Chasing Small Sets Optimally Against Adaptive Adversaries

Christian Coester
University of Oxford, UK

Alexa Tudose
University of Oxford, UK

Abstract

We study deterministic online algorithms for the problem of chasing sets of cardinality at most k in a metric space, also known as metrical service systems and equivalent to width-k layered graph traversal. We resolve the 30-year-old gap of $\Omega(2^k) \cap O(k2^k)$ on the competitive ratio of this problem by giving an $O(2^k)$-competitive deterministic algorithm. This bound is optimal even among randomized algorithms against adaptive adversaries. We also (slightly) improve the deterministic lower bound to D_k, defined recursively by $D_1 = 1$ and $D_{k+1} = 2D_k + \sqrt{8 + 8D_k} + 3$, which we conjecture to be exactly tight. For $k = 3$, we provide a matching upper bound of D_3. Our results imply slightly improved upper and lower bounds for distributed asynchronous collective tree exploration and for the k-taxi problem, respectively.

Our algorithm generalizes the classical doubling strategy, previously known to be optimal for $k = 2$. The previous best bound for general k was achieved by the generalized work function algorithm (WFA), and was known to be tight for WFA. Our improved bound therefore implies that WFA is sub-optimal for chasing small sets.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, competitive analysis, chasing small sets, layered graph traversal, metrical service systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.67

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2605.10927` [17]

Funding *Christian Coester*: Funded by the European Union (ERC, CCOO, 101165139). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Problem definition

Small set chasing, also known as metrical service systems, was introduced by Chrobak and Larmore in 1991 [14]. In this problem, a player must move in a metric space (M, d) in order to serve a sequence of requests. The player is initially located at $s_0 \in M$. At time $t \in \{1, \dots, T\}$, a set $S_t \subseteq M$ with $1 \leq |S_t| \leq k$ is revealed, and the player has to relocate to a point $s_t \in S_t$, paying cost $d(s_{t-1}, s_t)$. The number of requests T and the parameter k are not known to the player in advance. The goal of the player is to minimize the total incurred cost. The player is said to be ρ-competitive if its cost is at most

$$\rho \cdot \min \left\{ \sum_{t=1}^{T} d(x_{t-1}, x_t) \,\middle|\, x_0 = s_0, x_1 \in S_1, \dots, x_T \in S_T \right\}.$$

Related work on deterministic algorithms

When introducing the problem, Chrobak and Larmore gave an optimal k-competitive deterministic algorithm for uniform metric spaces, and a 9-competitive deterministic algorithm for $k = 2$ in arbitrary metric spaces [14]. The first competitive deterministic algorithm for general metric spaces and arbitrary k is due to [19] and achieves a competitive ratio of $O(9^k)$. In the same paper, they also show a lower bound of 2^{k-2} for deterministic algorithms. By designing better algorithms, the initial exponential gap of $O((9/2)^k)$ between upper and lower bounds was narrowed to polynomial $O(k^3)$ by Ramesh [23], and further to linear $O(k)$ by Burley [13], at which the problem had been stuck for 30 years. We close the gap to a constant factor, with a different algorithm than previous works.

The algorithm proposed by Burley is the generalized work function algorithm (WFA), a general purpose algorithm applicable to many online problems, and the leading candidate algorithm for resolving the famous k-server conjecture: For the k-server problem, WFA is known to achieve the optimal competitive ratio up to a factor of at most 2 [21], and it is conjectured to be exactly optimal. By contrast, our results imply that WFA is not optimal for chasing small sets, since Burley showed that his analysis is tight for WFA [13].

Context and motivation

The central role of chasing small sets is emphasized by its connection to numerous other problems. Its alternative name, metrical service systems, showcases the connection to metrical task systems, a more general problem in which each request comes with a function that specifies the cost to serve the request in each point of the metric space [9]. Note that metrical service systems is a particular case where the service costs are in $\{0, \infty\}$. Another related problem is convex body chasing [20, 12, 24, 4], where requested sets are not subject to cardinality constraints, but are required to be convex.

Furthermore, small set chasing is equivalent to the layered graph traversal problem, introduced by Papadimitriou and Yannakakis in [22]. In fact, some of the earlier mentioned results on small set chasing were obtained by working on layered graph traversal. In this problem, a searcher needs to traverse a connected undirected graph G with non-negative edge weights. The vertices of G are partitioned into layers $L_0, \ldots, L_N$, so that edges exist only between vertices in consecutive layers. The searcher is initially located in a starting vertex s in L_0 and needs to reach a target vertex t. Initially, the searcher knows only s, the vertices in L_1, and the weighted edges connecting s to these vertices. All other vertices and edges are hidden. When the searcher reaches a vertex in L_i for the first time, the vertices in L_{i+1} are revealed, together with the weighted edges between L_i and L_{i+1}. Then, the searcher needs to move from its current position to a vertex in L_{i+1}, using any edges which have been revealed so far.[1] When the searcher reaches the penultimate layer, the last layer containing only the target vertex t is revealed, together with the corresponding edges. If the graph G contains at most k vertices in each layer, we say that G has width k. Note that the searcher does not know k or the number of layers in advance. The goal of the searcher is to minimize the total distance traveled. The searcher is said to be ρ-competitive if its total traveled distance is at most $\rho \cdot \mathrm{dist}(s, t)$, where $\mathrm{dist}(s, t)$ is the distance between s and t in G.

Computing the offline optimum in layered graph traversal simultaneously captures the shortest path problem and the dynamic programming paradigm, and therefore gives a natural motivation for the problem. The special case in which the graph is unweighted has been

[1] This may require back-tracking through previous layers.

studied in [6], but we focus on the general, weighted version of the problem. In the equivalence between small set chasing and layered graph traversal, requests correspond to layers, and the width of the layered graph corresponds to the maximum cardinality of a set in a request. A proof of this equivalence can be found in [19].

For width $k = 2$, layered graph traversal is equivalent to the linear search problem (also known as the cow path problem), studied since at least 1970 [7] and later popularized in the online algorithms literature following [5]. In this problem, a searcher located on a line tries to find an unknown target. The optimal deterministic competitive ratio of 9 is achieved by a simple doubling strategy: the searcher explores the two directions alternately up to a threshold that increases in powers of 2. Similar doubling strategies are ubiquitous in online algorithms (e.g., [1]). Our algorithm for general k can be seen as a recursive generalization of this simple idea.

Layered graph traversal can be reduced to the k-taxi problem, where one has to move a set of k servers (i.e., taxis) in a metric space in order to serve passenger requests consisting of a starting point and a destination [15]. Moreover, the problem of distributed asynchronous collective tree exploration, where a tree is explored by a team of k agents, reduces to layered graph traversal [18]. Additionally, small set chasing also has applications in designing learning-augmented algorithms: [3] combine multiple online algorithms by essentially reducing this task to layered graph traversal on disjoint paths, and [2] show how to obtain a hybrid algorithm that is competitive against the best dynamic combination of the available predictors by reducing[2] the problem to layered graph traversal. More generally, set chasing may be viewed as a *meta problem* of online decision making, where the currently requested set corresponds to the legal configurations that an algorithm may choose at that step.

Randomized algorithms and different adversarial models

In the setting of randomized algorithms, an algorithm's cost is defined by taking the expectation over its randomness. Most work on randomized online algorithms focuses on the *oblivious adversary* model, as this is the only model where randomization can yield exponential improvements [8]. An oblivious adversary is one that fixes an instance upfront, before any random choices by the algorithm are made. For this setting, Chrobak and Larmore [14] gave an optimal H_k-competitive randomized algorithm for chasing small sets on uniform metric spaces and a 4.6-competitive randomized algorithm for $k = 2$ on general metric spaces. For arbitrary k and general metric spaces, the first polynomial competitive ratio of $O(k^{13})$ is due to Ramesh [23], who also gave a lower bound of $\Omega\left(\frac{k^2}{\log^{1+\epsilon} k}\right)$ for arbitrary $\epsilon > 0$. Recently, these bounds were tightened to $\Theta(k^2)$ by Bubeck, Coester and Rabani [10, 11]. Interestingly, [11] note that lower bounds for layered graph traversal inspired their construction used to refute the randomized k-server conjecture.

By contrast, in *adaptive adversary* models, requests are allowed to depend on past decisions made by the algorithm. It is well known that randomization offers only limited benefit against such adversaries.[3] For small set chasing under adaptive adversaries, we show that randomization can improve the competitive ratio by at most a constant factor. This model is particularly relevant to chasing small sets, as several close connections to other problems

[2] The general problem considered in [2] is in fact equivalent to layered graph traversal; the straightforward reduction in the reverse direction constructs k predictors whose suggested positions at any time step cover the entire current layer.

[3] In general, a ρ-competitive randomized algorithm against adaptive online adversaries implies a ρ^2-competitive deterministic algorithm [8].

are known to hold only in the adaptive setting. For example, the previously-mentioned reduction from small set chasing to k-taxi given in [15] requires adaptive adversaries, as it issues new requests depending on previous algorithmic actions. Similarly, the reduction from distributed asynchronous collective tree exploration to layered graph traversal in [18] requires an adaptive adversary. Here, the request set in small set chasing corresponds roughly to (part of) the boundary of the region explored so far, which itself depends on prior moves of the agents. The best known bounds for both problems are improved as a consequence of our results.

1.1 Our results

Our main result is the following.

▶ **Theorem 1.** *There exists an $O(2^k)$-competitive deterministic algorithm for chasing sets of cardinality at most k.*

Our upper bound matches the asymptotic lower bound of $\Omega(2^k)$, and thus settles the optimal asymptotic competitive ratio of small set chasing (and, equivalently, layered graph traversal). In fact, we show in the full version of our paper [17, Appendix E] that our algorithm is asymptotically optimal not only among deterministic algorithms, but also among randomized algorithms competing against adaptive adversaries.

▶ **Theorem 2.** *Every randomized online algorithm for chasing sets of cardinality at most k has competitive ratio at least 2^{k-1} against an adaptive online adversary.*

We additionally improve the exact lower bound for deterministic algorithms from 2^{k-2} [19] to D_k, defined recursively by

$$D_1 = 1 \quad \text{and} \quad D_{k+1} = 2D_k + \sqrt{8 + 8D_k} + 3. \tag{1}$$

▶ **Theorem 3.** *Every deterministic online algorithm for chasing sets of cardinality at most k has a competitive ratio of at least D_k, for D_k as defined in* (1).

Note that $D_2 = 9$ is known to be the optimal competitive ratio for $k = 2$. We conjecture that D_k is exactly tight for all $k \geq 1$. We prove in the full version of our paper [17, Appendix A] that this indeed holds for $k = 3$.

▶ **Theorem 4.** *There exists a D_3-competitive deterministic online algorithm for chasing sets of cardinality at most* 3.

Since distributed asynchronous collective tree exploration reduces to width-k layered graph traversal [18], and width-k layered graph traversal reduces to k-taxi [15], we also obtain improved upper and lower bounds, respectively, for these problems.

▶ **Corollary 5.** *There exists a distributed asynchronous algorithm that explores any tree of n nodes and depth D in at most $2n + O(k2^k D)$ moves.*

▶ **Corollary 6.** *Every deterministic online algorithm for k-taxi has a competitive ratio of at least D_k, for D_k as defined in* (1).

Since competitive k-taxi algorithms for general (infinite) metric spaces are unknown except for $k \leq 3$ [16], the constant-factor improvement is admittedly modest. Still, we hope it may offer insight for future algorithm design, especially since $D_2 = 9$ is known to be tight for the 2-taxi problem [15]. In a similar vein, our algorithm for small set chasing is directly inspired by our lower bound construction.

1.2 Overview of techniques

Layered graph traversal is known to be equivalent to the case where the graph is a tree [19]. To simplify the description and analysis of our algorithms, it is convenient to further reduce the problem to a two-player game on a tree which evolves over time, as already done in [10]. This reduction is very natural: after layer L_i is revealed, the algorithm only needs to remember the Steiner tree which connects the starting vertex s to the vertices in L_i (all vertices which are not part of this Steiner tree are "dead-ends", since they do not have any descendant in L_i). Moreover, moving to a vertex in L_i is equivalent to moving to a leaf of the Steiner tree. Therefore, we can model layered graph traversal as "chasing" the leaves of the Steiner tree, where the Steiner tree changes when a new layer is revealed. The game formulation helps express the changes of the Steiner tree as a sequence of simple transformations, which we can reason about more easily.

In Section 2, we first provide a sketch of our lower bound of D_k for deterministic algorithms, which will serve to motivate our algorithm. As in the 2^{k-2} lower bound of [19], we use a recursive construction consisting of two branches, each containing a concatenation of lower bound instances of width $k-1$. In each step, the branch currently occupied by the algorithm is extended with an additional lower bound instance, while the other branch advances with zero-length edges. Unlike the construction of [19], which could be stopped at any time to yield a ratio of 2^{k-2}, our improved lower bound is only attained at certain times when the ratio between the branch lengths is maximized over the course of the algorithm, and the bound we obtain depends on this ratio. Since we can stop an instance only at those specific moments, the adversary cannot easily control the absolute cost of an instance, as it depends on the behavior of the algorithm. Establishing a rigorous lower bound therefore requires some care: On the one hand, the cost of a recursive instance should not be too large, as otherwise an online algorithm could avoid it by switching to the other branch. On the other hand, it also should not be too small (e.g., exponentially decaying), as we need to be able to stack many recursive instances to obtain unbounded total cost. In the full version of our paper [17, Appendix D], we show that we can lower and upper bound the cost of an instance, which allows to use them effectively in the recursive construction.

Our lower bound suggests that there is a "sweet spot" for the ratio between the two branch lengths at which point the algorithm should switch to the other branch. Specifically, for instances of width k, the algorithm should switch to the other branch if the optimal value in its own branch is a factor $x_k := 1 + \sqrt{\frac{2}{1+D_{k-1}}}$ larger than the optimal value in the other branch. This immediately suggests a simple online algorithm: switch to the shorter branch when the sweet spot of ratio x_k between branch lengths is reached, and employ an analogous strategy (for smaller k) recursively within each branch. In the full version of our paper [17, Appendix A], we show that this strategy is indeed D_3-competitive for instances of width 3.

However, generalizing the potential used for width 3 to larger k does not seem straightforward [17, Appendix B]. In Section 4, we derive a natural potential function inspired by the lower bound and show that it can *almost* prove D_k-competitiveness, except for two types of problems triggered by dead-ends in layered graph traversal. We then address these issues in Section 5 by refining the algorithm and potential function through two methods, which we call *forgetting* and *imbalancing*. While the presentation of these techniques will be specific to layered graph traversal, the underlying principles may be applicable more broadly. The motivating idea is to transform an instance into a canonical hard form, as witnessed in the lower bound construction. *Forgetting* corresponds to treating parts of the instance as if they had not yet been revealed, which we execute by truncating terms in the potential function. *Imbalancing* is a method of rounding an instance towards the highly unbalanced structure of

worst-case instances, by distorting some edge lengths. Mathematically, the benefit of such distortions manifests through larger potential function values, and only costs a constant factor in the competitive ratio.

1.3 Evolving tree game

We now describe the evolving tree game, which is essentially the same as the one defined in [10] except that our growth operation is discrete rather than continuous and we define the game directly for binary trees.

We say that a tree is a *stemmed binary tree* if it satisfies the following conditions: a) it is a rooted tree, and the root has degree 1; b) all vertices except the root have either 0 or 2 children.

The evolving tree game is a two-player game involving stemmed binary tree T with nonnegative edge weights. We denote by w_e the weight of an edge e. For a non-root node u, we denote by $e(u)$ the edge between u and its parent. Initially, the tree consists of two vertices connected by a zero-length edge. The first player, called *the adversary*, can apply the following operations on T:

- **Growth**: for a leaf l and $h > 0$, increase the length of the edge $e(l)$ incident to l by h.
- **Deletion**: for a leaf l that is not the unique child of the root, delete l together with its incident edge $e(l)$. Since the parent p of l now has only one child remaining (and hence degree two), smooth the tree at p as follows: let $e_1 = \{u, p\}$ and $e_2 = \{p, v\}$ be the edges incident to p, and replace them by a single edge $e = \{u, v\}$ of weight $w_e := w_{e_1} + w_{e_2}$.
- **Fork**: for a leaf l, connect two new vertices to l by edges of length 0.

The other player, referred to as *the algorithm*, responds to each operation by choosing a leaf of T to occupy. If the adversary grows by h the leaf l where the algorithm is located, after the growth the algorithm is located at some point along the edge incident to l, at distance h from l. Therefore, the algorithm needs to either move back to l or choose a different leaf of T. If the adversary deletes the leaf where the algorithm is located, the algorithm has to move to another leaf of T. The algorithm pays a cost equal to the total distance it moves, and its goal is to minimize the total cost incurred during the game.

The adversary can end the game after any operation. The algorithm is said to be ρ-competitive if it incurs cost at most $\rho \cdot d$, where d is the length of the shortest root-to-leaf path in the evolving tree at the end of the game. The instance of the game is said to have width k if there are at most k leaves which exist at the same time in T. The algorithm does not have access to k in advance.

A proof of the following reduction, which was given similarly in [10], can be found in the full version of our paper [17, Appendix F].

▶ **Lemma 7.** *If there exists a ρ-competitive algorithm for the width-k evolving tree game, then there exists a ρ-competitive algorithm for width-k layered graph traversal.*

2 Lower bound sketch

In this section, we give an informal sketch of the lower bound proof (Theorem 3). The complete proof can be found in the full version of our paper [17, Appendix D]. The construction of our lower bound will be helpful in motivating our algorithm later. We formulate the hard lower bound instance as an instance of the evolving tree game (rather than as an instance of layered graph traversal, which we will do in the more formal proof).

Constructing the instance

We construct the hard instance inductively on the width k of the tree. Starting from a tree which contains a single leaf, we perform a fork to obtain a tree with two branches. While the algorithm is located in one of the branches, we repeatedly start a *phase* in which we play the hard instance for width $k-1$ in that branch, scaled by a sufficiently small constant. We ensure that at the end of a phase each branch contains a single leaf. When the algorithm moves to the other branch, we play phases there. We call a *super-phase* a maximal consecutive sequence of phases played in the same branch. The branch where the phases are played during a super-phase is called *active*, the other one *passive*. Note that the passive branch is just a single edge from the root to the unique leaf of the branch. Let opt_t denote the length of the passive branch during super-phase t, and define

$$x := \limsup_{t\to\infty} \frac{\mathrm{opt}_{t+1}}{\mathrm{opt}_t}.$$

We choose T such that $\frac{\mathrm{opt}_{T+1}}{\mathrm{opt}_T} \approx x$, and we stop after super-phase T is completed. We delete the leaf in the branch which was active in the last super-phase, so that the tree contains a single leaf in the end, connected to the root by an edge of length opt_T.

Online cost analysis

Since each sub-instance is scaled by a small enough constant, the cost that the algorithm can save by switching branches in the middle of a phase is negligible. Thus, denoting by A and P the active, respectively passive, branch in the last super-phase, and by applying the inductive hypothesis, we obtain that the algorithm pays cost at least $D_{k-1}\mathrm{opt}_{T+1}$ during the phases in A, and at least $D_{k-1}\mathrm{opt}_T$ during the phases in P. Moreover, the algorithm pays $\mathrm{opt}_{t+1} + \mathrm{opt}_t$ to switch branches at the end of each super-phase t. By the choice of x, we have $\mathrm{opt}_{T+1} \approx x\mathrm{opt}_T$ and $\mathrm{opt}_{t+1} \lesssim x\mathrm{opt}_t$ for all sufficiently large $t < T$. Therefore, the overall switching cost is

$$\sum_{t=1}^{T} \left(\mathrm{opt}_{t+1} + \mathrm{opt}_t\right) = \mathrm{opt}_{T+1} + 2\sum_{t=1}^{T} \mathrm{opt}_t \gtrsim \left(x + 2\sum_{t=0}^{T-1} \frac{1}{x^t}\right) \mathrm{opt}_T.$$

As we can make T arbitrarily large, we have

$$\sum_{t=0}^{T-1} \frac{1}{x^t} \approx \sum_{t=0}^{\infty} \frac{1}{x^t} = \frac{x}{x-1},$$

so the overall switching cost is at least roughly $\left(x + \frac{2x}{x-1}\right)\mathrm{opt}_T = \frac{x(x+1)}{x-1}\mathrm{opt}_T$. In total, the algorithm pays at least roughly

$$D_{k-1}\mathrm{opt}_{T+1} + D_{k-1}\mathrm{opt}_T + \frac{x(x+1)}{x-1}\mathrm{opt}_T \approx \left((x+1)D_{k-1} + \frac{x(x+1)}{x-1}\right)\mathrm{opt}_T.$$

This expression is minimized for $x = 1 + \sqrt{\frac{2}{1+D_{k-1}}}$, which yields the lower bound by Lemma 8 below.

To formalize the proof, we must show that it is indeed possible to scale down sub-instances in order to make the cost incurred in an individual sub-instance negligible. To this end, we need to upper bound opt_T. Also note that scaling a sub-instance might affect the algorithm's behavior (and thus the structure of the sub-instance, as it depends on the algorithm's behavior).

3 Notation

In this section, we define notation which will be useful to describe and analyze our algorithms.

We refer to an edge-weighted stemmed binary tree simply as "tree". If a tree S contains a single leaf, we call it trivial and we represent it by an edge $e(S)$ of length $w_{e(S)}$. Otherwise, we call it non-trivial and we represent it by an edge $e(S)$ of length $w_{e(S)}$ connected to two subtrees LS and RS (see Fig. 1). We say that LS is the sibling subtree of RS and vice-versa, and we say that S is the parent subtree of LS and RS. Additionally, we say that S' is a subtree of S if $S' = S$ or S' is a subtree of LS or RS.

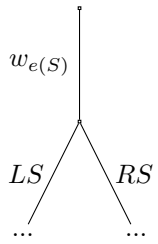


Figure 1 Non-trivial stemmed tree S.

Denoting by T the tree on which the evolving tree game is played, we define the depth of a node as the number of edges on the path connecting that node and the root of T, and define the depth of T as the maximum depth of one of its nodes. As in [10], we parametrize the evolving tree game by depth instead of width in sections 4 and 5 concerned with our main algorithm. We say that an instance has depth k if the depth of T is at most k at all times. Since the depth of a binary stemmed tree is smaller than or equal to its width, any instance of width k has depth at most k.[4] Since the parameter k is not known in advance, throughout the execution of the algorithm we denote by k the maximal depth of T since the beginning of the game.

We associate to each subtree S of T a level between 1 and k: if the depth of S's root is d, then S has level $k - d$. In particular, the level of T is k. By abuse of notation, we also associate levels to edges, so that $e(S)$ has the same level as S. Note that the level of a subtree S can increase in two ways: 1) if k is incremented (because the depth of T reached a new maximum); 2) if the smoothing that follows a deletion causes a decrease in the depth of S's root. In either case, we say that S is "promoted" to a higher level.

For notational convenience we often treat trees like sets of points, and hence use element and set difference notation ("$\in$" and "$\setminus$") accordingly. Additionally, we use $x \wedge y$ and $x \vee y$ as a shorthand for $\min\{x, y\}$ and $\max\{x, y\}$, respectively, and we assume that multiplication and addition take precedence over these operators.

We define OPT_S to be the shortest path from the root of S to one of its leaves. So

$$\mathrm{OPT}_S = w_{e(S)} + (\mathrm{OPT}_{LS} \wedge \mathrm{OPT}_{RS}).$$

We refer to our algorithm by ALG, and we denote the cost incurred by the algorithm so far by cost(ALG). To simplify notation, we assume that each tree has a distinguished leaf which describes ALG's location, so we can encode the full state of the game in a tree.

[4] On the other hand, an instance of depth k can have width up to 2^{k-1}, so our upper bounds hold even for certain instances of much larger width.

Finally, we define

$$x_k := 1 + \sqrt{\frac{2}{1 + D_{k-1}}} \qquad \text{for } k \geq 2, \tag{2}$$

and note the following lemma, which we will use repeatedly throughout the rest of the paper.

▶ **Lemma 8.** *For D_k and x_k as defined in (1) and (2) and $k \geq 2$, it holds that*

$$D_k = D_{k-1}(1 + x_k) + \frac{x_k(x_k + 1)}{x_k - 1}. \tag{3}$$

Proof.

$$\begin{aligned} &D_{k-1}(1 + x_k) + \frac{x_k(x_k + 1)}{x_k - 1} \\ &\qquad = D_{k-1}\left(2 + \sqrt{\frac{2}{1 + D_{k-1}}}\right) + 3 + \sqrt{\frac{2}{1 + D_{k-1}}} + \sqrt{2\,(1 + D_{k-1})} \\ &\qquad = 2D_{k-1} + \sqrt{8\,(1 + D_{k-1})} + 3 = D_k. \end{aligned}$$

◀

4 An approach that almost works

In this section, we use the lower bound as inspiration to derive an algorithm and a potential function which can *almost* be used to prove D_k-competitiveness. We parametrize the evolving tree game by depth instead of width, as the depth does not exceed the width.

Note that the lower bound is tight on the constructed instance when the algorithm applies the following strategy: Out of the two subtrees of the root, the algorithm stays in the current subtree as long as its optimum is not more than x_k times larger than the optimum of the other subtree. When this condition ceases to hold, the algorithm moves to an optimal leaf in the other subtree. To move within a subtree, the algorithm applies the recursive strategy for depth $k - 1$.

Let us analyze the performance of this naive algorithm. Given the current state of the tree T (which includes the location of ALG), we want to devise a pseudo-cost potential function $\Phi(T)$ which we can use to upper bound the cost incurred so far. If we could also show that $\Phi(T) \leq D_k \mathrm{OPT}_T$, this would mean that our algorithm achieves precisely the competitive ratio D_k for the depth-k evolving tree game.

4.1 Potential derivation

To derive $\Phi(T)$, we follow the intuition from the lower bound construction. The edge $e(T)$ could have been spawned by playing the game with depth k and then contracting everything into a single edge; during this time, ALG could have paid up to $D_k w_{e(T)}$.

Next, we bound the cost for switching between LT and RT. To simplify our reasoning, suppose that the adversary is only allowed to grow the leaf where the algorithm is currently located. Further suppose that this leaf first grows by the maximal amount for which ALG stays at this leaf, and then it grows by a very small (negligible) additional amount which triggers a switch to a different leaf. Therefore, ALG moves from LT to RT when ALG is in the optimal leaf in LT and the adversary grows this leaf so that $\mathrm{OPT}_{LT} = x_k \mathrm{OPT}_{RT}$. The algorithm pays $\mathrm{OPT}_{LT} + \mathrm{OPT}_{RT}$ for switching, of which OPT_{LT} is for backtracking to the shared root of LT and RT and OPT_{RT} for reaching the optimum leaf in RT. Suppose

ALG is currently located in RT. Then, OPT_{LT} stayed the same since the last switch, and we can deduce that the last switch cost $\mathrm{OPT}_{LT}(1 + 1/x_k)$, the previous switch cost $\mathrm{OPT}_{LT}(1/x_k + 1/{x_k}^2)$, and so on. The total switching cost can thus be bounded by

$$\sum_{i=0}^{\infty} \left(\frac{1}{{x_k}^i} + \frac{1}{{x_k}^{i+1}} \right) \mathrm{OPT}_{LT} = \frac{x_k + 1}{x_k - 1} \mathrm{OPT}_{LT}.$$

Of course, we swap the roles of LT and RT if ALG is currently in LT instead of RT. It remains to bound the cost paid while playing the game at depth $k-1$ in the two subtrees LT and RT. To this end, we apply an inductive argument to bound this cost by $\Phi_{k-1}(LT)$ and $\Phi_{k-1}(RT)$, respectively.

Putting everything together, for a non-trivial subtree S of level i, we define

$$\Phi_i(S) = D_i w_{e(S)} + \frac{x_i + 1}{x_i - 1} \mathrm{OPT}^{\mathrm{other}}(S) + \Phi_{i-1}(LS) + \Phi_{i-1}(RS), \tag{4}$$

where

$$\mathrm{OPT}^{\mathrm{other}}(S) = \begin{cases} \mathrm{OPT}_{LS} & \text{if ALG is in } RS, \\ \mathrm{OPT}_{RS} & \text{if ALG is in } LS, \\ \mathrm{OPT}_{LS} \vee \mathrm{OPT}_{RS} & \text{if ALG is not in } S. \end{cases} \tag{5}$$

Note that, if ALG is not in S, we defined $\mathrm{OPT}^{\mathrm{other}}(S) = \mathrm{OPT}_{LS} \vee \mathrm{OPT}_{RS}$ by following the intuition that the last visited leaf in S was the optimal one.

If S is trivial, we simply define $\Phi_i(S) = D_i w_{e(S)}$. We may omit the subscript in Φ if it is clear from the context.

4.2 Analysis sketch without deletion

Bounding the potential. Assuming that

$$\mathrm{OPT}_{LS} \vee \mathrm{OPT}_{RS} \leq x_i(\mathrm{OPT}_{LS} \wedge \mathrm{OPT}_{RS}), \tag{6}$$

a property which is indeed maintained by ALG's response to growth operations, we can prove by a simple inductive argument that $\Phi_i(S) \leq D_i \mathrm{OPT}_S$ for all subtrees S at level i. In particular, this implies $\Phi_k(T) \leq D_k \mathrm{OPT}_T$, as desired.

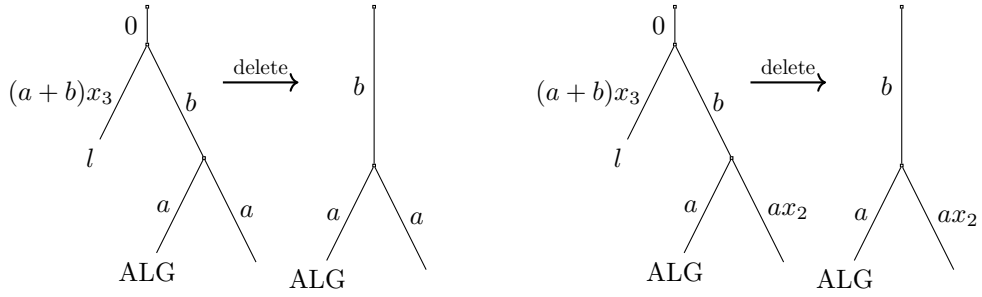
To show that $\mathrm{cost}(\mathrm{ALG}) \leq \Phi_k(T)$, it suffices to prove that the cost paid on each operation is bounded by the increase in the potential after that operation.

Growth. Suppose the adversary grows the leaf l where ALG is located. By unfolding the recursive formula for $\Phi(T)$, it is easy to see that it contains a term of the form $D_i w_{e(l)}$, where $w_{e(l)}$ is the length of the edge adjacent to l. Therefore, if ALG stays in l, the term $\Phi(T)$ increases by at least the cost paid by ALG. Otherwise, if ALG moves to a different leaf l', and if we denote by S the smallest subtree containing both l and l', then one can show that $\mathrm{OPT}^{\mathrm{other}}(S)$ increases enough to pay for the movement cost.

Fork. Since the new edges created by a fork initially have length 0, it is easy to see that a fork does not change $\Phi(T)$ or $\mathrm{cost}(\mathrm{ALG})$.

4.3 Why deletion causes problems

When deleting a leaf l together with its incident edge, the edges in the sibling subtree of l get "promoted" to a higher level. Although the promotion helps increase the potential, it may not be enough to counteract the edge deletion, so $\Phi(T)$ may decrease, as it happens in the example in Fig. 2a. The decrease of $\Phi(T)$ is problematic, as it may break the inequality $\text{cost}(\text{ALG}) \leq \Phi(T)$.



(a) Delete l in T_1 to obtain T_1'.

(b) Delete l in T_2 to obtain T_2'.

Figure 2 Deletion examples. ALG represents the position of the algorithm and the labels next to the edges indicate weights.

$$\Phi(T_1) = D_2(a+b)x_3 + D_2 b + 2D_1 a + \frac{x_3+1}{x_3-1}(a+b)x_3 + \frac{x_2+1}{x_2-1}a$$
$$\Phi(T_1') = D_3 b + 2D_2 a + \frac{x_3+1}{x_3-1}a$$
$$\Phi(T_1') - \Phi(T_1) = \left(2D_2 + \frac{x_3+1}{x_3-1} - D_2 x_3 - 2D_1 - \frac{x_3(x_3+1)}{x_3-1} - \frac{x_2+1}{x_2-1}\right)a \approx -2a < 0.$$

On the other hand, the potential may increase too much after deletion, violating the inequality $\Phi(T) \leq D_k \text{OPT}_T$. Recall that this inequality is satisfied if (6) holds for all subtrees S at level i. However, if some S is promoted from level i to $i+1$, (6) may no longer be satisfied, as $x_{i+1} < x_i$. This happens in the example in Fig. 2b, where $\text{OPT}_{RT_2'} > x_3 \text{OPT}_{LT_2'}$ (because $ax_2 > ax_3$).

So what went wrong with our derivation of the potential? Note that we implicitly assumed that deletion always occurs in a leaf whose sibling is also a leaf, so only trivial subtrees get promoted. Indeed, the lower bound instance is restricted to this kind of "easy" deletions, and it can be proved that the potential is well-behaved on "easy" deletions. By contrast, in both examples in Fig. 2 we delete a leaf whose sibling is a non-trivial subtree. Indeed, the core difficulty of the problem lies in handling deletions in the general case.

5 Refining the algorithm

We now provide an algorithm and analysis that achieves competitive ratio $O(2^k)$, proving Theorem 1. It is based on two refinements, *forgetting* and *imbalancing*, that make the analysis better aligned with the structure of hard instances. Conceptually, forgetting corresponds to *temporal* canonicalization by limiting the effect of information that is not currently relevant, while imbalancing achieves *geometric* canonicalization by reshaping the instance towards a worst-case form.

Forgetting

Forgetting stabilizes the potential by truncating certain terms, effectively redefining it as if some parts of the tree had not yet been revealed. In the example in Fig. 2b, we could pretend that the edge in RT_2' had length ax_3 instead of ax_2, obtaining the tree T'' from Fig. 3, which satisfies $\Phi(T'') = D_3\mathrm{OPT}_{T''}$. Intuitively, this corresponds to *rewinding* part of the exploration. One could extend the evolving tree game by introducing a new operation in which the algorithm could choose a subtree S and some $d > 0$, and forget everything in S at distance more than d from the root of S. However, this would make the algorithm too powerful, as repeated forgetting could prevent the adversary from growing the tree. Instead, we simulate the same effect analytically by capping certain terms in the potential function, ensuring bounded growth after deletions.

Imbalancing

Imbalancing enforces a structural regularity that serves to better align an instance with typical worst-case scenarios. It is based on the observation that imbalanced trees act as canonical hard instances, since for such trees the potential achieves its maximum value $\Phi_i(S) = D_i\mathrm{OPT}_S$. To move toward this canonical form, we slightly distort some edge lengths to create a controlled imbalance. In the example in Fig. 2a, distorting the rightmost edge in T_1' by a factor of x_3 yields the tree T'' from Fig. 3, satisfying $\Phi(T'') = D_3\mathrm{OPT}_{T''} = D_3\mathrm{OPT}_{T_1}$. More generally, one can show that if a tree is very imbalanced, then it is an extreme tree (i.e., a tree S for which the pseudo-cost $\Phi_i(S)$ attains the upper bound of $D_i\mathrm{OPT}_S$, reflecting that it is a hard instance).

To achieve a high imbalance, we can perform the following operation: *scale* the length of an edge e by a factor $f \geq 1$. If T^0 is the tree on which the game is played, we maintain a distorted tree T which has the same vertices and edges as T^0, but different edge weights. When the adversary performs some operation on T^0, we perform the same operation on T. Additionally, we may perform imbalancing operations on T. Provided the overall distortion is not too large, this only loses a constant factor.

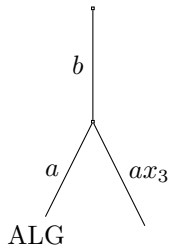


Figure 3 Extreme tree T''. ALG represents the position of the algorithm and the labels next to the edges indicate weights.

5.1 Algorithm description

We are now ready to describe our algorithm. Recall that the tree T^0 on which the game is played is initialized to a have a single leaf, connected to the root by a zero-length edge. We initialize the distorted tree T in the same manner. Since the depth k of the instance is not known in advance, we initialize $k := 1$, and we increment k when the depth of T exceeds the current value of k.

We denote by w^0 and w the edge weights in T^0 and T, respectively. We denote by S^0 a subtree of T^0, and by S the subtree corresponding to S^0 in T.

We will ensure that the following invariant always holds.

Ratio invariant. Let l_{ALG} be the leaf where ALG is located, and let S be a non-trivial subtree of T of level i such that $l_{\mathrm{ALG}} \in S$. Then, if $l_{\mathrm{ALG}} \in LS$ we have $\mathrm{OPT}_{LS} \leq x_i \mathrm{OPT}_{RS}$, and if $l_{\mathrm{ALG}} \in RS$ we have $\mathrm{OPT}_{RS} \leq x_i \mathrm{OPT}_{LS}$.

Based on the operation performed by the adversary on T^0, we describe our algorithm's response on T.

Growth. When the adversary grows a leaf and the ratio invariant remains satisfied, ALG stays in its current position. Otherwise, let S be the highest-level subtree for which the ratio invariant inequality is violated. Then ALG moves to the optimal leaf in S. Observe that the ratio invariant is now satisfied.

Deletion. When the adversary deletes a leaf l, we proceed as follows:

1. Pretend that the length of $e(l)$ grows to ∞, moving accordingly.
2. Let S be the parent subtree of l. If ALG is in S, ALG moves to an optimal leaf in S.
3. Delete l and its incident edge.
4. Transform S into an extreme tree (see Section 5.3.3).

Fork. If ALG is located in the leaf l which is forked, ALG moves to any of the new leaves. Additionally, if the depth of the tree becomes larger than k, then we increment k, ALG moves to the optimal leaf, and we transform T into an extreme tree.

5.2 Potential

We now show how to adapt the potential in order to "forget" parts of the tree.

For a non-trivial subtree S of T of level i, define

$$\Phi_i(S) = D_i w_{e(S)} + \frac{x_i + 1}{x_i - 1} \overline{\mathrm{OPT}^{\mathrm{other}}(S)} + \overline{\Phi_{i-1}(LS)} + \overline{\Phi_{i-1}(RS)}, \tag{7}$$

where

$$\overline{\mathrm{OPT}^{\mathrm{other}}(S)} = \mathrm{OPT}^{\mathrm{other}}(S) \wedge x_i(\mathrm{OPT}_{LS} \wedge \mathrm{OPT}_{RS}) \tag{8}$$

$$\overline{\Phi_{i-1}(XS)} = \Phi_{i-1}(XS) \wedge D_{i-1} x_i \, (\mathrm{OPT}_{LS} \wedge \mathrm{OPT}_{RS}) \quad \text{for } X \in \{L, R\}. \tag{9}$$

For a trivial subtree S, define $\Phi_i(S) = D_i w_{e(S)}$.

Note that, as long as $\mathrm{OPT}_{LS} \vee \mathrm{OPT}_{RS} \leq x_i(\mathrm{OPT}_{LS} \wedge \mathrm{OPT}_{RS})$, by Claim 10 below we have

$$\overline{\mathrm{OPT}^{\mathrm{other}}(S)} = \mathrm{OPT}^{\mathrm{other}}(S), \quad \overline{\Phi_{i-1}(RS)} = \Phi_{i-1}(RS), \quad \text{and} \quad \overline{\Phi_{i-1}(LS)} = \Phi_{i-1}(LS),$$

so we recover the same recursive formula for $\Phi_i(S)$ as in the previous version of the potential in (4). Otherwise, if, say, $\mathrm{OPT}_{LS} > x_i \mathrm{OPT}_{RS}$, the refined potential caps some terms – as if "forgetting" some part of LS so that OPT_{LS} gets reduced to $x_i \mathrm{OPT}_{RS}$.

It is also important to note that the subtrees which contain ALG's location will not have their potential capped, as shown in the following claim.

▷ Claim 9. For all subtrees S at level $i < k$ such that ALG is in S, we have $\overline{\Phi_i(S)} = \Phi_i(S)$.

Proof. Let P be the parent subtree of S and suppose without loss of generality that $S = LP$. By Claim 10 below, we have $\Phi_i(LP) \leq D_i \mathrm{OPT}_{LP}$, and by the ratio invariant we have $\mathrm{OPT}_{LP} \leq x_{i+1}\mathrm{OPT}_{RP}$. Thus, $\Phi_i(LP) \leq x_{i+1}D_i(\mathrm{OPT}_{LP} \wedge \mathrm{OPT}_{RP})$, so $\overline{\Phi_i(LP)} = \Phi_i(LP)$. ◁

5.3 Analysis

We will ensure that each edge in the distorted tree T is scaled by a factor between 1 and some constant C. Thus, $\mathrm{OPT}_T \leq C \cdot \mathrm{OPT}_{T^0}$. Moreover, when ALG moves from a leaf to another, we will charge the movement cost according to the distances in T instead of the distances in T^0, which means that we are overestimating the cost paid.

In the rest of this section we will prove that

$$\mathrm{cost}(\mathrm{ALG}) \leq \Phi(T) \leq D_k \mathrm{OPT}_T,$$

which together with the previous observations and the fact that $D_k = O(2^k)$ (as shown in Section 5.3.6) yields

$$\mathrm{cost}(\mathrm{ALG}) \leq C \cdot D_k \cdot \mathrm{OPT}_{T^0} = O(2^k) \cdot \mathrm{OPT}_{T^0}.$$

We denote by $\Delta\mathrm{cost}(\mathrm{ALG})$ the cost paid in the current operation and by $\Delta\Phi(T)$ the increase of the potential in the current operation. To ensure that $\mathrm{cost}(\mathrm{ALG}) \leq \Phi(T)$, it suffices to show that $\Delta\mathrm{cost}(\mathrm{ALG}) \leq \Delta\Phi(T)$ holds for all operations.

For two leaves $l, l' \in S$, we denote by $\mathrm{dist}_S(l, l')$ the distance between l and l' in subtree S. We denote by l_{OPT_S} the optimal leaf in S (breaking ties arbitrarily).

5.3.1 Bounding the potential

▷ Claim 10. For any subtree S of level i, it holds that $\Phi_i(S) \leq D_i \mathrm{OPT}_S$.

Proof. By induction. For a trivial subtree S, we have $\Phi_i(S) = D_i w_{e(S)} = D_i \mathrm{OPT}_S$. For the inductive step, suppose without loss of generality that $\mathrm{OPT}_{LS} \leq \mathrm{OPT}_{RS}$, so $\mathrm{OPT}_S = w_{e(S)} + \mathrm{OPT}_{LS}$. By the inductive hypothesis, $\Phi_{i-1}(LS) \leq D_{i-1}\mathrm{OPT}_{LS}$. Using this together with (8) and (9), we get

$$\begin{aligned}
\Phi_i(S) &= D_i w_{e(S)} + \frac{x_i + 1}{x_i - 1}\overline{\mathrm{OPT}^{\mathrm{other}}(S)} + \overline{\Phi_{i-1}(LS)} + \overline{\Phi_{i-1}(RS)} \\
&\leq D_i w_{e(S)} + \frac{x_i + 1}{x_i - 1} x_i \mathrm{OPT}_{LS} + D_{i-1}\mathrm{OPT}_{LS} + D_{i-1}x_i\mathrm{OPT}_{LS} \\
&= D_i(w_{e(S)} + \mathrm{OPT}_{LS}) = D_i \mathrm{OPT}_S,
\end{aligned}$$

where in the penultimate step we plugged in the formula for D_i. ◁

In the next sections, we show that $\Delta\mathrm{cost}(\mathrm{ALG}) \leq \Delta\Phi(T)$ holds for all possible operations and that the ratio invariant is maintained.

5.3.2 Growth

By the definition of the algorithm, one can see that the ratio invariant is maintained. It remains to show that the potential does not decrease after a growth operation. We first prove this in the following case.

▷ Claim 11. Suppose that the adversary grows a leaf l_g by h, and ALG is located at l_g before the growth operation. Then, $\Delta\text{cost}(\text{ALG}) \leq \Delta\Phi(T)$.

Proof. If the algorithm remains at l_g after the growth, since $w_{e(l_g)}$ increases by h, we have $\Delta\Phi(S_{l_g}) = D_i h$, where S_{l_g} is the trivial subtree containing l_g and i is its level. By repeatedly applying Claim 9, we obtain

$$\Delta\Phi(T) \geq D_i h \geq h = \Delta\text{cost}(\text{ALG}).$$

It remains to consider the case when ALG moves from its initial position l_g to another leaf. Let S be the highest-level subtree before the growth for which the ratio invariant inequality in the corresponding subtree S' after growth is violated, and let i be the level of S and S'. Suppose without loss of generality that ALG is currently at $l_g \in LS$ and moves to an optimal leaf in $RS' = RS$. Note that this movement can only be triggered when l_g is the (unique) optimum leaf in LS, and therefore the movement cost is $\text{OPT}_{LS} + \text{OPT}_{RS}$.

Since the ratio invariant is satisfied before the growth but violated after the growth, we have

$$\text{OPT}_{LS} \leq x_i \text{OPT}_{RS} \tag{10}$$

$$\text{OPT}_{LS'} > x_i \text{OPT}_{RS}. \tag{11}$$

Therefore, before the growth we have

$$\overline{\text{OPT}^{\text{other}}(S)} \leq \text{OPT}^{\text{other}}(S) = \text{OPT}_{RS},$$

and after the growth, by (11), we have

$$\overline{\text{OPT}^{\text{other}}(S')} = \text{OPT}_{LS'} \wedge (x_i \text{OPT}_{RS}) = x_i \text{OPT}_{RS}.$$

Therefore, $\overline{\text{OPT}^{\text{other}}(S)}$ increases by at least $(x_i - 1)\text{OPT}_{RS}$. Notice that no term in $\Phi(T)$ decreases after the growth, so by repeatedly applying Claim 9, we have

$$\begin{aligned}\Delta\Phi(T) \geq \Delta\Phi(S) &\geq \frac{x_i + 1}{x_i - 1}(x_i - 1)\text{OPT}_{RS} \\ &= (x_i + 1)\text{OPT}_{RS} \\ &\overset{(10)}{\geq} \text{OPT}_{RS} + \text{OPT}_{LS} = \Delta\text{cost}(\text{ALG}).\end{aligned}$$ ◁

The following claim provides a stronger bound on the potential, and will be used repeatedly in our analysis.

▷ Claim 12. Let l_{ALG} be the leaf where ALG is located, let S be a subtree of level i such that $l_{\text{ALG}} \in S$, and let l_{OPT_S} be an optimal leaf in S. Then,

$$\Phi_i(S) + \text{dist}_S(l_{\text{ALG}}, l_{\text{OPT}_S}) \leq D_i \text{OPT}_S.$$

Proof. Imagine repeatedly growing the leaf where ALG is located until eventually ALG moves to l_{OPT_S}. By Claim 11, by the time ALG reaches l_{OPT_S}, $\Phi(T)$ would have increased by at least the movement cost paid by ALG, which is at least $\text{dist}_S(l_{\text{ALG}}, l_{\text{OPT}_S})$ by the triangle inequality. Moreover, the potential at the end of this process would still be bounded by $D_i \text{OPT}_S$ by Claim 10, which concludes the proof. ◁

We next prove the following auxiliary claim, which will be used to show that the potential does not decrease after any growth operation.

▷ Claim 13. Suppose that in response to a growth operation ALG moves from its initial position $l_{\text{ALG}} \in LS$ to an optimal leaf in RS. Let P be a subtree of LS such that $l_{\text{ALG}} \in P$. Then, the change in potential after the growth and ALG's movement satisfies

$$\Delta\overline{\Phi(P)} \geq A_P - \text{OPT}_P,$$

where $A_P = \text{dist}_P(\text{root}_P, l_{\text{ALG}})$ and root_P denotes the root of P.

Proof. By induction. If P is a trivial subtree, then $A_P - \text{OPT}_P = 0 \leq \Delta\overline{\Phi(P)}$ and we are done. Thus, suppose P is a non-trivial subtree of level i, and let P' denote the subtree P after the growth. We begin by showing that it suffices to prove that $\Delta\Phi(P) \geq A_P - \text{OPT}_P$. By Claim 12, we have

$$\overline{\Phi(P)} \leq \Phi(P) \leq D_i\text{OPT}_P - \text{dist}(l_{\text{ALG}}, l_{\text{OPT}_P}). \tag{12}$$

By the triangle inequality,

$$\begin{aligned}\text{dist}(l_{\text{ALG}}, l_{\text{OPT}_P}) &\geq \text{dist}(\text{root}_P, l_{\text{ALG}}) - \text{dist}(\text{root}_P, l_{\text{OPT}_P}) \\ &= A_P - \text{OPT}_P.\end{aligned}$$

Plugging this into (12), we get

$$\overline{\Phi(P)} \leq \Phi(P) \leq D_i\text{OPT}_P - A_P + \text{OPT}_P. \tag{13}$$

Let Q be the sibling of P, and Q' be the subtree Q after the growth. Since the ratio invariant was satisfied before the growth, we have

$$x_{i+1}\text{OPT}_{Q'} \geq x_{i+1}\text{OPT}_Q \geq \text{OPT}_P,$$

and substituting into (9) yields

$$\overline{\Phi(P')} \geq \Phi(P') \wedge D_i\text{OPT}_P.$$

Combining this with (13), we get

$$\Delta\overline{\Phi(P)} \geq (A_P - \text{OPT}_P) \wedge \Delta\Phi(P).$$

Therefore, it suffices to show that $\Delta\Phi(P) \geq A_P - \text{OPT}_P$. Suppose without loss of generality that $l_{\text{ALG}} \in LP$. By the inductive hypothesis,

$$\Delta\overline{\Phi(LP)} \geq A_{LP} - \text{OPT}_{LP}. \tag{14}$$

If OPT_P is in LP, then $A_P - \text{OPT}_P = A_{LP} - \text{OPT}_{LP}$, and we can conclude by substituting this into (14). Therefore, suppose that l_{OPT_P} is in RP. Then,

$$A_P - \text{OPT}_P = (A_{LP} - \text{OPT}_{LP}) + (\text{OPT}_{LP} - \text{OPT}_{RP}). \tag{15}$$

The first term is bounded by (14), and we will show that the second term is bounded by $\Delta\overline{\text{OPT}^{\text{other}}(P)}$. To this end, note that

$$\overline{\text{OPT}^{\text{other}}(P)} \leq \text{OPT}^{\text{other}}(P) = \text{OPT}_{RP}$$

and

$$\overline{\text{OPT}^{\text{other}}(P')} = (\text{OPT}_{LP'} \vee \text{OPT}_{RP'}) \wedge (x_i(\text{OPT}_{LP'} \wedge \text{OPT}_{RP'})) \geq \text{OPT}_{LP},$$

since $\mathrm{OPT}_{LP'} \geq \mathrm{OPT}_{LP}$ and $x_i\mathrm{OPT}_{RP'} \geq x_i\mathrm{OPT}_{RP} \geq \mathrm{OPT}_{LP}$ by the ratio invariant. Thus,

$$\Delta\overline{\mathrm{OPT}^{\mathrm{other}}(P)} \geq \mathrm{OPT}_{LP} - \mathrm{OPT}_{RP}. \tag{16}$$

Combining (16), (15), (14), and (7), and noting that $\overline{\mathrm{OPT}^{\mathrm{other}}(P)}$ has coefficient $\frac{x_i+1}{x_i-1} > 1$, we get

$$\Delta\Phi(P) \geq A_P - \mathrm{OPT}_P,$$

as desired. ◁

We are now ready to prove that potential does not decrease after any growth operation.

▷ Claim 14. For any growth operation, we have $\Delta\mathrm{cost}(\mathrm{ALG}) \leq \Delta\Phi(T)$.

Proof. Suppose that the adversary grows a leaf l_g by h, and ALG is located at l_{ALG} before the growth operation. The case when $l_{\mathrm{ALG}} = l_g$ is already covered by Claim 11, so it remains to consider the case when $l_{\mathrm{ALG}} \neq l_g$.

If the algorithm does not move and stays in the same leaf l_{ALG} before and after the growth, we have $\Delta\Phi(T) \geq 0 = \Delta\mathrm{cost}(\mathrm{ALG})$ and we are done. Therefore, suppose ALG moves from its initial position l_{ALG} to another leaf. Let S be the highest-level subtree before the growth for which the ratio invariant inequality in the corresponding subtree S' after growth is violated, and let i be the level of S and S'. Suppose without loss of generality that ALG is currently in $l_{\mathrm{ALG}} \in LS$ and moves to an optimal leaf in $RS' = RS$. Note that this movement can only be triggered when the growing leaf l_g is the (unique) optimum leaf in LS. Let A_{LS} be the distance from l_{ALG} to the root of subtree LS, so that ALG pays $A_{LS} + \mathrm{OPT}_{RS}$ for the movement. By the auxiliary Claim 13,

$$\Delta\overline{\Phi(LS)} \geq A_{LS} - \mathrm{OPT}_{LS}. \tag{17}$$

Similarly to the proof of Claim 11, we have that $\overline{\mathrm{OPT}^{\mathrm{other}}(S)}$ increases by at least $(x_i - 1)\mathrm{OPT}_{RS}$. Notice that no term in $\Phi(T)$ decreases after the growth, so by repeatedly applying Claim 9, and using (17), we have

$$\begin{aligned}\Delta\Phi(T) \geq \Delta\Phi(S) &\geq \frac{x_i+1}{x_i-1}(x_i-1)\mathrm{OPT}_{RS} + A_{LS} - \mathrm{OPT}_{LS} \\ &= (x_i+1)\mathrm{OPT}_{RS} + A_{LS} - \mathrm{OPT}_{LS}\end{aligned}$$

Since the ratio invariant was satisfied before the growth, we have $\mathrm{OPT}_{LS} \leq x_i\mathrm{OPT}_{RS}$, and thus

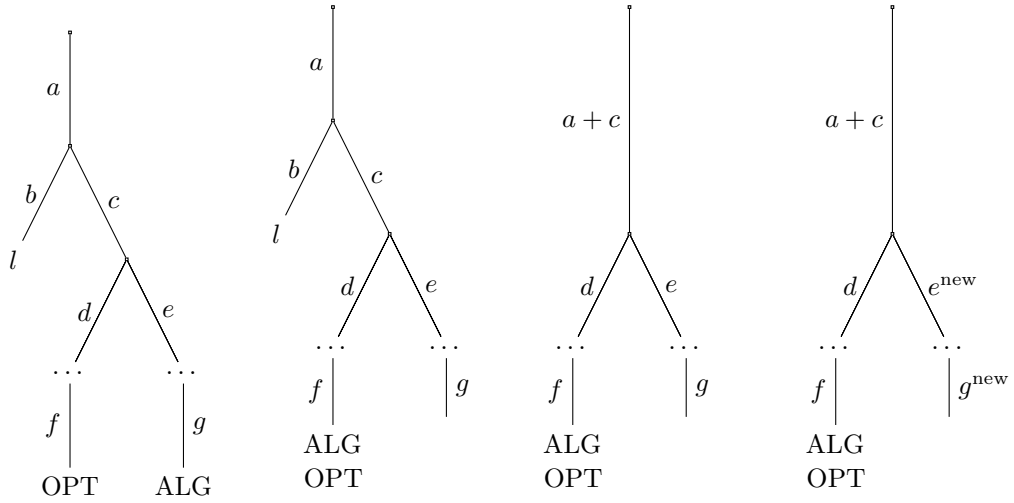
$$\Delta\Phi(T) \geq \mathrm{OPT}_{RS} + A_{LS} = \Delta\mathrm{cost}(\mathrm{ALG}).$$ ◁

5.3.3 Deletion

This section is concerned with the analysis for the deletion operation. Let l be the leaf deleted by the adversary. Without loss of generality, assume that $w_{e(l)} > 2\mathrm{OPT}_T$. If this is not the case, we can pretend that l grows by a sufficient amount just before the deletion is announced. Our assumption ensures that ALG cannot be located in l, by the ratio invariant.

Let S be the parent subtree of l, as shown in Fig. 4. Recall that we take the following steps in order to handle the deletion:

1. If ALG is in S, ALG moves to an optimal leaf in S. We denote the resulting subtree by S'.
2. Delete l together with its incident edge, and increase the levels of all remaining subtrees in S'. Call the resulting subtree S''.
3. Apply the extreme-imbalance procedure (described below) on S'' to obtain extreme subtree S'''.



(a) Subtree S. **(b)** Subtree S'. **(c)** Subtree S''. **(d)** Subtree S'''.

Figure 4 Stages of deletion. OPT denotes the location of the optimal leaf in the subtree and ALG denotes the location of the algorithm (if the algorithm is in the subtree). The edges are labelled by their weights.

Recall that a subtree N of level j is said to be extreme if the inequality in Claim 10 is tight, i.e. $\Phi_j(N) = D_j \mathrm{OPT}_N$. The following recursive procedure can be used to make a subtree extreme.

Extreme-imbalance procedure on a subtree N. If N is a trivial subtree, leave N unchanged. Otherwise, if N is a non-trivial subtree of level j, and assuming without loss of generality that the optimal leaf of N is in LN, do:

1. Apply the extreme-imbalance procedure to LN and RN.
2. If $\mathrm{OPT}_{RN} \le x_j \mathrm{OPT}_{LN}$, scale up all edges in RN by $x_j \frac{\mathrm{OPT}_{LN}}{\mathrm{OPT}_{RN}} \in [1, x_j]$.

▷ Claim 15. Assuming that ALG is located either in $T \setminus N$ or in the optimal leaf of N, the extreme-imbalance procedure transforms N into a subtree N' such that

- $\Phi_j(N') = D_j \mathrm{OPT}_{N'}$ (i.e., N' is extreme), and
- $\mathrm{OPT}_{N'} = \mathrm{OPT}_N$ (and more precisely, no edge on the optimal path is altered).

Proof. If N is trivial, then N is already extreme, so we are done. Therefore, suppose N is non-trivial. Since ALG is located either in the optimal leaf of LN or not in N at all, we can apply the procedure recursively to make LN and RN extreme. In the end, we have $\mathrm{OPT}_{LN'} = \mathrm{OPT}_{LN}$ and $\mathrm{OPT}_{RN'} \ge x_j \mathrm{OPT}_{LN}$. Putting everything together and using (7), a simple calculation shows that $\Phi_j(N') = D_j \mathrm{OPT}_N$, as desired. ◁

Let T^{new} be the tree obtained from T by replacing S with S'''. Let i be the level of S, S', S'', and S'''.

▷ Claim 16. We have $\Delta\Phi(T) = \Phi(T^{\text{new}}) - \Phi(T) \geq \Delta\text{cost}(\text{ALG})$.

Proof. Suppose that initially ALG is in a leaf $l_{\text{ALG}} \notin S$. Then, $\Delta\text{cost}(\text{ALG}) = 0$, since ALG does not move during any step. We have $\text{OPT}_S = \text{OPT}_{S'} = \text{OPT}_{S''}$, as the deleted leaf l was not optimal in S (by our assumption that $w_{e(l)} > 2\text{OPT}_T$). By Claim 15, we obtain

$$\Phi(S''') = D_i\text{OPT}_{S'''} = D_i\text{OPT}_S,$$

and by Claim 10 we have

$$\Phi(S) \leq D_i\text{OPT}_S.$$

Thus, $\Phi(S''') - \Phi(S) \geq 0$, and so $\Delta\Phi(T) \geq 0$.

Now suppose that initially ALG is in a leaf $l_{\text{ALG}} \in S$. Then, ALG pays $\Delta\text{cost}(\text{ALG}) = \text{dist}_S(l_{\text{ALG}}, l_{\text{OPT}_S})$ to move from l to the optimal leaf l_{OPT_S}. By Claim 12 we have

$$\Phi(S) + \text{dist}_S(l_{\text{ALG}}, l_{\text{OPT}_S}) \leq D_i\text{OPT}_S,$$

and by Claim 15 we have

$$\Phi(S''') = D_i\text{OPT}_{S'''} = D_i\text{OPT}_S.$$

Thus, $\Phi(S''') - \Phi(S) \geq \Delta\text{cost}(\text{ALG})$. Since ALG is initially in S and it remains in S''', by repeatedly applying Claim 9, we obtain $\Delta\Phi(T) = \Phi(S''') - \Phi(S) \geq \Delta\text{cost}(\text{ALG})$. ◁

▷ Claim 17. T^{new} satisfies the ratio invariant.

Proof. We know that the ratio invariant is satisfied in the beginning for T. ALG may move to an optimal leaf in S at step 1, but this maintains the invariant. The levels of the subtrees of S' increase at step 2, so the x_j's used for the ratio invariant inequality inside S' and S'' are different. But the ratio invariant inequality is satisfied trivially inside S'' because if ALG is in S'', then it is in an optimal leaf in S''. At step 3, the lengths of the edges on the path from the root to ALG's location do not change, whereas the lengths of other edges could only increase; thus, the invariant is preserved. ◁

5.3.4 Fork

If the fork increases the depth of T from k to $k+1$, we increment k. This increases the levels of all subtrees, which could make the ratio invariant become violated. To fix this, ALG moves from its current leaf l_{ALG} to an optimal leaf in T, and we apply the extreme-imbalance procedure on T to obtain a new tree T^{new}. The movement cost paid by ALG is $\Delta\text{cost}(\text{ALG}) = \text{dist}_T(l_{\text{ALG}}, l_{\text{OPT}_T})$, and by Claim 12 we know that

$$\Phi_k(T) + \text{dist}_T(l_{\text{ALG}}, l_{\text{OPT}_T}) \leq D_k\text{OPT}_T.$$

By Claim 15, $\text{OPT}_{T^{\text{new}}} = \text{OPT}_T$ and $\Phi_{k+1}(T^{\text{new}}) = D_{k+1}\text{OPT}_T > D_k\text{OPT}_T$, so we get that $\Phi(T^{\text{new}}) > \Phi(T) + \Delta\text{cost}(\text{ALG})$, as desired.

Otherwise, if the fork does not cause an increase of k, it is easy to see that cost(ALG) and $\Phi(T)$ stay the same, and the ratio invariant remains satisfied.

5.3.5 Bounding the distortion factor

In this section we bound the distortion in T.

▷ Claim 18. The distortion factor of a (non-zero length) edge e at level i satisfies

$$1 \leq \frac{w_e}{w_e^0} \leq \Pi_{j=2}^{i} x_{j+1} \dots x_k.$$

In particular, an edge e at level 1 is not distorted at all (i.e. $w_e = w_e^0$).

Proof. First note that growing e cannot increase w_e/w_e^0, since w_e and w_e^0 grow by the same amount.

It remains to bound how much e is scaled in the imbalancing operations. Note that edges at level 1 are never scaled: Edges scaled upon a deletion were promoted just before, so their level is greater than 1; if a fork step causes scaling, then the only level 1 edges are the two new edges of length 0, so scaling them has no effect. Thus, assume that $i > 1$. When we apply the extreme-imbalance procedure to a subtree S of level $j > i$ which contains e, e is scaled by at most x_j. Considering all of the recursive calls of the procedure, e is scaled by at most

$$x_j x_{j-1} \dots x_{i+1} \leq x_k x_{k-1} \dots x_{i+1}.$$

Since we apply the extreme-imbalance procedure only to subtrees whose level has just been increased, the level of e was at most $i-1$ when the previous scalings occurred, and we can conclude by an inductive argument. ◁

▷ Claim 19. Let $C := \max_e \frac{w_e}{w_e^0}$ be the distortion factor of T. Then $C < 60$.

Proof. By Claim 18,

$$C \leq (x_3 \dots x_k) \dots (x_{k-1} x_k) x_k = \Pi_{i=3}^{k} x_i^{i-2}.$$

Since $x_i = 1 + \sqrt{\frac{2}{1+D_{i-1}}}$ and $D_{i-1} > 2^{i+1}$ for $i \geq 5$, we have $x_i < 1 + 2^{-i/2}$. Using that $1 + r \leq e^r$ for $r \in \mathbb{R}$, we get $\ln(x_i) < 2^{-i/2}$. Thus,

$$\ln(C) \leq \sum_{i=3}^{k} (i-2) \ln(x_i) < \ln(x_3) + 2\ln(x_4) + \sum_{i=5}^{\infty} (i-2) 2^{-i/2} \approx 4.09,$$

so $C < 60$. ◁

5.3.6 Bounding D_k

Finally, we show that $D_k = O(2^k)$.

▷ Claim 20. For $k \geq 2$, it holds that $D_k \leq 2^{k+4} - \sqrt{2^{k+9}}$.

Proof. By induction. The base case $k = 2$ holds because $D_2 = 9 < 2^6 - \sqrt{2^{11}}$. For the inductive step, we plug in the formula for D_{k+1} and use the inductive hypothesis to obtain

$$D_{k+1} = 2D_k + \sqrt{8(1+D_k)} + 3 \leq 2^{k+5} - 2\sqrt{2^{k+9}} + \sqrt{8 \cdot 2^{k+4}} + 3.$$

To complete the proof, it suffices to show that

$$2\sqrt{2^{k+9}} \geq \sqrt{8 \cdot 2^{k+4}} + 3 + \sqrt{2^{k+10}}.$$

This is equivalent to

$$\sqrt{2^{k+7}}(3 - 2\sqrt{2}) \geq 3,$$

which holds for all $k \geq 2$. ◁

References

1 Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. In *Advances in Neural Information Processing Systems*, volume 34, pages 30504–30517, 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/ffeed84c7cb1ae7bf4ec4bd78275bb98-Paper.pdf.

2 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. In *International Conference on Machine Learning, ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 969–983, 2023. URL: https://proceedings.mlr.press/v202/antoniadis23b.html.

3 Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *ACM Transactions on Algorithms*, 19(2):1–34, 2023. doi:10.1145/3582689.

4 C. J. Argue, Anupam Gupta, Ziye Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *J. ACM*, 68(5):32:1–32:10, 2021. doi:10.1145/3450349.

5 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993. doi:10.1006/INCO.1993.1054.

6 Xingjian Bai, Christian Coester, and Romain Cosson. Unweighted layered graph traversal: Passing a crown via entropy maximization. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 3884–3900, 2025. doi:10.1137/1.9781611978322.131.

7 Anatole Beck and D. J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8(4):419–429, 1970. doi:10.1007/BF02798690.

8 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. doi:10.1007/BF01294260.

9 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

10 Sébastien Bubeck, Christian Coester, and Yuval Rabani. Shortest paths without a map, but with an entropic regularizer. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1102–1113, 2022. doi:10.1109/FOCS54457.2022.00036.

11 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC*, pages 581–594, 2023. doi:10.1145/3564246.3585132.

12 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. *SIAM J. Comput.*, 52(2):STOC19–339–STOC19–353, 2023. doi:10.1137/20M1312332.

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

14 Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64, 1991. doi:10.1090/DIMACS/007/02.

15 Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1136–1147, 2019. doi:10.1145/3313276.3316370.

16 Christian Coester and Tze-Yang Poon. Online 3-taxi on general metrics. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 6659–6673, 2026. doi:10.1137/1.9781611978971.238.

17 Christian Coester and Alexa Tudose. Chasing small sets optimally against adaptive adversaries. *arXiv preprint*, 2026. arXiv:2605.10927.

18 Romain Cosson and Laurent Massoulié. Asynchronous collective tree exploration: a distributed algorithm, and a new lower bound. *arXiv preprint arXiv:2507.15658*, 2025. doi:10.48550/arXiv.2507.15658.

19 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.

20 J. Friedman and N. Linial. On convex body chasing. *Discrete and computational geometry*, 9(3):293–322, 1993. doi:10.1007/BF02189324.

21 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

22 Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991. doi:10.1016/0304-3975(91)90263-2.

23 H. Ramesh. On traversing layered graphs on-line. *J. Algorithms*, 18(3):480–512, 1995. doi:10.1006/JAGM.1995.1019.

24 Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1509–1518, 2020. doi:10.1137/1.9781611975994.92.