), the starting node r, and the entire prediction function $f : V \to \mathbb{Z}$ are given up-front. The agent can use this information to plan its exploration of the graph. We propose an algorithm for this version and then prove the cost bound for trees, and then for a graph with bounded doubling dimension. We begin by defining the *implied-error* function $\varphi(v)$, which gives the total error if the goal is at node v.

▶ **Definition 24** (Implied-error). *The* implied-error function $\varphi : V \to \mathbb{Z}$ *maps each node* $v \in V$ *to* $\varphi(v) := |\{u \in V \mid d(u,v) \neq f(u)\}|$, *which is the* ℓ_0 *error if the goal were at* v.

The search algorithm for this planning version is particularly simple: we visit the nodes in rounds, where round ρ visits nodes with implied-error φ value at most $\approx 2^\rho$ in the cheapest possible way. The challenge is to show that the total cost incurred until reaching the goal is small. Observe that $|\mathcal{E}| = \varphi(g)$, so if this value is at most 2^ρ, we terminate in round ρ.

Algorithm 3 FullInfoX

```
3.1  ρ ← 0, S_{-1} ← ∅, r_{-1} ← r
3.2  while g not found do
3.3      S_ρ ← {v ∈ T | φ(v) < 2^ρ} \ (∪_{i=-1}^{ρ-1} S_i)
3.4      if S_ρ ≠ ∅ then
3.5          C_ρ ← min-length Steiner Tree on S_ρ
3.6          go to an arbitrary node r_ρ in S_ρ
3.7          visit all nodes in C_ρ using an Euler tour of cost at most 2|C_ρ|, and return to r_ρ
3.8      else
3.9          r_ρ ← r_{ρ-1}
3.10     ρ ← ρ + 1
```

5.1 Analysis

Recall our main claim for the planning algorithm:

▶ **Theorem 4** (Planning). *For the planning version of the graph exploration problem, there is an algorithm that incurs cost at most*

(i) $d(r, g) + O(\Delta \cdot |\mathcal{E}|)$ *if the graph is a tree, where* Δ *is the maximal degree.*

(ii) $d(r, g) + 2^{O(\alpha)} \cdot O(|\mathcal{E}|^2)$ *where* α *is the doubling dimension of* G*.*

Again, $\mathcal{E}$ *is the set of nodes with incorrect predictions.*

The proof relies on the fact that Algorithm 3 visits a node in S_ρ only after visiting all nodes in $\cup_{s<\rho} S_s$ and not finding the goal g; this serves a proof that $|\mathcal{E}| = \varphi(g) \geq 2^\rho$. The proof below shows that (a) the cost of the tour of C_ρ is bounded and (b) the total cost of each transition is small. Putting these claims together then proves Theorem 4. We start with a definition.

▶ **Definition 25** (Midpoint Set). *Given a set of nodes* U*, define its* midpoint set $M(U)$ *to be the set of points* w *such that the distance from* w *to all points in* U *is equal.*

▶ **Lemma 26** (φ-Bound Lemma). *For any two sets of nodes* $S, U \subseteq G$*, we have*

$$\sum_{v \in U} \varphi(v) \geq |S \setminus M(U)|.$$

Proof. If node $w \in S$ does not lie in $M(U)$, then there are two nodes $u, v \in U$ for which $d(u, w) \neq d(v, w)$. This means $f(w)$ cannot equal both of them, and hence contributes to at least one of $\varphi(u)$ or $\varphi(v)$. ◀

▶ **Corollary 27.** *For any two nodes* $u, v \in G$*, we have* $d(u, v) \leq \varphi(u) + \varphi(v)$*.*

Proof. Apply Lemma 26 for set $U = \{u, v\}$ and S being a (shortest) path between them (which includes both u, v). All edges have unit lengths so $|S| = d(u, v) + 1$; moreover, $|M(U) \cap S| \leq 1$. ◀

5.1.1 Analysis for Trees (Theorem 4(i))

▶ **Lemma 28** (Small Steiner Tree). *If* $\rho = 0$ *then* $|C_\rho| = 1$ *else* $|C_\rho| \leq O(\Delta \cdot 2^\rho)$*.*

Proof. If $\rho = 0$, then S_ρ contains all nodes with $\varphi(v) = 0$; there can be only one such node. Else if $|S_\rho| \leq 1$ then $|C_\rho| \leq 1 \leq 2^\rho$, so assume that $|S_\rho| > 1$ and let $u_1, u_2 := \arg\max_{u,v \in S_\rho} \{d(u, v)\}$ be a farthest pair of nodes in S_ρ. Consider path p from u_1 to u_2: if all nodes $w \in p$ have $d(w, u_1) \neq d(w, u_2)$, then the midpoint set $|M(\{u_1, u_2\})| = 0$, so Lemma 26 says $|C_\rho| \leq \varphi(u_1) + \varphi(u_2) \leq 2 \times 2^\rho = 2^{\rho+1}$, giving the proof. Hence, let's consider the case where there exists $w \in p$ with $d(w, u_1) = d(w, u_2)$.

Let w's neighbors in C_ρ be $q_1, \ldots, q_k$ for some $k \leq \Delta$. If we delete w and its incident edges, let $C_{\rho,i}$ be the subtree of C_ρ containing q_i; suppose that $u_1 \in C_{\rho,1}$ and $u_2 \in C_{\rho,2}$. Choose any arbitrary vertex $u_i \in (C_{\rho,i} \cap S_\rho)$; such a vertex exists because C_ρ is a min-length Steiner tree connecting S_ρ. Let $U := \{u_1, \ldots, u_k\}$.

Consider any node $x \neq w$ in C_ρ: this means $x \in C_{\rho,j}$ for some j. Choose $i \in \{1, 2\}$ such that $i \neq j$. By the tree properties, $d(x, u_i) = d(x, w) + d(w, u_i)$. Moreover, we have $d(u_i, u_{2-i}) \geq d(u_j, u_{2-i})$ by our choice of $\{u_1, u_2\}$, so $d(w, u_i) \geq d(w, u_j)$. This means

$$d(x, u_i) = d(x, w) + d(w, u_i) \geq d(x, w) + d(w, u_j) = d(x, q_j) + d(u_j, q_j) + 2 > d(x, u_j),$$

which means $x \notin M(U)$. In summary, $M(U) = \{w\}$ or $|M(U)| = 0$, so applying Lemma 26 in either case gives

$$|C_\rho| \le |C_\rho \setminus M(U)| + 1 \le \sum_{i=1}^{k} \varphi(u_i) + 1 \le \Delta \cdot (2^\rho + 1). \quad ◀$$

▶ **Lemma 29** (Small Cost for Transitions). *Consider the first round ρ_0 such that $r_{\rho_0} \neq r$, then $d(r, r_{\rho_0}) \le d(r, g) + |\mathcal{E}| + 2^{\rho_0}\mathbf{1}_{(\rho_0>0)}$. For each subsequent round $\rho > \rho_0$, $d(r_{\rho-1}, r_\rho) \le 2^{\rho+1}$.*

Proof. If the first transition happens in round ρ_0, its cost is

$$d(r, r_{\rho_0}) \le d(r, g) + d(g, r_{\rho_0}) \le d(r, g) + \varphi(g) + \varphi(r_{\rho_0}) \le d(r, g) + |\mathcal{E}| + 2^{\rho_0}\mathbf{1}_{(\rho_0>0)},$$

where we used Corollary 27 for the second inequality. For all other transitions, Corollary 27 again gives $d(r_{\rho-1}, r_\rho) \le \varphi(r_{\rho-1}) + \varphi(r_\rho) \le 2^{\rho-1} + 2^\rho \le 2^{\rho+1}$. ◀

Proof of Theorem 4(i). Suppose g belongs to S_ρ, then $|\mathcal{E}| \ge 2^{\rho-1} \cdot \mathbf{1}_{\rho>0}$. But now the cost over all the transitions is at most $d(r, g) + |\mathcal{E}| + O(2^\rho) \cdot \mathbf{1}_{\rho>0}$ by summing the results of Lemma 29. The cost of the Euler tours are at most $\sum_{s \le \rho} 2(|C_s| - 1)$ by Lemma 28, which gives at most $O(\Delta \cdot 2^\rho) \cdot \mathbf{1}_{\rho>0}$. Combining these proves the theorem. ◀

5.2 Analysis for Bounded Doubling Dimension (Theorem 4(ii))

For a graph $G = (V, E)$ with doubling dimension α, and unit-length edges, we consider running Algorithm 3, as for the tree case. We merely replace Lemma 28 by the following lemma, and the rest of the proof is the same as the proof of the tree case:

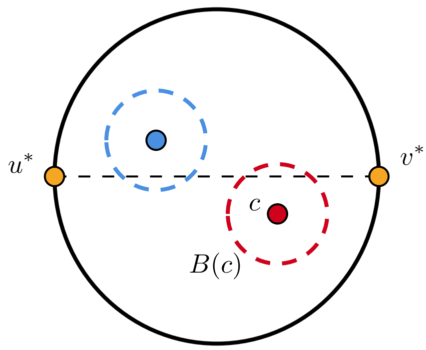


Figure 4 *Let u^*, v^* be the diameter of the set S_ρ (i.e, $u^*, v^* = argmax_{u,v \in S_\rho} d(u, v)$). c is any node in N and $B(c)$ is its neighbor. We show in Claim 31 that the size of $B(c)$ is $O(2^\rho)$.*

▶ **Lemma 30.** *The total length of the tree C_ρ is at most $2^{O(\alpha)} \cdot 2^{2\rho}$.*

Proof. If $|S_\rho| \le 1$, then $|C_\rho| \le 1$. Hence next we assume that $|S_\rho| \ge 2$. Define $R := \max_{u,v \in S_\rho} d(u, v)$, and let $u^*, v^* \in S_\rho$ be some points at mutual distance R. Let N be an $R/8$-net of S_ρ. (An ε-net N for a set S satisfies the properties (a) $d(x, y) \ge \varepsilon$ for all $x, y \in N$, and (b) for all $s \in S$ there exists $x \in N$ such that $d(x, s) \le \varepsilon$.) Since the metric has doubling dimension α, it follows that $|N| \le (\frac{R}{R/8})^{O(\alpha)} = 2^{O(\alpha)}$ [20]. Let each point in S_ρ choose a closest net point (breaking ties arbitrarily), and let $B(c) \subseteq S_\rho$ be the points that chose $c \in N$ as their closest net point (see Figure 4 for a sketch).

▷ Claim 31. For each net point $c \in N$, we have $|B(c)| \le O(2^\rho)$.

Proof. Because $d(v^*, c) + d(u^*, c) \geq d(u^*, v^*) = R$, hence without loss of generality we assume $d(v^*, c) \geq R/2$. For any point $w \in B(c)$, $d(w, v^*) \geq d(v^*, c) - d(c, w) \geq R/2 - R/8 > R/8 \geq d(w, c)$. Hence w is not in $M(\{c, v^*\})$. Hence by Lemma 26,

$$2^{\rho+1} \geq \varphi(c) + \varphi(v^*) \geq |S_\rho \setminus M(\{v^*, c\})| \geq |B(c)|.$$ ◀

There are $2^{O(\alpha)}$ net points, so $|S_\rho| \leq 2^{O(\alpha)} \cdot 2^\rho$. Finally, Corollary 27 holds for general unit-edge-length graphs, so the cost of connecting any two nodes in S_ρ is at most 2^ρ, and therefore $|C_\rho| \leq 2^{O(\alpha)} \cdot 2^{2\rho}$. ◀

Using Lemma 30 instead of Lemma 28 in the proof of Theorem 4(i) gives the claimed bound of $2^{O(\alpha)} \cdot |\mathcal{E}|^2$, and completes the proof of Theorem 4(ii).

5.3 Analysis for Bounded Doubling Dimension: Integer Lengths

In this part, we further generalize the proof above to the case when the edges can have positive integer lengths. Consider an graph $G = (V, E)$ with doubling dimension α and general (positive integer) edge lengths. Define the ℓ_1 analog of the implied-error function to be:

$$\varphi_1(v) := \sum_{u \in V} |f(u) - d(u, v)|.$$

Since we are in the full-information case, we can compute the φ_1 value for each node. Observe that $\varphi_1(g)$ is the ℓ_1-error; we prove the following guarantee.

▶ **Theorem 32.** *For graph exploration on arbitrary graphs with positive integer edge lengths, the analog of Algorithm 3 that uses φ_1 instead of φ, incurs a cost $d(r, g) + 2^{O(\alpha)} \cdot O(\varphi_1(g))$.*

The proof is almost the same as that for the unit length case. We merely replace Corollary 27 and Claim 31 by the following two lemmas.

▶ **Lemma 33.** *For any two vertices u, v, their distance $d(u, v) \leq 1/2(\varphi_1(u) + \varphi_1(v))$.*

Proof. By definition of φ_1 we have $\varphi_1(u)+\varphi_1(v) \geq |f(u)|+|f(v)-d(u, v)|+|f(u)-d(u, v)|+|f(v)| \geq 2d(u, v)$. ◀

▷ Claim 34. For each net point $c \in N$, we have $\sum_{v \in B(c)} d(v, u^*) \leq O(2^\rho)$.

Proof. Let w be the node among u^*, v^* that is further from c; by the triangle inequality, $d(c, w) \geq R/2$. By the properties of the net, $d(v, c) \leq R/8$. Again using the triangle inequality, $d(v, w) \geq 3R/8$. Hence

$$\varphi_1(w) + \varphi_1(c) \geq \sum_{v \in B(c)} \big(|f(v) - d(v, w)| + |f(v) - d(v, c)|\big) \geq |B(c)| \cdot (3R/8 - R/8).$$

Since both $w, c \in S_\rho$, this implies that

$$|B(c)| \cdot R \leq 4(\varphi_1(w) + \varphi_1(c)) \leq O(2^\rho).$$

Finally, we use that $d(v, u^*) \leq R$ by our choice of R to complete the proof. ◀

Now to prove Theorem 32, we mimic the proof of Theorem 4(ii), just substituting Lemma 33 and Claim 34 instead of Corollary 27 and Claim 31.

6 Closing Remarks

In this paper we study a framework for graph exploration problems with predictions: as the graph is explored, each newly observed node gives a prediction of its distance to the goal. While graph searching is a well-explored area, and previous works have also studied models where nodes give directional/gradient information ("which neighbors are better"), such distance-based predictions have not been previously studied, to the best of our knowledge. We give algorithms for exploration on trees, where the total distance traveled by the agent has a relatively benign dependence on the number of erroneous nodes. We then show results for the planning version of the problem, which gives us hope that our exploration results may be extendible to broader families of graphs. This is the first, and most natural open direction.

Another intriguing direction is to reduce the space complexity of our algorithms, which would allow us to use them on very large implicitly defined graphs (say computation graphs for large dynamic programming problems, say those arising from reinforcement learning problems, or from branch-and-bound computation trees). Can we give time-space tradeoffs? Can we extend our results to multiple agents? A more open-ended direction is to consider other forms of quantitative hints for graph searching, beyond distance estimates (studied in this paper) and gradient information (studied in previous works).

References

1 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, 2003.

2 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020*, 2020.

3 R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. URL: https://www.sciencedirect.com/science/article/pii/S0890540183710540, doi:https://doi.org/10.1006/inco.1993.1054.

4 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020*, 2020.

5 Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. In *International Conference on Machine Learning*, pages 822–831. PMLR, 2020.

6 Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, 1997. doi:10.1137/S0097539791194931.

7 Lucas Boczkowski, Uriel Feige, Amos Korman, and Yoav Rodeh. Navigating in trees with permanently noisy advice. *ACM Trans. Algorithms*, 17(2):15:1–15:27, 2021. doi:10.1145/3448305.

8 Sébastien Bubeck, Christian Coester, and Yuval Rabani. Shortest paths without a map, but with an entropic regularizer, 2022. URL: https://arxiv.org/abs/2202.04551, doi:10.48550/ARXIV.2202.04551.

9 William R. Burley. Traversing layered graphs using the work function algorithm. *J. Algorithms*, 20(3):479–511, 1996. doi:10.1006/jagm.1996.0024.

10 Argyrios Deligkas, George B. Mertzios, and Paul G. Spirakis. Binary search in graphs revisited. *Algorithmica*, 81(5):1757–1780, 2019. doi:10.1007/s00453-018-0501-y.

11 Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment I: the rectilinear case. *J. ACM*, 45(2):215–245, 1998. doi:10.1145/274787.274788.

12 Xiaotie Deng and Christos H Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

13 Dariusz Dereniowski, Stefan Tiegel, Przemyslaw Uznanski, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.4.

14 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In Péter Biró, Shuchi Chawla, and Federico Echenique, editors, *EC '21: The 22nd ACM Conference on Economics and Computation, Budapest, Hungary, July 18-23, 2021*, pages 409–429. ACM, 2021. doi:10.1145/3465456.3467623.

15 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 519–532. ACM, 2016. doi:10.1145/2897518.2897656.

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

17 Amos Fiat, Dean P. Foster, Howard J. Karloff, Yuval Rabani, Yiftach Ravid, and Sundar Vishwanathan. Competitive algorithms for layered graph traversal. *SIAM J. Comput.*, 28(2):447–462, 1998. doi:10.1137/S0097539795279943.

18 Shmuel Gal. *Search games*, volume 149 of *Mathematics in Science and Engineering*. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, 1980.

19 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: *A* search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 156–165. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070455.

20 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 534–543. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238226.

21 Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

22 Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. Online page migration with ml advice. *arXiv preprint arXiv:2006.05028*, 2020.

23 Patrick Jaillet and Matthew Stafford. Online searching. *Oper. Res.*, 49(4):501–515, 2001. doi:10.1287/opre.49.4.501.11227.

24 Patrick Jaillet, Matthew Stafford, and Shmuel Gal. Note: Online searching / on the optimality of the geometric sequences for the m ray search online searching. *Oper. Res.*, 50(4):744–745, 2002.

25 Camille Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math.*, 70:185–190, 1869. doi:10.1515/crll.1869.70.185.

26 Bala Kalyanasundaram and Kirk Pruhs. A competitive analysis of algorithms for searching unknown scenes. *Computational Geometry*, 3(3):139–155, 1993. URL: https://www.sciencedirect.com/science/article/pii/0925772193900322, doi:https://doi.org/10.1016/0925-7721(93)90032-2.

27 Bala Kalyanasundaram and Kirk R Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138, 1994.

28 Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Lisa Yin. Optimal constructions of hybrid algorithms. *J. Algorithms*, 29(1):142–164, 1998. doi:10.1006/jagm.1998.0959.

29 Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Inf. Comput.*, 131(1):63–79, 1996. doi:10.1006/inco.1996.0092.

30 Howard J. Karloff, Yuval Rabani, and Yiftach Ravid. Lower bounds for randomized k-server and motion-planning algorithms. *SIAM J. Comput.*, 23(2):293–312, 1994. doi:10.1137/S0097539792224838.

31 Richard M. Karp, Michael E. Saks, and Avi Wigderson. On a search problem related to branch-and-bound procedures. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 19–28. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.34.

32 Richard M. Karp and Yanjun Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computation. *J. ACM*, 40(3):765–789, 1993. doi:10.1145/174130.174145.

33 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

34 Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing, 2020. arXiv:2011.11743.

35 Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Allocating online advertisement space with unreliable estimates. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 288–294. ACM, 2007. doi:10.1145/1250910.1250952.

36 Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1856–1864, 2017.

37 Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theoretical Computer Science*, 463:62–72, 2012. URL: https://www.sciencedirect.com/science/article/pii/S0304397512006445, doi:https://doi.org/10.1016/j.tcs.2012.06.034.

38 Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 462–471, 2018.

39 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

40 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1096–1105. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347202.

41 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 379–388. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.32.

42 Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991. URL: `https://www.sciencedirect.com/science/article/pii/0304397591902632`, `doi:https://doi.org/10.1016/0304-3975(91)90263-2`.

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

44 Hariharan Ramesh. On traversing layered graphs on-line. *J. Algorithms*, 18(3):480–512, 1995. `doi:10.1006/jagm.1995.1019`.

7 Further Discussion

7.1 ℓ_0-versus-ℓ_1 Error in Suggestions

Most of the paper deals with ℓ_0 error: namely, we relate our costs to $|\mathcal{E}|$, the number of vertices that give incorrect predictions of their distance to the goal. Another reasonable notion of error is the ℓ_1 error: $\sum_v |f(v) - d(v,g)|$.

For the case of integer edge-lengths and integer predictions, both of which we assume in this paper, it is immediate that the ℓ_0-error is at most the ℓ_1-error: if v is erroneous then the former counts 1 and the latter at least 1. If we are given integer edge-lengths but fractional predictions, we can round the predictions to the closest integer to get integer-valued predictions f', and then run our algorithms on f'. Any prediction that is incorrect in f' must have incurred an ℓ_1-error of at least $1/2$ in f. Hence all our results parameterized by the ℓ_0 error imply results parameterized with the ℓ_1 error as well.

7.2 Extending to General Edge-Lengths

A natural question is whether a guarantee like the one proved in Theorem 1 can be shown for trees with general integer weights: let us see why such a result is not possible.

1. The first observation is that the notion of error needs to be changed from ℓ_0 error something that is homogeneous in the distances, so that scaling distances by $C > 0$ would change the error term by C as well. One such goal is to guarantee the total movement to be

   $$O(d(r,g) + \text{ some function of the } \ell_p \text{ error}),$$

   where ℓ_p-error is $(\sum_v |f(v) - d(v,g)|^p)^{1/p}$.
2. Consider a complete binary tree of height h, having 2^h leaves. Let all edges between internal nodes have length 0, and edges incident to leaves have length $L \gg 1$. The goal is at one of the leaves. Let all internal nodes have $f(v) = L$, and let all leaves have prediction $2L$. Hence the total ℓ_p error is $2L$, whereas any algorithm would have to explore half the leaves in expectation to find the goal; this would cost $\Theta(2^h \cdot L)$, which is unbounded as h gets large.
3. The problem is that zero-length edges allow us to simulate arbitrarily large degrees. Moreover, the same argument can be simulated by changing zero-length edges to unit-length edges; the essential idea remains the same. and setting $f(v)$ for each node v to be L plus its distance to the root. Setting $L \geq 2^h$ gives the total ℓ_p error to be $O(L + 2^h)$, whereas any algorithm would incur cost at least $\approx L \cdot 2^h$.

This suggests that the right extension to general edge-lengths requires us to go beyond just parameterizing our results with the maximum degree Δ; this motivates our study of graphs with bounded doubling dimension in §5.

7.3 Gradient Information

Consider the information model where the agent gets to see *gradient* information: each edge is imagined to be oriented towards the endpoint with lower distance to the goal. The agent can see some noisy version of these directions, and the error is the number of edges with incorrect directions. We now show an example where both the optimal distance and the error are D, but any algorithm must incur cost $\Omega(2^D)$. Indeed, take a complete binary tree of depth D, with the goal at one of the leaves. Suppose the agent sees all edges being directed towards the root. The only erroneous edges are the D edges on the root-goal path. But any algorithm must suffer cost $\Omega(2^D)$.