Dynamic Algorithm for Explainable k-medians Clustering under ℓ_p Norm

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study the problem of explainable k-medians clustering introduced by Dasgupta, Frost, Moshkovitz, and Rashtchian (2020). In this problem, the goal is to construct a threshold decision tree that partitions data into k clusters while minimizing the k-medians objective. These trees are interpretable because each internal node makes a simple decision by thresholding a single feature, allowing users to trace and understand how each point is assigned to a cluster.

We present the first algorithm for explainable k-medians under ℓ_p norm for every finite $p \geq 1$. Our algorithm achieves an $\tilde{O}\big(p(\log k)^{1+1/p-1/p^2}\big)$ approximation to the optimal k-medians cost for any $p \geq 1$. Previously, algorithms were known only for $p = 1$ and $p = 2$. For $p = 2$, our algorithm improves upon the existing bound of $\tilde{O}(\log^{3/2} k)$, and for $p = 1$, it matches the tight bound of $\log k + O(1)$ up to a multiplicative $O(\log \log k)$ factor.

We show how to implement our algorithm in a dynamic setting. The dynamic algorithm maintains an explainable clustering under a sequence of insertions and deletions, with amortized update time $O(d \log^3 k)$ and $O(\log k)$ recourse, making it suitable for large-scale and evolving datasets.

1 Introduction

Artificial intelligence systems play an increasingly important role in everyday life, influencing decisions that affect individuals, businesses, and society as a whole. As their impact grows, so does the need for transparency and human oversight. In response, there is a growing emphasis on making AI decisions understandable to people. This has led to the development of models that aim to present their decision-making processes in a clear and interpretable manner.

In this paper, we study algorithms for explainable clustering. The notion of explainable k-means and k-medians clustering was introduced by Dasgupta, Frost, Moshkovitz, and Rashtchian (2020) as a way to make clustering decisions more accessible to humans. Both k-means and k-medians are classical clustering objectives widely used in practice. Here, we focus on k-medians clustering under the ℓ_p norm. A k-medians clustering of a dataset $X \subset \mathbb{R}^d$ is defined by a collection of k centers $c^1, c^2, \ldots, c^k$. Each point $x \in X$ is assigned to the closest center in the ℓ_p norm, that is, the center minimizing $\|x - c^i\|_p$. Consequently, every clustering corresponds to a Voronoi partition under the ℓ_p norm. The cost of the clustering is defined as

$$\mathrm{cost}_p(X; c^1, \ldots, c^k) = \sum_{i=1}^{k} \sum_{x \in P_i} \|x - c^i\|_p,$$

where P_i denotes the set of points assigned to center c^i. We refer to this as *unconstrained* k-medians clustering.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

While this objective is simple to define and machines can easily compute the nearest centers, the resulting cluster assignments are often difficult for humans to interpret. To make clustering more comprehensible to humans, Dasgupta et al. (2020) proposed using threshold decision trees to represent clusterings. They referred to this approach as *explainable* k-means and k-medians. For k-medians, they considered the ℓ_1 norm. In a threshold decision tree, each internal node compares a single coordinate of the input to a threshold and directs the point to the left or right subtree accordingly. Each leaf of the tree represents a cluster. We denote the center assigned to x by the decision tree as $\mathcal{T}(x)$. The cost of the clustering is then defined similarly to the unconstrained case:

$$\mathrm{cost}_p(X, \mathcal{T}) = \sum_{x \in X} \|x - \mathcal{T}(x)\|_p.$$

Assigning a data point to a cluster using a threshold decision tree avoids complex distance computations and instead follows a simple, transparent process: each decision is based on a sequence of threshold comparisons. This makes it clear how a particular assignment was made and which features influenced it.

The central question is how much clustering quality is lost in exchange for interpretability. This trade-off is captured by the *the cost of explainability* or *competitive ratio*, defined as the worst-case ratio between the cost of the explainable clustering and that of the optimal unconstrained k-medians clustering:

$$\max_X \frac{\mathrm{cost}_p(X, \mathcal{T})}{\mathrm{OPT}_{k,p}(X)},$$

where $\mathrm{OPT}_{k,p}(X) = \min_{c_1,\ldots,c_k} \mathrm{cost}_p(X; c_1, \ldots, c_k)$ denotes the cost of the optimal (unconstrained) k-medians clustering of X.

Dasgupta et al. (2020) showed—perhaps surprisingly—that the competitive ratio for explainable k-medians under the ℓ_1 norm does not depend on the number of points in the dataset and can be bounded solely as a function of k; specifically, it is at most $O(k)$. They also established a lower bound of $\Omega(\log k)$. This result sparked significant interest and led to extensive study of explainable k-medians under the ℓ_1 norm. Makarychev and Shan (2021) and Esfandiari, Mirrokni, and Narayanan (2022) improved the upper bound to $\tilde{O}(\log k)$; see also Laber and Murtinho (2021) and Gamlath, Jia, Polak, and Svensson (2021) for related results. The approximation factor was later improved to $O(\log k)$ by Gupta, Pittu, Svensson, and Yuan (2023) and Makarychev and Shan (2023). Finally, Gupta et al. (2023) established a tight upper bound of $(1 + H_{k-1})$ for the ℓ_1 norm, where H_{k-1} denotes the $(k-1)$st harmonic number.

Beyond the ℓ_1 case, much less was known. For $p > 1$, the only prior result was due to Makarychev and Shan (2021), who provided a $\tilde{O}(\log^{3/2} k)$-competitive algorithm for the ℓ_2 norm. In this paper, we extend the study of explainable k-medians clustering to general ℓ_p norms with finite $p \geq 1$. Specifically, we design an algorithm that constructs a threshold decision tree with k leaves, such that the cost of the resulting clustering satisfies

$$\mathbf{E}[\mathrm{cost}_p(X, \mathcal{T})] \leq O(p \cdot \log^{1+1/p-1/p^2} k \cdot \log\log k) \cdot \mathrm{OPT}_k(X).$$

This improves upon the best known bound for $p = 2$, and for $p = 1$ it matches the optimal guarantee up to an $O(\log\log k)$ factor. Note that the exponent of the logarithm, $1 + 1/p - 1/p^2$, always lies in the interval $[1, 1.25]$.

We now discuss the second contribution of the paper. In recent years, researchers have turned their attention to dynamic clustering algorithms, which maintain a high-quality clustering as the dataset evolves and is continuously updated. Recent work in this area includes papers by Lattanzi and Vassilvitskii (2017); Chan, Guerqin, and Sozio (2018); Cohen-Addad, Hjuler, Parotsidis, Saulpic, and Schwiegelshohn (2019); Deng, Li, and Rabani (2022); Bhattacharya, Costa, Lattanzi, and Parotsidis (2023); Bhattacharya, Costa, Garg, Lattanzi, and Parotsidis (2024b); Bhattacharya, Costa, and Farokhnejad (2024a).

Dynamic algorithms are typically evaluated based on two key metrics: the update time for insertions and deletions, and the *recourse*—the number of changes made to the solution (in this case, centers inserted or deleted) in response to each update. Bhattacharya et al. (2024a) presented an approximation algorithm with $O(1)$-approximation ratio, $O(\log^2 \Delta)$ recourse and $\tilde{O}(k)$ update time (where Δ is an aspect ratio of the metric space).

In this paper, we initiate the study of dynamic algorithms for *explainable* k-medians clustering. Specifically, we ask whether our explainable algorithm can be combined with state-of-the-art dynamic k-medians clustering algorithms—and we answer this question affirmatively.

Most known algorithms for explainable k-medians clustering first compute a clustering using an existing off-the-shelf method, which we refer to as the *reference clustering*, and then use it to construct a decision tree. Importantly, this second step is *oblivious* to the dataset–that is, it relies only on the reference clustering and not on the actual data points. Our algorithm is no exception: it takes as input a set of reference centers and outputs a threshold decision tree whose cost is upper bounded by $\tilde{O}(p \cdot \log^{1+1/p-1/p^2} k)$ times the cost of the reference clustering. However, existing algorithms for explainable clustering are not designed to operate in a dynamic setting.

We present a dynamic implementation of our algorithm, in which the set of reference centers evolves over time through insertions and deletions. Our algorithm supports updates in $O(d \log^3 k)$ time and modifies only $O(\log k)$ nodes in the tree per update (i.e., it has $O(\log k)$ recourse), while maintaining the same $\tilde{O}(p \cdot \log^{1+1/p-1/p^2} k)$ competitive ratio.

Our algorithm can be integrated with the dynamic algorithms for unconstrained k-medians mentioned above. We begin by updating the set of centers using one of these low-recourse algorithms, and then apply our dynamic algorithm to update the decision tree for explainable clustering. Our algorithm can also be used to construct explainable clusterings for multiple values of k – for example, when selecting a suitable k within a given range using the elbow method. In such cases, we can run an algorithm (such as k-means++) that outputs centers incrementally, and feed these centers into our dynamic algorithm, which updates the decision tree on the fly.

1.1 Techniques

Our static algorithm for explainable k-medians under the ℓ_p norm builds on and refines a prior algorithm by Makarychev and Shan (2021) developed for the ℓ_2 norm. In this work, we generalize the approach to all ℓ_p norms with finite $p \geq 1$ and provide a tighter analysis. In particular, for the ℓ_2 norm, we improve the competitive ratio from the previous bound of $\tilde{O}(\log^{1.5} k)$ to $\tilde{O}(\log^{1.25} k)$.

As we noted earlier, our algorithm takes as input a set of reference centers produced by an off-the-shelf clustering algorithm and does not access the dataset points directly.

This algorithm relies on the PARTITION_LEAF procedure. Each call to PARTITION_LEAF takes a cell of the space containing some subset of centers C_u and constructs a partial threshold decision tree that partitions the cell into several subcells, each containing at most a $\tilde{\gamma}$ fraction of the input centers, where $\tilde{\gamma} < 1$. We apply PARTITION_LEAF recursively, starting with the cell containing all centers $c_1, \dots, c_k$, to construct the full decision tree.

PARTITION_LEAF first selects an *anchor* point within the cell. This anchor, denoted m^u, is the median or an approximate median of the centers in C_u and remains fixed throughout the execution of PARTITION_LEAF. The procedure partitions the space using random cuts drawn from a specially crafted distribution. Each time a cut is sampled and applied (some cuts may be discarded), the algorithm removes the centers that are separated from the anchor and places them into one of the output parts. Each cut is defined by a coordinate i and a threshold θ, and has the form $Left = \{x : x_i < \theta\}$ and $Right = \{x : x_i \geq \theta\}$. If a sampled cut does not separate any centers, it is discarded.

Random cuts in the algorithm are drawn as follows: PARTITION_LEAF selects a random coordinate $i \in \{1, \dots, d\}$, a random threshold $\theta' \in [0, R_t]$, and a random sign $\sigma \in \{\pm 1\}$ (where R_t is the radius of the cell; see Section 2 for details). It lets $\theta = m^u_i + \sigma\theta'$. The cumulative density function for θ' is given by x^p / R_t^p. The algorithm terminates when fewer than γn centers remain unseparated from the anchor.

We note that using a uniform distribution for θ (i.e., selecting a random coordinate i and then choosing a threshold θ uniformly at random from $[-R_t, R_t]$) would result in a poor competitive ratio, as illustrated in the following example. Consider a k-medians clustering with the ℓ_p norm, defined by $k+1$ centers located at the positions $e_1, \dots, e_k$, and 0, where e_i denotes the i-th standard basis vector. We focus on a single data point x with coordinates $(\varepsilon, \dots, \varepsilon)$. Suppose we pick cuts by selecting a random coordinate $i \in \{1, \dots, d\}$ and a threshold $\theta \in [0, 1]$ uniformly at random. In this case, a constant fraction of the centers will be separated from the anchor m^u in $\Theta(k)$ steps. The

probability that one of the cuts made during these steps separates x from its closest center (the center located at the origin) is $\Theta(\varepsilon k)$, assuming ε is sufficiently small. If x is separated from 0, it will be assigned to a different center, i.e., one of the vectors e_i. In that case, the ℓ_p distance from x to the new center is approximately 1. Therefore, the expected cost of the clustering produced by this variant of the algorithm for point x is $\Theta(\varepsilon k)$, while the optimal (unconstrained) cost is $\varepsilon k^{1/p}$. Hence, the competitive ratio of such an algorithm is at least $\Theta(k^{1-1/p})$.

In this paper, we prove – through a careful analysis of the algorithm – that the aforementioned choice of random distribution yields an $O(p \log^{1+1/p-1/p^2} \log \log k)$ upper bound on the algorithm's competitive ratio.

We then show how to implement our static clustering algorithm in the dynamic setting. Our approach builds on the idea of assigning each decision node a timestamp drawn from an exponential distribution – a technique previously introduced in Gupta et al. (2023); Makarychev and Shan (2023) solely for the purpose of analyzing an explainable clustering algorithm under the ℓ_1 norm. We extend this idea by integrating the exponential clock directly into the algorithm's design. Specifically, we assume that random cuts are selected with arrival rates governed by a Poisson process. Each cut is assigned a timestamp corresponding to its selection time.

The high-level idea behind the dynamic algorithm is as follows. When a new center is inserted, we identify the earliest cut – based on its timestamp – that separates the new center from the anchor. To efficiently find such a cut, we employ data structures that enable this operation in $O(d \log k)$ time. We prove that this earliest cut corresponds to the one that would have been used by the static algorithm to separate the center c from the anchor m^u. There are two possible cases: either the decision tree already contains a node corresponding to this cut, or it does not. In the latter case, the algorithm creates a new decision node to incorporate the cut.

Implementing this idea presents several challenges. The dynamic PARTITION_LEAF algorithm is not permitted to modify the anchor; consequently, it may need to rebuild the entire decision tree for a cell and its descendants once the number of updates in that cell exceeds a certain threshold. Moreover, the dynamic algorithm must terminate at a fixed time–one that cannot be adjusted as centers are added or removed. As a result, unlike the static version, it cannot stop based on the number of remaining centers falling below a given threshold. In this paper, we address these challenges and present a complete dynamic algorithm for the problem.

2 Algorithm

In this section, we present our algorithm for constructing an explainable clustering tree for the k-medians problem in ℓ_p space. The algorithm takes a set of k centers C as input and produces a binary threshold tree $\mathcal{T}$ with k leaves, each leaf containing a distinct center in C. The construction begins by initializing the root node r of the tree with all centers C, and recursively partitioning the centers using the procedure PARTITION_LEAF (as shown in Figure 1). We initiate the construction by calling PARTITION_LEAF(r).

While this algorithm is static, we show an efficient dynamic algorithm that achieves the same behavior as this algorithm in Section 4. To couple the dynamic algorithm with the static algorithm, we present our algorithm based on two oracles: STOPPING_ORACLE and GET_ANCHOR. The STOPPING_ORACLE takes a cut ω and the current subtree $\mathcal{T}_u$ rooted at u as input and outputs a Boolean value; if it is True, then it stops partitioning centers; otherwise, the algorithm continues to partition centers. This oracle guarantees that when partitioning stops, every leaf in $\mathcal{T}_u$ contains at most a $\tilde{\gamma}$ fraction of centers in C_u, where $\tilde{\gamma} < 1$. The oracle GET_ANCHOR takes a subset of centers C_u and returns an anchor point $m^u \in \mathbb{R}^d$ such that for each coordinate $i \in [d]$, at least $1/4$ of centers in C_u lie on either side of m_i^u, i.e. $|\{c \in C_u : c_i \geq m_i^u\}| \geq |C_u|/4$ and $|\{c \in C_u : c_i < m_i^u\}| \geq |C_u|/4$. In the static version, we can simply choose the anchor m^u as the coordinate-wise median of C_u, and the STOPPING_ORACLE returns True if and only if the main part contains fewer than $1/2$ of centers in C_u, i.e. $|C_{u_0}| < |C_u|/2$.

We now describe the procedure PARTITION_LEAF(u). The procedure PARTITION_LEAF(u) operates on a node u that contains a set of centers C_u. It first queries the oracle GET_ANCHOR to get an anchor point m^u. We always refer to the leaf that contains m^u as the *main part*, and denote it by u_0. Initially, we set $u_0 = u$.

Algorithm PARTITION_LEAF

Input: a node u with a set of centers $C_u \subseteq \mathbb{R}^d$
Output: a threshold tree $\mathcal{T}_u$

1. Set the anchor $m^u = \text{GET\_ANCHOR}(C_u)$.
2. Set the main part $u_0 = u$ and $C_{u_0} = C_u$ and step $t = 0$. Set the subtree $\mathcal{T}_u$ to have only the root u.
3. We iteratively sample cuts ω_t until $\text{STOPPING\_ORACLE}(\omega_t, \mathcal{T}_u)$ returns True:
   (a) Update $t = t + 1$ and the radius $R_t = \max_{c \in C_{u_0}} \|c - m^u\|_p$.
   (b) Sample a threshold cut $\omega_t = (i_t, \vartheta_t)$ as follows. Sample $i_t \in \{1, 2, \cdots, d\}$, $\sigma_t \in \{-1, 1\}$, and $(\theta_t)^p \in [0, (R_t)^p]$ uniformly at random. Then, set $\vartheta_t = m^u_{i_t} + \sigma_t \theta_t$.
   (c) **If** the cut ω_t separates any two centers in C_{u_0}, **then**
   - Add two new children u_L, u_R to the main part u_0 and split the centers into two parts $C_{u_L} = \{c \in C_{u_0} : c_{i_t} < \vartheta_t\}$ and $C_{u_R} = \{c \in C_{u_0} : c_{i_t} \geq \vartheta_t\}$.
   - Update the main part $u_0 = u_L$ and $C_{u_0} = C_{u_L}$ if $\sigma_t = 1$; otherwise $u_0 = u_R$ and $C_{u_0} = C_{u_R}$ (u_0 always contains m^u).
4. Call $\text{PARTITION\_LEAF}(v)$ for each leaf v containing more than one center in the subtree rooted at u.
5. Return the tree $\mathcal{T}_u$ rooted at node u.

Figure 1: Algorithm PARTITION_LEAF for explainable k-medians in ℓ_p

PARTITION_LEAF iteratively splits the subset C_u using randomized threshold cuts until the STOPPING_ORACLE returns True. In each iteration t, it computes the maximum ℓ_p distance from m^u to any center in the current main part C_{u_0}, denoted by $R_t = \max_{c \in C_{u_0}} \|c - m^u\|_p$. Then, it samples a random threshold cut ω_t as follows. A coordinate $i_t \in \{1, 2, \cdots, d\}$ and a sign $\sigma_t \in \{-1, 1\}$ are chosen uniformly at random. Next, it draws a random variable Z_t uniformly from the interval $[0, (R_t)^p]$ and set $\theta_t = (Z_t)^{1/p}$. The resulting threshold cut is $\omega_t = (i_t, \vartheta_t)$, where $\vartheta_t = m^u_i + \sigma_t \cdot \theta_t$. If this threshold cut separates at least two centers in C_{u_0}, the algorithm partitions the current main part u_0 into two disjoint cells. It adds two children u_L, u_R to the node u_0 and assigns centers $C_{u_L} = \{c \in C_{u_0} : c_{i_t} < \vartheta_t\}$ to node u_L and centers $C_{u_R} = \{c \in C_{u_0} : c_{i_t} \geq \vartheta_t\}$ to node u_R. The child node, either u_L or u_R, that contains anchor m^u becomes the updated main part u_0. This process continues until the STOPPING_ORACLE returns True. Finally, it recursively calls the $\text{PARTITION\_LEAF}(v)$ on each leaf v that contains more than one center in the subtree rooted at u.

3 Analysis of approximation factor

In this section, we provide the approximation guarantees for our algorithm.

Theorem 3.1. *Given a set of points X and a set of k centers C, for any $p \geq 1$, Algorithm finds a threshold tree $\mathcal{T}$ with k leaves that has k-medians cost*

$$\mathbf{E}[\text{cost}_p(X, \mathcal{T})] \leq O\left(p \cdot (\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \log\log k\right) \text{cost}_p(X, C).$$

We analyze the approximation guarantee by bounding the expected cost incurred by each point $x \in X$. Fix an arbitrary point $x \in X$ and let $c \in C$ be its closest center. We show that the expected cost of assigning x in the constructed threshold tree $\mathcal{T}$ is bounded by

$$\mathbf{E}[\text{cost}_p(x, \mathcal{T})] \leq O\left(p \cdot (\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \log\log k\right) \|x - c\|_p. \tag{1}$$

If x equals its closest center c, then x is always assigned to c by any tree $\mathcal{T}$, and thus incurs zero cost, $\text{cost}_p(x, \mathcal{T}) = 0$. In this case, the above bound holds trivially. Therefore, we may assume from now on that $x \neq c$.

Consider the path from the root to the leaf in the tree that contains this point x. We index the node on this path by $t = 1, 2, \cdots, T$, where u_1 is the root of the tree and u_T is the leaf that contains x. Let $\mathcal{T}_t$ be the partially built tree when the node u_t is generated in the algorithm. Given any tree $\mathcal{T}_t$, let $\mathcal{T}_t(x)$ be the closest center in the same leaf as x in tree $\mathcal{T}_t$. We define the following upper bound on the approximation factor.

Definition 3.2. *Let A_k be the smallest number such that the following inequality holds for every partially built tree $\mathcal{T}_t$,*

$$\mathbf{E}\left[\mathrm{cost}_p(x, \mathcal{T}) \mid \mathcal{T}_t\right] \le A_k \cdot \|x - \mathcal{T}_t(x)\|_p.$$

Since all centers are contained in the root u_1, we have $\mathcal{T}_1(x) = c$. Thus, we have A_k is an upper bound on the approximation factor. We then prove the following lemma, which provides a recurrence relation for bounding A_k.

Lemma 3.3. *For some absolute constant $\beta > 0$, we have for any step t^**

$$\mathbf{E}\left[\frac{\mathrm{cost}_p(x, \mathcal{T})}{\|x - \mathcal{T}_{t^*}(x)\|_p} \mid \mathcal{T}_{t^*}\right] \le \frac{A_k}{k} + \beta \cdot p(\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \cdot \log(A_k \log^2 k).$$

We first show how to use Lemma 3.3 to get the desired bound on A_k, which also provides the approximation factor for the algorithm.

Proof of Theorem 3.1. By Lemma 3.3 and the definition of A_k, we get the following recurrence relation on A_k, $A_k \le \frac{A_k}{k} + \beta \cdot p(\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \cdot \log(A_k \log^2 k)$. Then, we have that A_k is bounded by $A_k \le O\left(p(\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \log\log k\right)$. By the definition of A_k, we bound the expected cost of any point $x \in X$ given by tree $\mathcal{T}$ as shown in Equation (1). By taking the sum over all points in X, we get the approximation factor for the algorithm. □

3.1 Radius and diameter bounds

Before proving the main recurrence lemma, we establish several key results that describe how the radius and diameter of clusters evolve during the recursive partitioning process. These results serve as essential tools in our main proof. We defer the proofs to Appendix A.1.

We first show that the radius R_u decreases exponentially in one partition leaf call. Consider any partition leaf call on a node u. Let R_t be the radius of the main part before the iteration t of this partition leaf call. Then, we have $R_1 = R_u$. We use $\mathcal{T}_t$ to denote the partial tree given by the algorithm before the iteration t of this partition leaf call.

Lemma 3.4. *Consider any partition leaf call on node u. Let $L = \lceil 2^{p+3} d \ln k \rceil$. Then for every $t \ge 1$, we have $\Pr\{R_{t+L} > R_t/2 \mid \mathcal{T}_t\} \le \frac{1}{k^3}$.*

We define the diameter of a node u to be $D_u := \max_{c,c' \in C_u} \|c - c'\|_p$. We use the following relation between R_u and D_u for a node u at the beginning of a partition leaf call, which generalizes Lemma 6.1 in Makarychev and Shan (2022) to ℓ_p norm.

Lemma 3.5 (Lemma 6.1 in Makarychev and Shan (2022))**.** *For every node u on which the algorithm calls partition leaf, we have $R_u/2^{1/p} \le D_u \le 2R_u$.*

We define $\widetilde{D}_u$ for every node u as follows. If the algorithm calls partition leaf on node u, then $\widetilde{D}_u = D_u$. Now consider any node v in the partition leaf call of a node u, on which the algorithm does not call the partition leaf. Let $d(u, v)$ be the distance from v to u in the tree. We set $\widetilde{D}_v = \max\left\{D_v, \widetilde{D}_u \cdot \frac{R_v}{R_u}\right\}$. By the definition, $\widetilde{D}_u$ is an upper bound of the diameter D_u for every node u. We now show that $\widetilde{D}_u$ is non-increasing along any path from the root to a leaf in the tree. Since R_v is non-increasing in one partition leaf call, $\widetilde{D}_v$ is also non-increasing in one partition leaf call. Moreover, since $\widetilde{D}_v \ge D_v$ for every node v and $\widetilde{D}_u = D_u$ on node u where the algorithm calls partition leaf, we have $\widetilde{D}_v$ is also non-increasing across partition leaf calls.

Lemma 3.6. *For every node u, we have $R_u/2^{1/p} \le \widetilde{D}_u \le 2R_u$.*

We then show that $\widetilde{D}_u$ decreases exponentially along any path from the root to a leaf in the tree.

Lemma 3.7. *Let $L' = \lceil 2^{2p+5} d \ln k \rceil$. For every node u, let node v be any descendant of u at depth L' in the tree $\mathcal{T}$. Then, we have $\Pr\{\widetilde{D}_v \geq \widetilde{D}_u/2 \mid \mathcal{T}_u\} \leq \frac{1}{k^3}$.*

3.2 Recurrence lemma

In this section, we provide a proof overview of Lemma 3.3, which establishes the recurrence relation of A_k. The details of the proof are deferred to Appendix A.2.

We fix an arbitrary point $x \in X$. Without loss of generality, we consider the step $t^* = 1$ and then $\mathcal{T}_{t^*}(x) = c$ is the closest center to x in C. We then focus on the nodes in $\mathcal{T}$ that contain this point x, which form a path from the root to the leaf containing x. We index the node along this path by step $t = 1, 2, \cdots, T$, where u_1 is the root of the tree and u_T is the leaf that contains x. Let $\mathcal{T}_t$ be the partially built tree when the node u_t is generated in the algorithm.

We now bound the cost of this point x given by the tree $\mathcal{T}$. We begin by assuming that the radius R_t and the diameter substitute $\widetilde{D}_t$ decrease by a factor of 2 after every L' and L'' steps, respectively. By Lemma 3.4 and 3.7, and applying the union bound over all iterations, this good event holds with probability at least $1 - 1/k$. If this good event fails to hold, then we simply upper bound the expected cost of x by $A_k \|x - c\|_p$, which contributes the A_k/k factor.

Consider a node u_t such that both x and c are contained in u_t, and let ω_t be the cut sampled at this node. Let C_t be the set of centers contained in u_t and D_t be the diameter of u_t. If x and c are separated by this cut ω_t, then x is eventually assigned to a different center in C_t by $\mathcal{T}$. By the triangle inequality, we have the cost of x in $\mathcal{T}$ is at most $\|x - c\|_p + D_t$. Alternatively, we can use a more refined bound based on the notion of the fallback center, following the approach in Makarychev and Shan (2021, 2022). If x is separated from c by this cut ω_t, then we define the fallback center of x to be the closest center $c' \in C_{t+1}$ to x that is not separated from x by this cut ω_t. This fallback center depends on the tree $\mathcal{T}'$ and the cut ω_t. Let $M_t(\omega_t)$ denote the distance from x to the fallback center. Then, by the definition of A_k, the expected cost of x can also be upper bounded by $A_k M_t(\omega_t)$.

We now partition the steps $\{1, 2, \cdots, T\}$ into three disjoint cases based on the radius R_t and the fallback distance $M_t(\omega)$ as follows. We introduce the following definitions.

Definition 3.8. *For a fixed parameter $\alpha > 0$, we say that step t is a light step if the radius satisfies*

$$R_t \leq 6 \log^{\alpha} k \cdot \max\left\{\|x - m^t\|_p, \|c - m^t\|_p\right\}.$$

Otherwise, step t is called a heavy step.

If x and c are separated by a cut ω_t, then we refer to this cut as a light cut if step t is a light step, and a heavy cut if step t is a heavy step.

Definition 3.9. *For each step t, we say a cut ω_t separating x and c a safe cut if $A_k M_t(\omega_t) \leq \frac{R_t}{6^p \log^2 k}$. Otherwise, this cut ω_t is called an unsafe cut.*

Therefore, if x and c are separated by the tree $\mathcal{T}$, then exactly one of the following three events must occur: (1) they are separated by a safe cut; (2) they are separated by a light cut; (3) they are separated by a heavy and unsafe cut. We then show how to bound the contribution of each case to the expected cost separately.

Safe cut: Suppose x and c are contained in node u_t. The probability that x and c are separated by the cut ω_t is at most

$$\Pr\{x \text{ \& } c \text{ separated by } \omega_t \mid \mathcal{T}_t\} \leq \frac{1}{2d} \cdot \frac{p\|x - c\|_p (\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1})}{R_t^p}.$$

In this case, we use $A_k M_t$ as the upper bound of the expected cost since it is much smaller than the radius R_t. We show that $3R_t \geq \|x - m^t\|_p + \|c - m^t\|_p$. Thus, the expected cost of a safe cut at step t is at most $\frac{p}{2d} \cdot \frac{A_k M_t}{R_t} \cdot 3^{p-1} \cdot \|x - c\|_p$. In each partition leaf call, we know that M_t is non-decreasing as t increases and R_t decreases by a factor of 2 after every L' steps. Hence, $A_k M_t / R_t$ forms an increasing geometric series in every L' steps. Since $A_k M_t / R_t \leq 1/(6^p \log^2 k)$ for safe cuts, the expected cost due to safe cuts in one partition leaf call is at most

$$L' \cdot \frac{p}{2d} \cdot \frac{2}{6^p \log^2 k} \cdot 3^{p-1} \cdot \|x - c\|_p \leq O\left(\frac{1}{\log k}\right) \|x - c\|_p.$$

Combining over all $O(\log k)$ partition leaf calls, this case is bounded by $O(1) \cdot \|x - c\|_p$.

Light cut: Consider the node u_t contains x and c. The probability that x or c is separated from the anchor m^t by ω_t is at least

$$\Pr\{x \text{ or } c \text{ separated from } m^t \text{ by } \omega_t \mid \mathcal{T}_t\} \geq \frac{1}{2d} \cdot \frac{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}}{R_t^p}.$$

Thus, in each partition leaf call, the probability that x and c are separated by a light cut at the end of the partition leaf call is most

$$\frac{p\|x - c\|_p(\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1})}{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}}.$$

We upper bound the expected penalty by $D_t \leq 2R_t \leq 12 \log^\alpha k \cdot \max\{\|x - m^t\|_p, \|c - m^t\|_p\}$ by the definition of a light cut. Since the number of partition leaf calls is at most $O(\log k)$, the expected cost due to a light cut is at most

$$O(\log k) \cdot D_t \cdot \frac{p\|x - c\|_p(\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1})}{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}} \leq O(p \log^{1+\alpha} k)\|x - c\|_p.$$

Heavy and unsafe cut: Consider a heavy step t when x and c are contained in node u_t. For each coordinate i, we define $U_i(t) = \{\vartheta : (i, \vartheta) \text{ is unsafe}\}$ to be all thresholds ϑ such that the cut $\omega_t = (i, \vartheta)$ is unsafe at step t. Let $\delta_i(t)$ be the Lebesgue measure of the unsafe threshold $U_i(t)$. Then, the probability that x and c are separated by an unsafe cut at the heavy step t is at most

$$\frac{p}{2d} \cdot \sum_{i=1}^{d} \frac{\max\{|x_i - m_i^t|, |c_i - m_i^t|\}^{p-1}}{R_t^p} \cdot \delta_i(t).$$

Note that all steps in P_s in one partition leaf call s uses the same anchor point m^s. Let $P_s' \subseteq P_s$ be all heavy steps in the partition leaf call. We define a vector $\Delta(s) \in \mathbb{R}^d$ whose i-th coordinate is $\Delta_i(s) = \sum_{t \in P_s'} \delta_i(t)$. By summing the above separation probability over all steps in P_s' and applying Hölder's inequality, the probability that x and c are separated by a heavy and unsafe cut in partition leaf call s is at most

$$\frac{p}{2d} \cdot \|\Delta(s)\|_p \cdot \frac{\|x - m^s\|_p^{p-1} + \|c - m^s\|_p^{p-1}}{R_t^p}.$$

In this case, we upper bound the penalty of separation by $D_t \leq 2R_t$. Since $R_t \geq 6 \log^\alpha k \cdot \max\{\|x - m^t\|_p, \|c - m^t\|_p\}$ for heavy steps, we have the expected penalty due to heavy and unsafe cuts is at most

$$\frac{p}{d} \cdot \frac{2}{(6 \log^\alpha k)^{p-1}} \cdot \sum_{s=1}^{S} \|\Delta(s)\|_p.$$

We then bound $\sum_{s=1}^{S} \|\Delta(s)\|_p$. Since the number of partition leaf calls is $S = O(\log k)$, we show that

$$\sum_{s=1}^{S} \|\Delta(s)\|_p \leq \log^{1-\frac{1}{p}} k \left\| \sum_{s=1}^{S} \Delta(s) \right\|_p.$$

Consider any fixed cut $\omega = (i, \vartheta)$ that separates x and c. This cut is unsafe at step t if and only if $M_t(\omega) \geq R_t/(6^p \log^2 k \cdot A_k)$. Moreover, it always holds $M_t(\omega) \leq D_t$. By Lemma 3.6, we have $R_t \geq \widetilde{D}_t$ and $\widetilde{D}_t \geq D_t$. Since by Lemma 3.7, $\widetilde{D}_t$ decreases by a factor of 2 after every $L'' = \lceil 2^{2p+5} d \ln k \rceil$ steps, this cut ω is unsafe in at most $L'' \cdot \log(2 \cdot 6^p \log^2 k \cdot A_k)$ steps. Thus, we have

$$\left\| \sum_{s=1}^{S} \Delta(s) \right\|_p \leq O(4^p \cdot d \log k \cdot p \log(\log^2 k \cdot A_k))\|x - c\|_p.$$

Therefore, the expected cost due to heavy and unsafe cuts is at most

$$O\left((\log k)^{2-\frac{1}{p}-\alpha(p-1)} \log(\log^2 k \cdot A_k)\right)\|x - c\|_p.$$

Finally, combining all three cases and taking $\alpha = 1/p - 1/p^2$, we get the conclusion.

4 Dynamic algorithm

In this section, we present a dynamic algorithm for the setting where the input set of points X and centers C change over time. In the appendix, we show that after each update, our algorithm maintains a threshold tree with low k-medians cost and analyze the update time.

We focus on updates to the center set as the construction of the threshold tree only depends on C. The input changes through a sequence of updates, where each update inserts or deletes a center. Let $C_1, C_2, \ldots, C_t, \ldots$ denote the corresponding sequence of center sets, where each update modifies the previous set: if the t^{th} update inserts a center c, then $C_t = C_{t-1} \cup \{c\}$; if it deletes c, $C_t = C_{t-1} \setminus \{c\}$. We show the following theorem, with the proof in Appendix B.

Theorem 4.1. *Given a stream of updates that generates a sequence of center sets $C_1, C_2, \ldots$, there is a dynamic algorithm that for each center set C_t, outputs a threshold tree $\mathcal{T}_t$ with approximation $\tilde{O}(p \cdot \log^{1+1/p-1/p^2} k)$. The amortized update time of the algorithm is $O(d \log^3 k)$ and the amortized recourse is $O(\log k)$.*

To implement our dynamic algorithm, we reinterpret the PARTITION_LEAF procedure (Figure 1) in an equivalent but more convenient way using the exponential clock. This version generates all random cuts in advance. Without loss of generality, we assume that all centers lie within $[-1, 1]^d$; otherwise, we rescale the instance accordingly. The procedure generates an infinite sequence of candidate cuts $\omega_1, \omega_2, \ldots$, where each cut $\omega_t = (i_t, \vartheta_t)$ is constructed as follows: a coordinate i_t, a sign $\sigma_t \in \{-1, 1\}$, and a parameter $Z_t \in [0, 2^p]$ are sampled uniformly at random. The threshold is then set to $\vartheta_t = m_{i_t} + \sigma_t \cdot (Z_t)^{1/p}$, where m denotes the anchor point. Additionally, each cut ω_t is assigned an *arrival time* ρ_t, such that $\rho_1 \leq \rho_2 \leq \ldots$ follows the arrival times of a Poisson Process with rate $\lambda = 1$.

The algorithm attempts the next cut (ω_t, ρ_t) in the sequence until the STOPPING_ORACLE returns True. If ω_t separates at least two centers from the main part, the cut is made; otherwise, it is ignored. Since the arrival times ρ_t are independent of cut choices ω_t, this version yields the same distribution of threshold trees as the original PARTITION_LEAF procedure. These arrival times ρ_t are crucial for the design of our dynamic algorithm. In the following discussion, we assume there is a data structure that stores this sequence of cuts with their arrival times. It also provides a function GET_EARLIEST_CUT that takes a center c and returns the earliest cut ω from the sequence that separates c and the anchor m.

We provide a dynamic implementation of the PARTITION_LEAF procedure, which we apply recursively to obtain a fully dynamic version of the entire clustering algorithm. The dynamic variant of PARTITION_LEAF supports three operations: (1) REBUILD, (2) INSERT CENTER, and (3) DELETE CENTER. We now briefly describe the insert operation and defer the discussion of the remaining two operations to the appendix.

Let u be the node on which this operation is applied. Suppose a center c is inserted into or deleted from the set of centers assigned to u. For each partition leaf call, we maintain a counter that tracks the number of such updates since the last rebuild. Let k' be the number of centers in node u at the time of the last rebuild. When the update count exceeds $k'/4$, we rebuild the partial tree rooted at node u. We now proceed to handle the update.

Insert: Suppose a new center c is inserted. The algorithm calls GET_EARLIEST_CUT to find the earliest cut ω in the pre-generated sequence with its arrival time ρ that separates c from the anchor m^u. Let $(\omega'_1, \rho'_1), \cdots, (\omega'_r, \rho'_r)$ be the cuts currently used in this partition leaf call. Let ρ^u be the stopping time assigned to this partition leaf call during its most recent rebuild. We consider three cases as follows: (1) $\rho = \rho'_j$ for some $j \in [r]$; (2) $\rho > \rho^u$; (3) $\rho \leq \rho^u$ and $\rho \neq \rho'_j$ for any $j \in [r]$.

Case (1): Assign this new center c to the node v generated by cut ω'_j and recursively maintain the partition leaf call rooted at v.

Case (2): This new center c remains in the main part u_0 until this partition leaf call ends. We then recursively maintain the partition leaf call on the main part u_0.

Case (3): It finds the smallest index $j \in [r]$ such that $\rho < \rho'_j$ or sets $j = r + 1$ if no such index exists. Then we insert this new cut ω at position j and add a new leaf node containing c to the tree.

References

Sayan Bhattacharya, Martin Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in o(k) update time. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Sayan Bhattacharya, Martín Costa, and Ermiya Farokhnejad. Fully dynamic k-median with near-optimal update time and recourse. *arXiv preprint arXiv:2411.03121*, 2024a.

Sayan Bhattacharya, Martín Costa, Naveen Garg, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering with fast update time and small recourse. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 216–227. IEEE, 2024b.

TH Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *Proceedings of the 2018 World Wide Web Conference*, pages 579–587, 2018.

Vincent Cohen-Addad, Niklas Oskar D Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. *Advances in Neural Information Processing Systems*, 32, 2019.

Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. Explainable k-means and k-medians clustering. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7055–7065, 2020.

Shichuan Deng, Jian Li, and Yuval Rabani. Approximation algorithms for clustering with dynamic points. *Journal of Computer and System Sciences*, 130:43–70, 2022.

Hossein Esfandiari, Vahab Mirrokni, and Shyam Narayanan. Almost tight approximation algorithms for explainable clustering. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2641–2663. SIAM, 2022.

Buddhima Gamlath, Xinrui Jia, Adam Polak, and Ola Svensson. Nearly-tight and oblivious algorithms for explainable clustering. *Advances in Neural Information Processing Systems*, 34:28929–28939, 2021.

Anupam Gupta, Madhusudhan Reddy Pittu, Ola Svensson, and Rachel Yuan. The price of explainability for clustering. *arXiv preprint arXiv:2304.09743*, 2023.

John Frank Charles Kingman. *Poisson processes*, volume 3. Clarendon Press, 1992.

Eduardo S Laber and Lucas Murtinho. On the price of explainability for some clustering problems. In *International Conference on Machine Learning*, pages 5915–5925. PMLR, 2021.

Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *International Conference on Machine Learning*, pages 1975–1984. PMLR, 2017.

Konstantin Makarychev and Liren Shan. Near-optimal algorithms for explainable k-medians and k-means. In *International Conference on Machine Learning*, pages 7358–7367. PMLR, 2021.

Konstantin Makarychev and Liren Shan. Explainable k-means: don't be greedy, plant bigger trees! In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1629–1642, 2022.

Konstantin Makarychev and Liren Shan. Random cuts are optimal for explainable k-medians. *Advances in Neural Information Processing Systems*, 36:66890–66901, 2023.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: We provide complete proofs of all the claims we make in the abstract, introduction, and other sections of the paper.

   Guidelines:

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

2. **Limitations**

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

   Answer: [NA]

Justification: The performance of the algorithms is analyzed without making any assumptions about the data.

Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: We provide complete proofs of all our results.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [NA]

Justification: The paper does not include experiments.

Guidelines:

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

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

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

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

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics and confirm that this research was conducted in accordance with it.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [NA]

    Justification: This work focuses on the development and analysis of approximation algorithms for explainable k-medians clustering under general ℓ_p norms. It does not develop any technology that has harmful or malicious applications. As such, we believe there are no direct negative societal impacts resulting from the work in its current form.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [NA]

Justification: The paper poses no such risks.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [NA]

Justification: The paper does not use existing assets.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [NA]

Justification: The paper does not release any assets.

Guidelines:

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

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

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

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

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

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

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

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [NA]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Proofs in Section 3

A.1 Proofs in Section 3.1

Lemma 3.4. *Consider any partition leaf call on node u. Let $L = \lceil 2^{p+3} d \ln k \rceil$. Then for every $t \geq 1$, we have $\Pr\{R_{t+L} > R_t/2 \mid \mathcal{T}_t\} \leq \frac{1}{k^3}$.*

Proof of Lemma 3.4. Let C_t be the centers contained in the main part before the iteration t of the partition leaf call. Then, we have $C_1 = C_u$ be the set of centers contained in node u. Let m^u be the median of centers in C_u. Consider any center $c \in C_t$ with $\|c - m^u\|_p > R_t/2$. Suppose the algorithm chooses the coordinate i at iteration t. Then, this center c is separating from m^u at iteration t if and only if $\sigma_t = \operatorname{sgn}(c_i - m_i^u)$ and $\theta_t \in (0, |c_i - m_i^u|]$. Thus, we have

$$\Pr\{c, m^u \text{ are separated at } t \mid \mathcal{T}_t, i_t = i\} = \frac{1}{2} \frac{|c_i - m_i^u|^p}{R_t^p}.$$

Combining all coordinates, the probability that c is separated from m^u at iteration t is at least

$$\begin{aligned}\Pr\{c, m^u \text{ are separated at } t \mid \mathcal{T}_t\} &= \sum_{i=1}^{d} \frac{1}{d} \cdot \Pr\{c, m^u \text{ are separated at } t \mid \mathcal{T}_t, i_t = i\} \\ &= \sum_{i=1}^{d} \frac{1}{2d} \cdot \frac{|c_i - m_i^u|^p}{R_t^p} = \frac{1}{2d} \frac{\|c - m^u\|_p^p}{R_t^p} \geq \frac{1}{2d \cdot 2^p} = \frac{1}{2^{p+1} d}.\end{aligned}$$

Since in one partition leaf call, the radius R_t is non-increasing as t increases, for any iteration $t' \geq t$, we have $\|c - m^u\|_p > R_{t'}/2$. Hence, conditioned on $\mathcal{T}_t$, if c is not separated from m^u before iteration $t' \geq t$, then c is separated from m^u at iteration t' with probability at least $1/2^{p+1} d$. Therefore, the probability that c is not separated from m^u after $L = \lceil 2^{p+3} d \ln k \rceil$ iterations is at most

$$\left(1 - \frac{1}{2^{p+1} d}\right)^L \leq e^{-\frac{L}{2^{p+1} d}} = \frac{1}{k^4}.$$

Since there are at most k centers with distance to m^u greater than $R_t/2$, by the union bound over all such centers, we have

$$\Pr\{R_{t+L} > R_t/2 \mid \mathcal{T}_t\} \leq \frac{1}{k^3}.$$

□

We show the following relation between the radius R_u and the diameter D_u for each node u on which the algorithm calls the partition leaf. We assume that the oracle GET_ANCHOR sets the anchor m^u as the coordinate-wise median of all centers in C_u in the lemma below. Suppose that GET_ANCHOR returns an approximate median m^u such that for each coordinate i, at least $1/4$ of the centers in C_u lie on either side of m_i^u. Then, we obtain a similar relation $R_u/4^{1/p} \leq D_u \leq 2R_u$.

Lemma 3.5 (Lemma 6.1 in Makarychev and Shan (2022))**.** *For every node u on which the algorithm calls partition leaf, we have $R_u/2^{1/p} \leq D_u \leq 2R_u$.*

Proof of Lemma 3.5. It is easy to get the second bound from the triangle inequality of the ℓ_p norm. Let m^u be the median of centers in C_u. We have for any two centers $c, c' \in C_1^u$,

$$\|c - c'\|_p \leq \|c - m^u\|_p + \|m^u - c'\|_p \leq 2R_u.$$

We then show the first bound. For any function $f : C_u \to \mathbb{R}$, let $\operatorname{avg}_{c \in C_u} f(c) = \frac{1}{|C_u|} \sum_{c \in C_u} f(c)$ be the average of $f(c)$ over all centers in C_u. Let $c' = \arg\max_{c \in C_u} \|c - m^u\|_p$ be the center that is farthest from the median m^u in ℓ_p norm. For any pair of centers $c, \hat{c} \in C_u$, the distance between c and $\hat{c}$ is at most the diameter of u, $\|c - \hat{c}\|_p \leq D_u$. Thus, we have

$$D_u^p \geq \operatorname{avg}_{c \in C_u} \|c' - c\|_p^p = \operatorname{avg}_{c \in C_u} \sum_{i=1}^{d} |c_i' - c_i|^p = \sum_{i=1}^{d} \operatorname{avg}_{c \in C_u} |c_i' - c_i|^p.$$

Since m^u is the median of the centers in C_u, at least half of the centers $c \in C_u$ lie on the opposite of the hyperplane $\{x : x_i = m_i^u\}$ from the center c'. Thus, for these centers $c \in C_u$, we have $|c'_i - c_i| \geq |c'_i - m_i^u|$. Thus, we have

$$D_u^p \geq \sum_{i=1}^{d} \mathrm{avg}_{c \in C_u} |c'_i - c_i|^p \geq \sum_{i=1}^{d} \frac{1}{2} \cdot |c'_i - m_i^u|^p = \frac{1}{2} \cdot \|c' - m^u\|_p^p = \frac{1}{2} R_u^p,$$

which implies $R_u/2^{1/p} \leq D_u$. □

Lemma 3.6. *For every node u, we have $R_u/2^{1/p} \leq \widetilde{D}_u \leq 2R_u$.*

Proof of Lemma 3.6. For any node u on which the algorithm calls the partition leaf, we have $\widetilde{D}_u = D_u$. By Lemma 3.5, we have $R_u/2^{1/p} \leq \widetilde{D}_u \leq 2R_u$.

We then consider any node v which is not a partition leaf call node. Let u be the node of partition leaf call that generates the node v. Since $\widetilde{D}_u \leq 2R_u$, we have $\widetilde{D}_u R_v / R_u \leq 2R_v$. Note that $D_v \leq 2R_v$. Thus, we have $\widetilde{D}_v \leq 2R_v$. Since $\widetilde{D}_u \geq R_u/2^{1/p}$, we have $\widetilde{D}_v \geq \widetilde{D}_u \cdot R_v/R_u \geq R_v/2^{1/p}$. □

We then show that $\widetilde{D}_u$ decreases exponentially along any path from the root to a leaf in the tree. First, we show that any pair of centers that are far apart in the node are separated with high probability. Let $\mathcal{T}_u$ be the partial tree when node u is generated in the algorithm.

Lemma A.1. *For every two centers c' and c'' in C_u at distance at least $\widetilde{D}_u/2$,*

$$\Pr\{c', c'' \textit{ are separated at } u \mid \mathcal{T}_u\} \geq \frac{1}{d \cdot 2^{2p+1}}.$$

Proof. Suppose the algorithm picks coordinate i at node u. For every two centers $c', c'' \in C_u$, we consider the following two cases: (1) c' and c'' are on the same side of the median m^u in coordinate i; (2) c' and c'' are on the opposite side of the median m^u on coordinate i.

For the first case, without loss of generality, we assume that $c''_i \geq c'_i \geq m_i^u$. Then, two centers c' and c'' are separated by the cut at node u if and only if the algorithm picks $\sigma_u = 1$ and $\theta_u \in (c'_i - m_i^u, c''_i - m_i^u]$. Let $\mathcal{T}_u$ be the partial tree when node u is generated. Then, we have

$$\begin{aligned}\Pr\{c', c'' \text{ are separated at } u \mid i_u = i, \mathcal{T}_u\} =& \frac{1}{2} \cdot \frac{(c''_i - m_i^u)^p - (c'_i - m_i^u)^p}{R_u^p} \\ \geq& \frac{(c''_i - c'_i)^p}{2R_u^p},\end{aligned}$$

where the inequality is because x^p is convex and increasing on $[0, \infty)$.

For the second case, c'_i and c''_i are on the opposite side of m_i^u. Assume that $c'_i \geq m_i^u \geq c''_i$. Thus, centers c' and c'' are separated by the cut at node u if and only if $\sigma = +1, \theta \in (0, c'_i - m_i^u]$ or $\sigma = -1, \theta \in (0, c''_i - m_i^u]$. Thus, we have

$$\begin{aligned}\Pr\{c', c'' \text{ are separated at } u \mid i_u = i, \mathcal{T}_u\} =& \frac{1}{2} \cdot \frac{|c''_i - m_i^u|^p + |c'_i - m_i^u|^p}{R_u^p} \\ \geq& \frac{|c''_i - c'_i|^p / 2^{p-1}}{2R_u^p},\end{aligned}$$

where the inequality is from $(a^p + b^p)/2 \geq ((a+b)/2)^p$ for $a, b \geq 0$ since x^p is convex on $[0, \infty)$.

Combining all coordinates, we have the probability that c' and c'' are separated at node u is at least

$$\Pr\{c', c'' \text{ are separated at } u \mid \mathcal{T}_u\} \geq \sum_{i=1}^{d} \frac{1}{d} \cdot \frac{|c''_i - c'_i|^p}{(2R_u)^p} \geq \frac{\|c'' - c'\|_p^p}{d(2R_u)^p}.$$

Since $\widetilde{D}_u \geq R_u/2^{1/p}$, we have for every two centers $c', c'' \in C_u$ with $\|c'' - c'\|_p \geq \widetilde{D}_u/2$,

$$\Pr\{c', c'' \text{ are separated at } u \mid \mathcal{T}_u\} \geq \frac{R_u^p}{2 \cdot 2^p} \cdot \frac{1}{d(2R_u)^p} \geq \frac{1}{2^{2p+1} d}.$$

□

Lemma 3.7. *Let $L' = \lceil 2^{2p+5} d \ln k \rceil$. For every node u, let node v be any descendant of u at depth L' in the tree $\mathcal{T}$. Then, we have* $\Pr\{\widetilde{D}_v \geq \widetilde{D}_u/2 \mid \mathcal{T}_u\} \leq \frac{1}{k^3}$.

Proof of Lemma 3.7. Let u' be the node at which the algorithm calls the partition leaf that generates the node v. Then, we consider two cases: (1) $d(u', v) \geq 3 \cdot L$; (2) $d(u', v) < 3 \cdot L$, where $L = \lceil 2^{p+3} d \ln k \rceil$ used in Lemma 3.4.

In the first case, by Lemma 3.4, we have with probability at least $1 - 1/k^3$,

$$\widetilde{D}_v \leq 2R_v \leq 2 \cdot \frac{R_{u'}}{2^3} \leq 2 \cdot \frac{2^{1/p} D_{u'}}{2^3} \leq \frac{\widetilde{D}'_u}{2}.$$

In the second case, we have $d(u, u') \geq d(u, v) - d(v, u') \geq 2^{2p+4} d \ln k$. Thus, by Lemma A.1, we have every two centers in node u at distance of at least $\widetilde{D}_u/2$ are not separated at node u' with probability at most

$$\left(1 - \frac{1}{d \cdot 2^{2p+1}}\right)^{2^{2p+4} d \ln k} \leq \frac{1}{k^5}.$$

By the union bound over all pairs of centers and all nodes, we have with probability at least $1 - 1/k^3$, all such pairs are separated at node u'. Thus, we have with probability at least $1 - 1/k^3$

$$\widetilde{D}_v \leq \widetilde{D}_{u'} = D_{u'} \leq \frac{\widetilde{D}_u}{2}.$$

□

A.2 Proof of Lemma 3.3

Lemma 3.3. *For some absolute constant $\beta > 0$, we have for any step t^**

$$\mathbf{E}\left[\frac{\mathrm{cost}_p(x, \mathcal{T})}{\|x - \mathcal{T}_{t^*}(x)\|_p} \mid \mathcal{T}_{t^*}\right] \leq \frac{A_k}{k} + \beta \cdot p (\log k)^{1 + \frac{1}{p} - \frac{1}{p^2}} \cdot \log(A_k \log^2 k).$$

Proof of Lemma 3.3. Fix an arbitrary point $x \in X$. Without loss of generality, suppose the step $t^* = 1$, in which case $\mathcal{T}_{t^*}(x) = c$ is the closest center to x in C. Otherwise, if $t^* > 1$, then conditioned on $\mathcal{T}_{t^*}$, we consider the subinstance consisting of centers that lie in the same leaf of $\mathcal{T}_{t^*}$ as x.

We consider all steps in which the algorithm samples a cut to split the node containing x in the partial tree. Some of these sampled cuts may be rejected by the algorithm if they fail to separate any centers within the node. With a slight abuse of notation, we index these steps by $t = 1, 2, \cdots, T$. Let $\mathcal{T}_t$ be the partially built tree before the cut at step t and let u_t be the node containing x in $\mathcal{T}_t$. The sequence of nodes $u_1, \cdots, u_T$ thus form a path from the root to the leaf in the final tree $\mathcal{T}$ that contains x.[1] We divide the iterations into consecutive parts $P_1, \cdots, P_S$, each corresponding to one of the S partition leaf calls. Within each part P_s, all steps $t \in P_s$ for $t \in P_s$ occur in the same partition leaf call and share the same anchor point m^s. Since each partition leaf call reduces the number of centers in every leaf node by at least a factor of two compared to its root, the total number of partition leaf call is at most $O(\log k)$.[2]

Suppose that at step t, the point x and the center c are contained in the same node u_t before the cut is applied. Let $\omega_t = (i, \vartheta)$ be the cut selected by the algorithm at this step. We define the penalty $\phi_t(\omega_t)$, or equivalently $\phi_t(i, \vartheta)$, for the cut (i, ϑ) at step t as follows. If x and c are not separated by cut (i, ϑ), then we set $\phi_t(i, \vartheta) = 0$. Otherwise, the penalty is given by

$$\phi_t(i, \vartheta) = \mathbf{E}[\mathrm{cost}_p(x, \mathcal{T}) \mid \mathcal{T}_t, \omega_t = (i, \vartheta)] - \|x - c\|_p.$$

[1] Some of the nodes in the path may appear multiple times in the sequence since certain cuts may be rejected by the algorithm, leaving the node containing x unchanged.

[2] The STOPPING_ORACLE ensures that when partitioning stops, each leaf contains at most a γ fraction of the centers in its root for some constant $\gamma < 1$. Therefore, for any valid STOPPING_ORACLE, the number of partition leaf call is bounded by $O(\log k)$.

We now show two upper bounds on this penalty term. Conditioned on the partial tree $\mathcal{T}_t$, we know that in the final tree $\mathcal{T}$, the point x must eventually be assigned to a center in C_{u_t}, the set of centers contained in node u_t. By the triangle inequality, the final cost for x is at most $\|x - c\|_p + D_t$, where D_t is the diameter of node u_t. Thus, the penalty is at most D_t. If x and c are separated by cut (i, ϑ) at iteration t, then we call the center c' closest to x in u_{t+1} as the fallback center. Define $M_t(i, \vartheta) = \|x - c'\|_p$ as the distance from x to its fallback center. By the definition of A_k, we have the penalty in this case is at most $A_k \cdot M_t(i, \vartheta)$. Combining both bounds, we obtain $\phi_t(i, \vartheta) \le \min\{D_t, A_k M_t(i, \vartheta)\}$.

Let $L = \lceil 2^{p+3} d \ln k \rceil$ and $L' = \lceil 2^{2p+5} d \ln k \rceil$. We define the stopping time τ to be the first step t such that one of the following events happens: (1) $R_t < \|x - c\|_p$; (2) x and c are separated by the cut chosen at step t; (3) $\widetilde{D}_t \ge \widetilde{D}_{t-L'}/2$ for $t > L'$; (4) $R_t \ge R_{t-L}/2$ for $t > L$. We define four disjoint events as follows,

- $\mathcal{E}_1 = \{R_\tau < \|x - c\|_p\}$,
- $\mathcal{E}_2 = \{x \text{ and } c \text{ are separated by the cut chosen at step } \tau\} \setminus \mathcal{E}_1$,
- $\mathcal{E}_3 = \{\widetilde{D}_\tau \ge \widetilde{D}_{\tau - L'}/2, t > L'\} \setminus (\mathcal{E}_1 \cup \mathcal{E}_2)$,
- $\mathcal{E}_4 = \{R_\tau \ge R_{\tau - L}/2, t > L\} \setminus (\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3)$.

We call $\mathcal{E}_1, \mathcal{E}_2$ good events and $\mathcal{E}_3, \mathcal{E}_4$ bad events. By Lemma 3.4 and 3.7, we have that the events $\mathcal{E}_3$ and $\mathcal{E}_4$ happen with probability at most $\Pr\{\mathcal{E}_3\} \le 1/k$ and $\Pr\{\mathcal{E}_4\} \le 1/k$. If either $\mathcal{E}_3$ or $\mathcal{E}_4$ occurs, we upper bound the expected cost of x in $\mathcal{T}$ by $A_k \cdot \|x - c\|_p$ since x and c remain unseaprated at step τ. Therefore, the expected cost of point x given by the tree $\mathcal{T}$ is at most

$$\begin{aligned}\mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})] &= \mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_1 \cup \mathcal{E}_2\}] + \mathbf{E}[\mathrm{cost}_p(x, \mathcal{T}) \mid \mathcal{E}_3 \cup \mathcal{E}_4] \Pr\{\mathcal{E}_3 \cup \mathcal{E}_4\} \\ &\le \mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_1 \cup \mathcal{E}_2\}] + A_k \|x - c\|_p \cdot \frac{1}{k}.\end{aligned}$$

We then bound the expected cost of point x under the good events, $\mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_1 \cup \mathcal{E}_2\}]$.

When the event $\mathcal{E}_1$ happens, we have x and c are not separated before step τ. Since the diameters of nodes containing x are non-increasing, the final cost for x in this case can be bounded by

$$\|x - c\|_p + D_\tau \le \|x - c\|_p + 2R_\tau < 3\|x - c\|_p.$$

Thus, we have

$$\mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_1\}] \le 3\|x - c\|_p \cdot \Pr\{\mathcal{E}_1\} \le 3\|x - c\|_p.$$

We now turn to analyzing the event $\mathcal{E}_2$. We further partition this event based on the step at which x and c are first separated. For each step $1 \le t \le T$, we define

$$\mathcal{E}_{2,t} = \{x \text{ and } c \text{ are separated by the cut chosen at step } \tau \;\&\; \tau = t\} \setminus \mathcal{E}_1.$$

These events $\mathcal{E}_{2,t}$ are disjoint and we have $\mathcal{E}_2 = \bigcup_{t=1}^{T} \mathcal{E}_{2,t}$. Therefore, the expected cost of x under $\mathcal{E}_2$ can be expressed as

$$\mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_2\}] = \sum_{t=1}^{T} \mathbf{E}[\mathrm{cost}_p(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_{2,t}\}].$$

We upper bound the expected cost of x under event $\mathcal{E}_2$ by Lemma A.2.

By combining all events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4$, we have that the expected cost of x is at most

$$\begin{aligned}\mathbf{E}[\mathrm{cost}_{\ell_p}(x, \mathcal{T})] &= \mathbf{E}[\mathrm{cost}_{\ell_p}(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_1\}] + \mathbf{E}[\mathrm{cost}_{\ell_p}(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_2\}] + \mathbf{E}[\mathrm{cost}_{\ell_p}(x, \mathcal{T})\, \mathbf{1}\{\mathcal{E}_3 \cup \mathcal{E}_4\}] \\ &\le \left(2 + \frac{A_k}{k}\right) \|x - c\|_p + \sum_{t=1}^{T} \mathbf{E}[\phi_t(\omega_t)\, \mathbf{1}\{\mathcal{E}_{2,t}\}] \\ &\le \left(\frac{A_k}{k} + \beta' \cdot p(\log k)^{1 + \frac{1}{p} - \frac{1}{p^2}} \cdot \log(A_k \log^2 k)\right) \|x - c\|_p,\end{aligned}$$

where β' is an absolute constant. We now proceed to prove Lemma A.2. □

Lemma A.2. *For some absolute constant $\beta > 0$, we have*

$$\sum_{t=1}^{T} \mathbf{E}[\phi_t(\omega_t)\,\mathbf{1}\{\mathcal{E}_{2,t}\}] \le \beta \cdot p(\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \cdot \log(A_k \log^2 k)\|x-c\|_p.$$

Proof. Under the event $\mathcal{E}_2$, the point x and the center c are separated by a cut. We classify the cut that separates x and c into three cases as follows. To do so, we first recall the definitions of light and heavy cuts, as well as safe and unsafe cuts, given in Definitions 3.8 and 3.9.

Fix a parameter $\alpha > 0$ which is specified later. We say that the step t is a *light* step if

$$R_t \le 6\log^{\alpha} k \max\{\|x-m^t\|_p, \|c-m^t\|_p\}.$$

Otherwise, we call it a *heavy* step. Furthermore, if the cut separates x and c at a light step, then we call it a light cut; otherwise, it is a heavy cut. Additionally, at step t, we say that a cut $\omega_t = (i,\vartheta)$ that separates x and c is *safe*, if

$$A_k M_t(i,\vartheta) < \frac{R_t}{6^p \log^2 k}.$$

Otherwise, we call this cut *unsafe*.

Then, we split the analysis into three cases: (1) safe cuts; (2) light and unsafe cuts; (3) heavy and unsafe cuts.

Case 1 (Safe cuts): Suppose the event $\mathcal{E}_{2,t}$ happens and x and c are separated by a safe cut $\omega_t = (i,\vartheta)$. By definition, a safe cut satisfies that the distance from x to the fallback center c' after separation is significantly smaller than the current radius, specifically $A_k M_t(i,\vartheta) < R_t/(6^p \log^2 k)$. In this case, we use $A_k M_t(i,\vartheta)$ as an upper bound on the penalty incurred by separating x and c.

For each step t, coordinate $i \in \{1,2,\cdots,d\}$, and direction $\sigma \in \{-1,1\}$, we define the safe cut set

$$G_{t,i,\sigma} = \left\{\theta : A_k M_t(i, m_i^t + \sigma\theta) < \frac{R_t}{6^p \log^2 k} \,\&\, (i, m_i^t+\sigma\theta) \text{ separates } x \text{ and } c\right\},$$

which contains all parameters θ such that the corresponding cut $\omega_t = (i, m_i^t + \sigma\theta)$ is safe. Then, the expected penalty due to safe cuts is at most

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbf{E}\left[\phi_t(\omega_t)\,\mathbf{1}\{\omega_t \text{ is safe}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le &\sum_{t=1}^{T} \mathbf{E}\left[A_k M_t(i_t, m_i^t+\sigma_t\theta_t)\,\mathbf{1}\{\theta_t \in G_{t,i_t,\sigma_t}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le &\sum_{t=1}^{T}\sum_{i=1}^{d}\sum_{\sigma\in\{-1,1\}} \frac{1}{2d}\int_{G_{t,i,\sigma}} A_k M_t(i, m_i^t+\sigma\theta)\cdot\frac{p\cdot\theta^{p-1}}{R_t^p}\cdot\mathbf{1}\{\mathcal{E}_{2,t}\}\cdot \mathrm{d}\theta \\
= &\sum_{t=1}^{T}\sum_{i=1}^{d}\sum_{\sigma\in\{-1,1\}} \frac{1}{2d}\int_{G_{t,i,\sigma}} \frac{A_k M_t(i, m_i^t+\sigma\theta)}{R_t}\cdot\frac{p\cdot\theta^{p-1}}{R_t^{p-1}}\cdot\mathbf{1}\{\mathcal{E}_{2,t}\}\cdot \mathrm{d}\theta.
\end{aligned}$$

Here, the second inequality uses the fact that the coordinate i is chosen uniformly from $\{1,2,\cdots,d\}$ and the direction σ is chosen uniformly from $\{-1,1\}$ and that θ is drawn from a distribution with density $p\theta^{p-1}/R_t^p$. The safe cuts are those with $\theta \in G_{t,i,\sigma}$.

Now we derive an upper bound for θ/R_t to control the integral. Since center c lies in node u_t, we have $\|c-m^t\|_p \le R_t$. Additionally, since the event $\mathcal{E}_1$ does not occur, we have $R_t \ge \|x-c\|_p$. Using the triangle inequality, we have

$$\|x-m^t\|_p \le \|x-c\|_p + \|c-m^t\|_p \le 2R_t.$$

Therefore, we have

$$3R_t \ge \|x-m^t\|_p + \|c-m^t\|_p.$$

Furthermore, for any $\theta \in G_{t,i,\sigma}$, the cut $(i, m_i^t+\sigma\theta)$ separates x and c, which implies

$$\theta \le \max\{|x_i - m_i^t|, |c_i - m_i^t|\}.$$

Therefore, conditioned on the event $\mathbf{1}\{\mathcal{E}_{2,t}\} = 1$, we have for any $\theta \in G_{t,i,\sigma}$,

$$\frac{\theta}{R_t} \le \frac{3\max\{|x_i - m_i^t|, |c_i - m_i^t|\}}{\|x - m^t\|_p + \|c - m^t\|_p}.$$

We now analyze each partition leaf call separately. Fix a partition leaf call P_s. The expected penalty due to safe cuts within this call is at most

$$\begin{aligned}
&\sum_{t \in P_s} \mathbf{E}\left[\phi_t(\omega_t)\, \mathbf{1}\{\omega_t \text{ is safe}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le &\frac{p}{2d} \sum_{i=1}^{d} \frac{3^{p-1}\max\{|x_i - m_i^s|, |c_i - m_i^s|\}^{p-1}}{(\|x - m^s\|_p + \|c - m^s\|_p)^{p-1}} \sum_{t \in P_s} \sum_{\sigma \in \{-1,1\}} \int_{G_{t,i,\sigma}} \frac{A_k M_t(i, m_i^s + \sigma\theta)}{R_t} \cdot \mathrm{d}\theta.
\end{aligned}$$

By Holder's inequality, the expected penalty above is at most

$$\begin{aligned}
&\frac{p}{2d} \cdot \left(\sum_{i=1}^{d} \frac{3^p \max\{|x_i - m_i^s|, |c_i - m_i^s|\}^p}{(\|x - m^s\|_p + \|c - m^s\|_p)^p}\right)^{\frac{p-1}{p}} \\
&\quad \cdot \left(\sum_{i=1}^{d} \left(\sum_{t \in P_s} \sum_{\sigma \in \{-1,1\}} \int_{G_{t,i,\sigma}} \frac{A_k M_t(i, m_i^s + \sigma\theta)}{R_t} \cdot \mathrm{d}\theta\right)^p\right)^{\frac{1}{p}}.
\end{aligned}$$

Then, we bound the two terms in the above formula separately. First, we have

$$\sum_{i=1}^{d} \max\{|x_i - m_i^s|, |c_i - m_i^s|\}^p \le \|x - m^s\|_p^p + \|c - m^s\|_p^p.$$

Thus, we have the first term is

$$\left(\sum_{i=1}^{d} \frac{3^p \max\{|x_i - m_i^s|, |c_i - m_i^s|\}^p}{(\|x - m^s\|_p + \|c - m^s\|_p)^p}\right)^{\frac{p-1}{p}} \le 3^{p-1}.$$

We now bound the second term. Note that for any fixed cut $\omega = (i, \vartheta)$, the fallback distance $M_t(i, \vartheta)$ is non-decreasing with respect to the step t. Meanwhile, within each partition leaf call P_s, the radius R_t is non-increasing and decreases by a factor of 2 after every L steps under event $\mathcal{E}_2$. Therefore, for each coordinate $i \in \{1, 2, \cdots, d\}$, we have

$$\begin{aligned}
&\sum_{t \in P_s} \sum_{\sigma \in \{-1,1\}} \int_{G_{t,i,\sigma}} \frac{A_k M_t(i, m_i^s + \sigma\theta)}{R_t} \cdot \mathrm{d}\theta \\
\le &\int \sum_{t \in P_s} \sum_{\sigma \in \{-1,1\}} \frac{A_k M_t((i, m_i^s + \sigma\theta))}{R_t}\, \mathbf{1}\{\theta \in G_{t,i,\sigma}\} \cdot \mathrm{d}\theta \\
\le &2L \cdot \frac{1}{6^p \log^2 k} \cdot |x_i - c_i|,
\end{aligned}$$

where the last inequality follows from the definition of safe cuts, which ensures that $A_k M_t(i, \vartheta) < \frac{R_t}{6^p \log^2 k}$ whenever $\theta \in G_{t,i,\sigma}$, and $\frac{A_k M_t(i,\vartheta)}{R_t}$ forms a geometric sequence increases by a factor of 2 every L steps. Therefore, we have the second term is at most

$$\left(\sum_{i=1}^{d} \left(\sum_{t \in P_s} \sum_{\sigma \in \{-1,1\}} \int_{G_{t,i,\sigma}} \frac{A_k M_t(i, m_i^s + \sigma\theta)}{R_t} \cdot \mathrm{d}\theta\right)^p\right)^{\frac{1}{p}} \le 2L \cdot \frac{1}{6^p \log^2 k} \cdot \|x - c\|_p.$$

Since there are at most $O(\log k)$ partition leaf calls and $L = \lceil 2^{p+3} d \ln k \rceil$, the expected penalty due to safe cuts is at most

$$O(\log k) \cdot \frac{p}{2d} \cdot 3^{p-1} \cdot 2L \cdot \frac{1}{6^p \log^2 k} \|x - c\|_p \le O(p) \cdot \|x - c\|_p.$$

Case 2 (Light and unsafe cuts): In this case, we have that the radius R_t is relatively small compared to $\|x - m^t\|_p$ and $\|c - m^t\|_p$, specifically, $R_t \le 6\log^\alpha k \max\{\|x - m^t\|_p, \|c - m^t\|_p\}$. Therefore, in this case, we use R_t as an upper bound on the penalty. Then, the expected penalty due to a light and unsafe cut is

$$\sum_{t=1}^{T} \mathbf{E}\left[\phi_t(\omega_t)\,\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\omega_t \text{ is unsafe}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \le \sum_{t=1}^{T} \mathbf{E}\left[\phi_t(\omega_t)\,\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \le \sum_{t=1}^{T} \mathbf{E}\left[R_t\,\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right].$$

For each step t, suppose both x and c are contained in the node u_t. We define the new event $\mathcal{E}'_t$ as the event that either x or c is first separated from the anchor m^t by the cut chosen at step t. To bound the expected penalty above, we show that

$$\sum_{t=1}^{T} \mathbf{E}\left[R_t\,\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \le 6p\log^\alpha k \cdot \|x - c\|_p \cdot \sum_{t=1}^{T} \mathbf{E}\left[\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}'_t\}\right].$$

To show this, we define the stochastic process $\{Y_t\}_{t\ge 0}$ as follows. Let $Y_0 = 0$ and for any $t \ge 1$,

$$Y_t = \sum_{t'=1}^{t} \left(R_{t'}\,\mathbf{1}\{\mathcal{E}_{2,t'}\} - \|x - c\|_p \cdot 6p\log^\alpha k\,\mathbf{1}\{\mathcal{E}'_{t'}\}\right)\mathbf{1}\{t' \text{ is light}\}.$$

We now show that this stochastic process $\{Y_t\}_{t\ge 0}$ forms a supermartingale. Note that for each step $t \ge 1$, we have

$$Y_t = Y_{t-1} + \left(R_t\,\mathbf{1}\{\mathcal{E}_{2,t}\} - \|x - c\|_p \cdot 6p\log^\alpha k\,\mathbf{1}\{\mathcal{E}'_t\}\right)\mathbf{1}\{t \text{ is light}\}.$$

If step t is heavy, then $Y_t = Y_{t-1}$. In the following analysis, we focus on the case where t is a light step and both x and c are contained in the node u_t. In this case, we first analyze the probability that the chosen cut separates x and c, and the probability that separates either x or c from the anchor m^t.

Claim A.3. *Suppose both x and c are contained in the node u_t at this step t. Then, the probability that x and c are separated by the chosen cut is at most*

$$\Pr\{x \text{ and } c \text{ are separated at step } t \mid \mathcal{T}_{t-1}\} \le \frac{p}{d}\|x - c\|_p \frac{\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1}}{R_t^p}.$$

The probability that either x or c is first separated from m^t by the cut chosen at step t is at least

$$\Pr\{\mathcal{E}'_t \mid \mathcal{T}_{t-1}\} \ge \frac{1}{2d} \cdot \frac{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}}{R_t^p}.$$

Thus, we have for a light step t,

$$\begin{aligned}
&\mathbf{E}[Y_t \mid \mathcal{T}_{t-1}] - Y_{t-1} = R_t \Pr\{\mathcal{E}_{2,t} \mid \mathcal{T}_{t-1}\} - 6p\log^\alpha k \cdot \|x - c\|_p \Pr\{\mathcal{E}'_t \mid \mathcal{T}_{t-1}\} \\
&\le \frac{p}{d}\|x - c\|_p \frac{\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1}}{R_t^{p-1}} - \|x - c\|_p \frac{6p\log^\alpha k}{d} \frac{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}}{R_t^p} \\
&\le \frac{p}{d}\|x - c\|_p \frac{2\max\{\|x - m^t\|_p^{p-1}, \|c - m^t\|_p^{p-1}\}}{R_t^{p-1}} \left(1 - 6\log^\alpha k \frac{\max\{\|x - m^t\|, \|c - m^t\|\}}{R_t}\right) \\
&\le 0,
\end{aligned}$$

where the last inequality follows from the definition of a light step. Therefore, $\{Y_t\}_{t\ge 0}$ is a supermartingale. Hence, $\mathbf{E}[Y_T] \le \mathbf{E}[Y_0] = 0$, which implies

$$\sum_{t=1}^{T} \mathbf{E}\left[R_t\,\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right] \le 6p\log^\alpha k \cdot \|x - c\|_p \cdot \sum_{t=1}^{T} \mathbf{E}\left[\mathbf{1}\{t \text{ is light}\}\,\mathbf{1}\{\mathcal{E}'_t\}\right].$$

To bound the right-hand side, it suffices to control the expected number of times the event $\mathcal{E}'_t$ occurs. Recall that $\mathcal{E}'_t$ denotes the event that either x or c is first separated from the anchor m^t at step t.

We begin by noting that the number of partition-leaf calls is at most $O(\log k)$. Within each partition-leaf call, the anchor point m^t remains fixed, and once x is separated from m^t, it will no longer be involved in further cuts associated with that anchor. Therefore, x can be separated from m^t in at most one step per partition-leaf call, contributing at most $O(\log k)$ occurrences of $\mathcal{E}'_t$. Additionally, observe that the center c can be separated from the anchor m^t without x being separated at most once. After such a separation, c will no longer lie in the same node as x and will not contribute to future events $\mathcal{E}'_t$.

Combining these observations, we conclude that the expected number of steps where $\mathcal{E}'_t$ occurs is at most $O(\log k)$, which yields

$$\mathbf{E}\left[\sum_{t=1}^{T} \phi_t(\omega_t)\, \mathbf{1}\{t \text{ is light}\}\, \mathbf{1}\{\omega_t \text{ is unsafe}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right] \le O\left(p \log^{1+\alpha} k\right) \|x - c\|_p.$$

Case 3 (Heavy and unsafe cuts): Suppose the event $\mathcal{E}_{2,t}$ occurs and that x and c are separated by an unsafe cut $\omega_t = (i, \vartheta)$. For each step t, coordinate $i \in \{1, 2, \cdots, d\}$, and direction $\sigma \in \{-1, 1\}$, we define the the corresponding unsafe cut set as

$$U_{t,i,\sigma} = \left\{\theta : A_k M_t(i, m_i^t + \sigma\theta) \ge \frac{R_t}{6^p \log^2 k} \,\&\, (i, m_i^t + \sigma\theta) \text{ separates } x \text{ and } c\right\},$$

that is, the set of threshold θ for which the cut $(i, m_i^t + \sigma\theta)$ is both unsafe and separates x from c. Let $\delta_{i,\sigma}(t) = \mu(U_{t,i,\sigma})$ denote the Lebesgue measure of the set $U_{t,i,\sigma}$ and define $\delta_i(t) = \delta_{i,-1}(t) + \delta_{i,1}(t)$ as the total measure across both directions for coordinate i.

Thus, the probability that ω_t is an unsafe cut is at most

$$\begin{aligned}
\Pr\{\omega_t \text{ is unsafe}\} =& \frac{1}{2d} \int_{U_{t,i,\sigma}} \frac{p \cdot \theta^{p-1}}{R_t^p} \cdot \mathrm{d}\theta \\
\le& \frac{p}{2d} \sum_{i=1}^{d} \sum_{\sigma \in \{-1,1\}} \frac{\max\{|x_i - m_i^t|, |c_i - m_i^t|\}^{p-1}}{R_t^p} \cdot \delta_{i,\sigma}(t) \\
=& \frac{p}{2d} \sum_{i=1}^{d} \frac{\max\{|x_i - m_i^t|, |c_i - m_i^t|\}^{p-1}}{R_t^p} \cdot \delta_i(t).
\end{aligned}$$

In this case, we use the radius R_t as the upper bound on the penalty for separating x and c. Therefore, the expected penalty incurred from heavy and unsafe cuts is bounded by

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbf{E}\left[\phi_t(\omega_t)\, \mathbf{1}\{\omega_t \text{ is unsafe}\}\, \mathbf{1}\{t \text{ is heavy}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le& \sum_{t=1}^{T} \mathbf{E}\left[R_t \Pr\{\omega_t \text{ is unsafe}\}\, \mathbf{1}\{t \text{ is heavy}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le& \sum_{t=1}^{T} \mathbf{E}\left[R_t \sum_{i=1}^{d} \frac{p}{2d}\, \delta_i(t) \frac{\max\{|x_i - m_i^t|, |c_i - m_i^t|\}^{p-1}}{R_t^p}\, \mathbf{1}\{t \text{ is heavy}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right].
\end{aligned}$$

Since step t is heavy, we have $R_t \ge 6 \log^\alpha k \cdot \max\{\|x - m^t\|_p, \|c - m^t\|_p\}$, which implies

$$\frac{1}{R_t^{p-1}}\, \mathbf{1}\{t \text{ is heavy}\} \le \frac{1}{(6 \log^\alpha k)^{p-1} \max\{\|x - m_t\|_p, \|c - m_t\|_p\}^{p-1}}.$$

Substituting this into the previous bound, we obtain that the expected penalty in this case is at most

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbf{E}\left[\phi_t(\omega_t)\, \mathbf{1}\{\omega_t \text{ is unsafe}\}\, \mathbf{1}\{t \text{ is heavy}\}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right] \\
\le& \sum_{t=1}^{T} \mathbf{E}\left[\sum_{i=1}^{d} \frac{p}{2d}\, \delta_i(t) \frac{\max\{|x_i - m_i^t|, |c_i - m_i^t|\}^{p-1}}{(6 \log^\alpha k)^{p-1} \max\{\|x - m_t\|_p, \|c - m_t\|_p\}^{p-1}}\, \mathbf{1}\{\mathcal{E}_{2,t}\}\right].
\end{aligned}$$

Note that all steps within the same partition leaf call P_s share the same anchor point. Let $\bar{m}^s$ denote the anchor point used in the partition leaf call P_s, and define $\Delta_i(s) = \sum_{t \in P_s} \delta_i(t)$. Then, the expected penalty above is at most

$$\begin{aligned}
&\mathbf{E}\left[\sum_{s=1}^{S}\sum_{t\in P_s}\sum_{i=1}^{d}\frac{p}{2d}\,\delta_i(t)\frac{\max\{|x_i-\bar{m}_i^s|,|c_i-\bar{m}_i^s|\}^{p-1}}{(6\log^\alpha k)^{p-1}\max\{\|x-\bar{m}^s\|_p,\|c-\bar{m}^s\|_p\}^{p-1}}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right]\\
\le&\,\mathbf{E}\left[\sum_{s=1}^{S}\frac{p}{2d}\sum_{i=1}^{d}\Delta_i(s)\frac{\max\{|x_i-\bar{m}_i^s|,|c_i-\bar{m}_i^s|\}^{p-1}}{(6\log^\alpha k)^{p-1}\max\{\|x-\bar{m}^s\|_p,\|c-\bar{m}^s\|_p\}^{p-1}}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right].
\end{aligned}$$

Let $\Delta(s)$ denote the d-dimensional vector with coordinates $\Delta_i(s)$ for $i \in \{1, 2, \cdots, d\}$. Applying Hölder's inequality, we get

$$\begin{aligned}
&\sum_{t=1}^{T}\mathbf{E}\left[\phi_t(\omega_t)\,\mathbf{1}\{\omega_t \text{ is unsafe}\}\,\mathbf{1}\{t \text{ is heavy}\}\,\mathbf{1}\{\mathcal{E}_{2,t}\}\right]\\
\le&\,\mathbf{E}\left[\sum_{s=1}^{S}\frac{p}{2d}\|\Delta(s)\|_p\frac{\left(\sum_{i=1}^{d}|x_i-\bar{m}_i^s|^p\right)^{\frac{p-1}{p}}+\left(\sum_{i=1}^{d}|c_i-\bar{m}_i^s|^p\right)^{\frac{p-1}{p}}}{(6\log^\alpha k)^{p-1}\max\{\|x-\bar{m}^s\|_p,\|c-\bar{m}^s\|_p\}^{p-1}}\,\mathbf{1}\{\mathcal{E}_2\}\right]\\
\le&\,\mathbf{E}\left[\sum_{s=1}^{S}\frac{p}{2d}\|\Delta(s)\|_p\frac{\|x-\bar{m}^s\|_p^{p-1}+\|c-\bar{m}^s\|_p^{p-1}}{(6\log^\alpha k)^{p-1}\max\{\|x-\bar{m}^s\|_p,\|c-\bar{m}^s\|_p\}^{p-1}}\,\mathbf{1}\{\mathcal{E}_2\}\right]\\
\le&\,\frac{p}{(6\log^\alpha k)^{p-1}d}\,\mathbf{E}\left[\sum_{s=1}^{S}\|\Delta(s)\|_p\cdot\mathbf{1}\{\mathcal{E}_2\}\right].
\end{aligned}$$

Finally, we use the following claim to bound the expected penalty.

Claim A.4. *We have*

$$\mathbf{E}\left[\sum_{s=1}^{S}\|\Delta(s)\|_p\cdot\mathbf{1}\{\mathcal{E}_2\}\right] = O\left(4^p\cdot d\cdot\log^{2-\frac{1}{p}}k\cdot\log(6^p\cdot A_k\cdot\log^2 k)\right)\|x-c\|_p.$$

By Claim A.4, we have that the expected penalty in this case is at most

$$\begin{aligned}
&\frac{p}{(6\log^\alpha k)^{p-1}d}\,\mathbf{E}\left[\sum_{s=1}^{S}\|\Delta(s)\|_p\cdot\mathbf{1}\{\mathcal{E}_{2,t}\}\right]\\
\le&\,\frac{p}{(6\log^\alpha k)^{p-1}d}\cdot O\left(4^p\cdot d\cdot\log^{2-\frac{1}{p}}k\cdot\log(6^p\cdot A_k\cdot\log^2 k)\right)\|x-c\|_p\\
\le&\,O\left((\log k)^{2-\frac{1}{p}-\alpha(p-1)}\cdot\log(A_k\cdot\log^2 k)\right)\|x-c\|_p.
\end{aligned}$$

Combining all three cases, we get the conclusion.

To complete the proof, we prove Claim A.3 and A.4 below. □

Proof of Claim A.3. We first analyze the probability that x and c are separated by the cut chosen at step t. To bound the separation probability, we fix a coordinate $i \in \{1, 2, \cdots, d\}$ and consider the probability that the cut on coordinate i separates x and c.

Suppose x and c are on the same side of anchor m^t in coordinate i. Then, the threshold cut $\omega_t = (i, m_i^t + \sigma\theta)$ separates x and c if and only if σ has the same sign as $x_i - m_i^t$ and θ is between $|x_i - m_i^t|$ and $|c_i - m_i^t|$. Thus, the separation probability on this coordinate is at most

$$\frac{1}{2}\cdot\frac{||c_i-m_i^t|^p-|x_i-m_i^t|^p|}{R_t^p}\le\frac{p\cdot\max\{|x_i-m_i^t|^{p-1},|c_i-m_i^t|^{p-1}\}}{R_t^p}\cdot|x_i-c_i|,$$

where the inequality is from the mean value theorem.

Suppose x and c are on the opposite side of anchor m^t in coordinate i. Then, the separation probability on this coordinate is at most

$$\frac{1}{2} \cdot \frac{|c_i - m_i^t|^p + |x_i - m_i^t|^p}{R_t^p} \leq \frac{p \cdot \max\{|x_i - m_i^t|^{p-1}, |c_i - m_i^t|^{p-1}\}}{R_t^p} \cdot |x_i - c_i|.$$

Combining all coordinates and applying the Holder inequality, we obtain

$$\begin{aligned}
&\frac{1}{d} \sum_{i=1}^{d} \frac{p \cdot \max\{|x_i - m_i^t|^{p-1}, |c_i - m_i^t|^{p-1}\}}{R_t^p} \cdot |x_i - c_i| \\
\leq &\frac{p}{d \cdot R_t^p} \|x - c\|_p \cdot \left(\left(\sum_{i=1}^{d} |x_i - m_i^t|^p \right)^{\frac{p-1}{p}} + \left(\sum_{i=1}^{d} |c_i - m_i^t|^p \right)^{\frac{p-1}{p}} \right) \\
\leq &\frac{p}{d} \|x - c\|_p \cdot \frac{\|x - m^t\|_p^{p-1} + \|c - m^t\|_p^{p-1}}{R_t^p}.
\end{aligned}$$

For point x, the probability that it is separated from m^t at step t is given by

$$\frac{1}{2d} \sum_{i=1}^{d} \frac{|x_i - m_i^t|^p}{R_t^p} = \frac{1}{2d} \cdot \frac{\|x - m^t\|_p^p}{R_t^p}.$$

An identical argument applies to the center c, yielding the same expression with $\|c - m^t\|_p^p$. Therefore, the probability that either x or c is separated from m^t by the threshold cut at step t is at least

$$\frac{1}{2d} \cdot \frac{\max\{\|x - m^t\|_p^p, \|c - m^t\|_p^p\}}{R_t^p},$$

as claimed. □

To prove Claim A.4, we first show the following lemma.

Lemma A.5. *For k vectors $v^1, \cdots, v^k \in \mathbb{R}^d$ that are entrywise non-negative, we have*

$$\sum_{i=1}^{k} \|v^i\|_p \leq k^{1-\frac{1}{p}} \cdot \left\| \sum_{i=1}^{k} v^i \right\|_p.$$

Proof. We first upper bound the left-hand side. By Hölder's inequality, we have

$$\sum_{i=1}^{k} \|v^i\|_p = \sum_{i=1}^{k} 1 \cdot \|v^i\|_p \leq k^{\frac{1}{q}} \left(\sum_{i=1}^{k} \|v^i\|_p^p \right)^{\frac{1}{p}} = k^{1-\frac{1}{p}} \left(\sum_{i=1}^{k} \|v^i\|_p^p \right)^{\frac{1}{p}}.$$

We then lower bound the right-hand side. Since vectors $v^1, \cdots, v^k$ are nonnegative in every coordinate, we have for any coordinate j,

$$\left(\sum_{i=1}^{k} v_j^i \right)^p \geq \sum_{i=1}^{k} (v_j^i)^p.$$

Combining all coordinates, we have

$$\left\| \sum_{i=1}^{k} v_i \right\|_p^p = \sum_{j=1}^{d} \left(\sum_{i=1}^{k} v_j^i \right)^p \geq \sum_{j=1}^{d} \sum_{i=1}^{k} (v_j^i)^p = \sum_{i=1}^{k} \|v_i\|_p^p.$$

Combining the two parts, we get the conclusion. □

Proof of Claim A.4. By Lemma A.5 and the number of partition leaf calls is at most $O(\log k)$, we have

$$\mathbf{E}\left[\sum_{s=1}^{S}\|\Delta(s)\|_p \cdot \mathbf{1}\{\mathcal{E}_2\}\right] \le O\left(\log^{1-\frac{1}{p}} k\right)\mathbf{E}\left[\left\|\sum_{s=1}^{S}\Delta(s)\right\|_p \cdot \mathbf{1}\{\mathcal{E}_2\}\right].$$

For any fixed coordinate i, we have

$$\sum_{s=1}^{S}\Delta_i(s) = \sum_{s=1}^{S}\sum_{t\in P_s}\delta_i(t) = \sum_{t=1}^{T}\delta_i(t) = \sum_{t=1}^{T}\int \mathbf{1}\{\theta \in U_{t,i,1}\}\mathrm{d}\theta + \int \mathbf{1}\{\theta \in U_{t,i,-1}\}\mathrm{d}\theta.$$

We now show that each cut $\omega = (i, \vartheta)$ that separates x and c is unsafe in at most $L'' = L' \cdot \log(6^p \cdot A_k \cdot \log^2 k)$ steps. Consider any cut $\omega = (i, \vartheta)$ that separates x and c. This cut ω is unsafe at step t if and only if $R_t \le 6^p \log^2 k \cdot A_k M_t(i, \vartheta)$. For every step t, by the triangle inequality, the penalty to the fallback center is at most $M_t(i, \vartheta) \le D_t \le \widetilde{D}_t$. We know that $M_t(i, \vartheta)$ is non-decreasing as t increases. Let t_ω be the first step when ω is unsafe. Let t'_ω be the last step when ω is unsafe. Then, by the definition of unsafe cut, we have $R_{t_\omega} \le 6^p \log^2 k \cdot A_k M_{t_\omega}(i, \vartheta)$. Then, we have

$$\widetilde{D}_{t'_\omega} \ge M_{t'_\omega}(\omega) \ge M_{t_\omega}(\omega) \ge \frac{R_{t_\omega}}{6^p \cdot \log^2 k \cdot A_k}.$$

Since $R_t/2^{1/p} \le \widetilde{D}_t \le 2R_t$, we have

$$\widetilde{D}_{t'_\omega} \ge \frac{R_{t_\omega}}{6^p \cdot \log^2 k \cdot A_k} \ge \frac{\widetilde{D}_{t_\omega}}{2 \cdot 6^p \cdot \log^2 k \cdot A_k}.$$

By Lemma 3.7, we have $\widetilde{D}_t$ decreases by a factor of 2 after $L' = \lceil 2^{2p+5} d \ln k \rceil$ steps. Thus, we have that the number of unsafe steps is at most

$$t'_\omega - t_\omega \le L' \cdot \log_2(2 \cdot 6^p \cdot \log^2 k \cdot A_k) \le O(4^p \cdot d \cdot \log k \cdot p \log(A_k \cdot \log^2 k)).$$

Therefore, we have that when the event $\mathcal{E}_2$ happens,

$$\sum_{s=1}^{S}\Delta_i(s) \le O(4^p \cdot d \cdot \log k \cdot \log(6^p \cdot A_k \cdot \log^2 k))|x_i - c_i|.$$

Hence, combining all coordinates, we have

$$\mathbf{E}\left[\left\|\sum_{s=1}^{S}\Delta^u(s)\right\|_p \mathbf{1}\{\mathcal{E}_2\}\right] = O(4^p \cdot d \cdot \log k \cdot \log(6^p \cdot A_k \cdot \log^2 k))\|x - c\|_p,$$

which completes the proof. □

B Dynamic algorithm implementation and analysis

In this section, we provide the full description of the dynamic algorithm, along with an analysis of its approximation guarantee, update time, and recourse.

B.1 Dynamic algorithm and approximation guarantee

We begin by presenting the detailed dynamic algorithm and proving that, after each update, the distribution of its output is equivalent to that of a corresponding static algorithm.

Lemma B.1. *Given a stream of updates that generates a sequence of center sets $C_1, C_2, \ldots$, let $\mathcal{T}_k$ be the threshold tree maintained by the dynamic algorithm for the center set C_k. Let $\mathcal{T}'_k$ be the tree constructed by the static algorithm* PARTITION_LEAF *(Figure 1) with specific oracles on centers C_k. Then, the two trees are identically distributed $\mathcal{T}_k \overset{d}{=} \mathcal{T}'_k$.*

The following corollary is immediate from Lemma B.1 and Theorem 3.1.

Corollary B.2. *The dynamic algorithm achieves an* $O\left(p(\log k)^{1+\frac{1}{p}-\frac{1}{p^2}} \log\log k\right)$ *approximation.*

We provide a dynamic implementation of the PARTITION_LEAF procedure in Figure 2, which is applied recursively to obtain a fully dynamic version of the entire clustering algorithm. The dynamic variant of PARTITION_LEAF supports three operations: (1) REBUILD, (2) INSERT CENTER, and (3) DELETE CENTER.

We begin with the REBUILD operation, which reconstructs the subtree from scratch using the PARTITION_LEAF procedure as follows.

REBUILD: Reconstruct the subtree rooted at node u, partitioning all centers in C_u into distinct leaves via recursive calls to the PARTITION_LEAF procedure. During each such PARTITION_LEAF call on node v in this operation, the following oracle outputs are used and remain fixed throughout subsequent updates until the next rebuild:

- GET_ANCHOR sets the anchor m^v as the coordinate-wise median of centers in C_v.
- STOPPING_ORACLE determines whether to stop accepting further cuts based on a stopping time ρ^v. It returns True if and only if the the timestamp ρ of the input cut ω satisfies $\rho > \rho^v$. The stopping time ρ^v is defined during the rebuild as the timestamp of the last accepted cut such that the main part v_0 contains at most half of centers in C_v, i.e. $|C_{v_0}| \le |C_v|/2$.

We now describe the condition under which the rebuild operation is triggered in the dynamic algorithm. Let u be the node on which this operation is applied. Suppose a center c is inserted into or deleted from the set of centers assigned to u. For each partition leaf call, we maintain a counter that tracks the number of such updates since the last rebuild. Let k' be the number of centers in node u at the time of the last rebuild. When the update count exceeds $k'/4$, we rebuild the partial tree rooted at node u.

We now proceed to handle the update.

INSERT CENTER: Suppose a new center c is inserted. The algorithm calls GET_EARLIEST_CUT to find the earliest cut ω in the pre-generated sequence with its arrival time ρ that separates c from the anchor m^u. Let $(\omega'_1, \rho'_1), \cdots, (\omega'_r, \rho'_r)$ be the cuts currently used in this partition leaf call. Let ρ^u be the stopping time assigned to this partition leaf call during its most recent rebuild. We consider three cases as follows: (1) $\rho = \rho'_j$ for some $j \in [r]$; (2) $\rho > \rho^u$; (3) $\rho \le \rho^u$ and $\rho \ne \rho'_j$ for any $j \in [r]$.

Case (1): Assign this new center c to the node v generated by cut ω'_j and recursively maintain the partition leaf call rooted at v.

Case (2): This new center c remains in the main part u_0 until this partition leaf call ends. We then recursively maintain the partition leaf call on the main part u_0.

Case (3): It finds the smallest index $j \in [r]$ such that $\rho < \rho'_j$ or sets $j = r + 1$ if no such index exists. Then we insert this new cut ω at position j and add a new leaf node containing c to the tree.

DELETE CENTER: Now suppose a center $c \in C_u$ is deleted. We locate the leaf node containing c in this partition leaf call. If this leaf contains only one center c, we remove both the leaf and the cut that created it. Otherwise, we delete c from the leaf and maintain the next partition call recursively.

Proof of Lemma B.1. We describe an implementation of the static algorithm on the set of centers C_k, using specific oracles GET_ANCHOR and STOPPING_CONDITION.

To couple with the dynamic algorithm, we mirror each partition leaf call currently maintained in the dynamic algorithm solution. We begin with the partition leaf call at the root node. Let $t \le k$ denote the time of the most recent rebuild of this root partition leaf as of time k, and let $k' = |C_t|$ be the number centers present at that rebuild time. Assume both the dynamic and static algorithms use the same infinite sequence of candidate cuts with associated timestamps for the root PARTITION_LEAF call.

For any fixed sequence of cuts with timestamps, let m^r be the anchor and ρ^r be the stopping time used by the dynamic algorithm for this root partition leaf. In the static algorithm, we adopt the same oracles as the dynamic one: the oracle GET_ANCHOR returns m^r and STOPPING_ORACLE returns True if and only if the timestamp of the input cut exceeds ρ^r. As a result, the static algorithm accepts

Algorithm DYNAMIC_PARTITION_LEAF

Input: A sequence of updates $Q = (q_1, q_2, \dots)$, where each q_t is either: INSERT(c): insert a new center $c \in \mathbb{R}^d$; or DELETE(c): delete a center c.
Output: For each update q_t in Q, maintain a threshold tree $\mathcal{T}_t$ over the current center set C_t.
Main(Q)**:**

1. Initialize the root r to be empty.
2. For each update q_t at time t:
   - **If** q_t is INSERT(c): call INSERT_CENTER(c, r) where r is the root.
   - **If** q_t is DELETE(c): call DELETE_CENTER(c, r) where r is the root.
   - Output the updated tree $\mathcal{T}_t$.

Procedure REBUILD(u):

1. Let C_u be the current center set at u, set anchor m^u be the coordinate-wise median of centers C_u. Initialize the main part $u_0 = u$.
2. Initialize an update counter at u to be $\mathrm{Cnt}_u = 0$ and set $k_u = |C_u|$.
3. Compute a sequence of candidate cuts $\{(\omega_t, \rho_t)\}$ using an exponential clock:
   For each t, sample $i_t \in [d]$, $\sigma_t \in \{-1, 1\}$, $Z_t \sim \mathrm{Unif}[0, 2^p]$. Define the cut $\omega_t = (i_t, \vartheta_t)$, where $\vartheta_t = m_{i_t} + \sigma_t (Z_t)^{1/p}$. Assign the timestamps ρ_t as the arrival times of a Poisson process.
4. Iterate over the cuts ω_t in increasing order of their timestamps ρ_t. Accept it iff ω_t separates two centers in the main part u_0. After each accepted cut, update the main part u_0 to the side containing the anchor. Stop when the main part contains fewer than $|C_u|/2$ centers. Then, set the stopping time ρ^u to be the timestamp of the last accepted cut,
   $$\rho^u = \max\{\rho_t : \text{cut } \omega_t \text{ is accepted}\}.$$
5. Call REBUILD(v) for each leaf v containing more than one center in the subtree rooted at u.

Procedure INSERT_CENTER(c, u):

1. Increment update counter at u; if updates exceed $k_u/4$, call REBUILD(u).
2. Get the earliest cut $(\omega, \rho) =$ GET_EARLIEST_CUT(c) that separates c and m^u.
3. Let $(\omega'_1, \rho'_1), \dots, (\omega'_r, \rho'_r)$ be cuts used by u and ρ^u be the stopping time.
4. **If** $\rho = \rho'_j$ for some j:
   Assign c to node v separated by the cut ω'_j, and call INSERT_CENTER(c, v).
5. **If** $\rho > \rho^u$: Assign c to the main part u_0, and call INSERT_CENTER(c, u_0).
6. **If** $\rho \le \rho^u$ and $\rho \ne \rho'_j$ for every j:
   Insert new cut ω into the sequence of cuts used by u, maintaining increasing order by ρ. Create a new leaf node containing c, and attach it to the tree at the cut point.

Procedure DELETE_CENTER(c, u):

1. Increment update counter at u; if updates exceed $k_u/4$, call REBUILD(u).
2. Locate the leaf node v containing c.
3. **If** the leaf contains only c: Remove both the leaf and its parent cut.
4. **Else**: Delete c from the leaf v and call DELETE_CENTER(c, v).

Figure 2: Dynamic algorithm for explainable k-medians in ℓ_p

exactly the same sequence of cuts as the dynamic algorithm. Therefore, the partial tree rooted at r produced by this PARTITION_LEAF call in the static algorithm is identical to that maintained by the dynamic algorithm. We will show that these two oracles are valid for the static algorithm, which means they satisfy the required properties in Section 2.

We first show that GET_ANCHOR returns an approximate median of centers C_k. Because this is the most recent rebuild of the root node r, there have been fewer than $k'/4$ updates since then. Note that the anchor m^r is chosen as the coordinate-wise median of all centers in C_t at time t. For each coordinate i, at most half of the centers in C_t lie on either side of m^r. Hence, even after $k'/4$ updates, there remain at most $3k/4$ centers in C_k on either side of m^r along every coordinate.[3] Therefore, the anchor m^r remains an approximate median for the current set of centers C_k.

We next show that the STOPPING_ORACLE guarantees that when partitioning stops, every leaf contains at most a $3/4$ fraction of centers in C_k. Consider any leaf that is separated from the main part during the partitioning. Each such leaf contains only centers that lie on one side of the anchor m^r along the coordinate used by the cut that separates it. Since the anchor m^r is an approximate median of centers in C_k, at most $3k/4$ centers lie on either side of m^r along every coordinate. Therefore, each separated leaf contains at most a $3/4$ fraction of centers in C_k. As for the main part, recall that at the stopping time ρ^r during the last rebuild, it contains at most $k'/2$ centers in C_t. After at most $k'/4$ updates, the main part contains at most a $3/4$ fraction of centers in C_k.

At each recursive step, we use the same sequence of cuts and adopt the corresponding anchor and stopping time used by the dynamic algorithm. This guarantees that the static algorithm mirrors the behavior of the dynamic one at every level of the recursion. Therefore, the static algorithm constructs exactly the same threshold tree as the dynamic algorithm. This completes the coupling argument and establishes that the output of the dynamic algorithm is identically distributed to that of the static algorithm on input C_k.

□

B.2 Efficient implementation and analysis

In this section, we present a practical implementation of dynamic algorithm as shown in Figure 2. We evaluate the efficiency of the algorithm from two perspectives: update time and recourse.

First, the update time at request q_t refers to the time required to modify the threshold tree $\mathcal{T}_{t-1}$ in response to the t-th request q_t (either an insertion or deletion of a center), resulting in a new tree $\mathcal{T}_t$. Second, the recourse at request q_t is defined to be the number of nodes that differ between $\mathcal{T}_{t-1}$ and $\mathcal{T}_t$, i.e., the size of their symmetric difference between the two trees.

We focus on bounding these quantities in the amortized sense, i.e., the total update time and total recourse over all k requests, averaged across the requests. The following lemma summarizes the performance guarantees of the dynamic algorithm.

Lemma B.3. *Given a sequence of k requests, where each request is either an insertion or a deletion of a single center, the dynamic algorithm satisfies the following guarantees*

1. *the amortized recourse is $O(\log k)$.*
2. *the amortized update time is $O(d \log^3 k)$.*

We first describe an efficient implementation of the dynamic algorithm. For each node u where REBUILD is called, we maintain a self-balancing binary search tree that stores all cuts with timestamps $(\omega'_1, \rho'_1), (\omega'_2, \rho'_2), \ldots (\omega'_r, \rho'_r)$ used in the partial tree rooted at u. This data structure enables efficient updates. When a new request arrives to insert or delete a center c, we call GET_EARLIEST_CUT(c) to compute the earliest cut that separates c from the anchor m^u, and then search the binary search tree to locate where this separation occurs in the partition leaf path of u.

We now describe an efficient implementation of the function GET_EARLIEST_CUT. Without loss of generality, we assume that all centers are in $[-1, 1]^d$. The function GET_EARLIEST_CUT takes

[3] Consider any fixed coordinate i and one side of m^r_i. The fraction of centers lying on this side of m^r is maximized when all $k'/4$ updates remove centers from the opposite side. Thus, the fraction of centers lying on this side is at most $2/3$ after updates.

a center c as input and outputs the earliest cut (ω, ρ) that separates c from the anchor m^u among a sequence of candidate cuts $(\omega_1, \rho_1), (\omega_2, \rho_2), \ldots$. Each cut $\omega_t = (i_t, \vartheta_t)$ is generated by sampling a coordinate i, a sign $\sigma \in \{-1, 1\}$ uniformly at random, and a parameter $\theta \in [0, 2]$ drawn from the distribution with density $f(x) = px^{p-1}/2^p$. The threshold is then set as $\vartheta_t = m_i^u + \sigma\theta$. The associated timestamps ρ_t follow the arrival times of a Poisson Process with rate $\lambda = 1$.

To facilitate efficient implementation, we first observe that the problem naturally decomposes across coordinates. Specifically, for each coordinate $i \in \{1, 2, \cdots, d\}$, we can independently maintain and query the earliest cut that separates c from m^u along coordinate i. We then return the cut with the minimum timestamp across all coordinates.

To achieve this, we maintain an independent stream of candidate cuts for each pair of coordinate i and direction $\sigma \in \{-1, 1\}$. Each such stream consists of cuts $\omega = (i, \vartheta)$ where $\vartheta = m_i^u + \sigma\theta$ and the timestamps given by the arrival times of a Poisson process with rate $1/2d$. This decomposition is formally justified by the Coloring Theorem (see, e.g. Kingman (1992), page 53), which states:

Theorem B.4 (Coloring Theorem)**.** *Let Π_t be a Poisson process on the real line with rate λ. Assign to each event of the process a color from a finite set $\{1, \cdots, M\}$, where each event is independently colored with probability p_i of receiving color i. Then the counts of events of each color, $\Pi_1, \cdots, \Pi_M$, form independent Poisson processes, with rates $\lambda p_1, \cdots, \lambda p_M$, respectively.*

The original sequence of candidate cuts has timestamps given by the arrival times of a Poisson process with rate 1. Each cut is independently assigned a pair (i, σ) with uniform probability $1/2d$ over all $2d$ possible combinations. By the Coloring Theorem, the subset of cuts corresponding to any fixed pair (i, σ) forms an independent Poisson process with rate $1/2d$ and these $2d$ streams are independent. Therefore, the union of all these subsequences of cuts has the same distribution as the original sequence of candidate cuts.

We then formulate the earliest cut along each coordinate as the following general problem. We are given a fixed anchor value $m \in [-1, 1]$, and a sequence of random cuts specified by thresholds ϑ_t drawn from $[m, m + 2]$ according to a probability density function $f(x)$, with associated timestamps ρ_t, corresponding to the arrival times of a Poisson process with rate λ_0. For a query point $y \in [m, 1]$, we aim to find the earliest cut that separates y and m, i.e., the cut with the smallest timestamp such that its threshold ϑ_t lies in $(m, y]$. This formulation arises naturally in our setting, where $\lambda_0 = 1/2d$, the density function $f(x) = p(x - m)^{p-1}/2^p$, m represents the i-th coordinate of the anchor, and y corresponds to the i-th coordinate of some center c. A simple approach for solving this problem is to simulate the sequence of cuts with timestamps and return the first one that lies in $(m, y]$. We refer to this as the static algorithm.

We now describe a data structure that efficiently retrieves the earliest cut along a given coordinate. This data structure maintains a self-balancing binary search tree. Given an anchor m and a set of values $m \le y_1 < y_2 < \cdots < y_k \le 1$, this binary search tree maintains these values in increasing order. Each node in the binary search tree stores a value y along with the earliest cut that separates y from the anchor m, including the timestamp of that cut. If the queried value y is present in the tree, the associated earliest separating cut can be retrieved in $O(\log k)$ time.

Now suppose we need to insert a new value $y \in [m, 1]$ into this data structure. Assume the binary search tree currently stores k values $m \le y_1 < y_2 < \cdots < y_k \le 1$. We first locate the position of y in the tree in $O(\log k)$ time, either identifying the smallest index j such that $y < y_j$, or determining that $y > y_k$. Let $y_0 = m$. If there exists some $1 \le j \le k$ such that $y_{j-1} < y < y_j$, then we first retrieve the earliest cut (ϑ, ρ) that separates y_j from m. We consider two different cases:

1. $y_{j-1} < y < y_j$ for some $1 \le j \le k$ and (ϑ, ρ) also separates y from m, (i.e. $\vartheta \le y$);
2. either $y_{j-1} < y < y_j$ and (ϑ, ρ) does not separates y from m (i.e. $\vartheta > y$) or $y_k < y \le 1$.

For the first case, we store this cut (ϑ, ρ) at the node y as the earliest cut that separates y from m.

For the second case, we first sample a new cut as follows. If $y \ge y_k$, then let $y_{j-1} = y_k$. Sample a new threshold $\vartheta' \in (y_{j-1}, y]$ using the weighted density function

$$\tilde{f}(x) = \frac{f(x)}{\Pr\{\vartheta \in (y_{j-1}, y]\}} = \frac{f(x)}{\int_{y_{j-1}}^{y} f(t)\mathrm{d}t}, \; x \in (y_{j-1}, y].$$

We then sample a timestamp for this cut as $\rho' = \rho + z$, if $y \le y_k$, otherwise if $y > y_k$, $\rho' = z$, where $z \sim \exp(\lambda)$ with rate

$$\lambda = \lambda_0 \cdot \Pr\{\vartheta \in (y_{j-1}, y]\} = \lambda_0 \cdot \int_{y_{j-1}}^{y} f(t)\mathrm{d}t,$$

where λ_0 is a parameter of the data structure. Let (ϑ'', ρ'') be the earliest cut that separates y_{j-1} from m. We then compare the two cuts and store at node y the one with the smaller timestamp. If $\rho' < \rho''$, then we store the new cut (ϑ', ρ') at node y as the earliest cut; otherwise, we store the cut (ϑ'', ρ'').

Lemma B.5. *Given a sequence of query points $y_1, y_2, \cdots$, the earliest cuts maintained by the data structure are distributed identically to those returned by the static algorithm.*

Proof. We prove this lemma by induction. For the first query point, the data structure and the static algorithm samples the earliest cut that separates this point from the same distribution. We now assume that for the first k query points $y_1, \cdots, y_k$, the earliest cuts returned by the data structure are distributed identically to those returned by the static algorithm. By coupling these two algorithms, we further assume that the data structure and the static algorithm return exactly the same earliest cuts for these query points.

We now consider a new query point y_{k+1} and argue that the earliest cuts returned by two algorithms are distributed identically. Let $y_{(1)}, y_{(2)}, \cdots, y_{(k)}$ be the first k query points sorted in increasing order. Let $y_{(0)} = m$. Suppose this new query point is in the first case, which means there exists $1 \le j \le k$ such that $y_{(j-1)} < y_{k+1} < y_{(j)}$ and the earliest cut (ϑ_t, ρ_t) that separates $y_{(j)}$ maintained by the data structure also separates y_{k+1}. Since $y_{k+1} < y_{(j)}$ and this cut (ϑ_t, ρ_t) is the earliest cut that separates $y_{(j)}$ in the static algorithm, this cut is also the earliest cut for y_{k+1} returned by the static algorithm.

We now consider this new query point is in the second case, either $y_{(j-1)} < y_{k+1} < y_{(j)}$ and the earliest cut (ϑ_t, ρ_t) that separates $y_{(j)}$ does not separates y_{k+1} or $y_{(k)} < y_{k+1} \le 1$. If $y_{k+1} > y_{(k)}$, we set $y_{(j-1)} = y_{(k)}$. We can decompose the sequence of cuts used in the static algorithm into three disjoint subsequences. These three subsequences contain all cuts in three disjoint intervals $(m, y_{(j-1)}]$, $(y_{(j-1)}, y_{k+1}]$, and $(y_{k+1}, m+2]$ respectively. By the Coloring Theorem, the timestamps of these subsequences follow the arrival times of three independent Poisson processes. Since the cut is sampled from $(y_{(j-1)}, y_{k+1}]$ with probability $p = \int_{y_{(j-1)}}^{y_{k+1}} f(t)\mathrm{d}t$, the timestamps of all cuts in $(y_{(j-1)}, y_{k+1}]$ follows the arrival times of a Poisson process with rate

$$\lambda = \lambda_0 \cdot \Pr\{\vartheta \in (y_{(j-1)}, y_{k+1}]\} = \lambda_0 \cdot \int_{y_{(j-1)}}^{y_{k+1}} f(t)\mathrm{d}t.$$

Suppose there exists $1 \le j \le k$ such that $y_{(j-1)} < y_{k+1} < y_{(j)}$. Since the earliest cut (ϑ_t, ρ_t) that separates $y_{(j)}$ does not separate y_{k+1} in the static algorithm, the first cut in the interval $(y_{(j-1)}, y_{k+1}]$ must arrive after ρ_t. The time of the first arrival of this subsequence follows an exponential distribution with rate λ. Due to the memoryless property of the exponential distribution, the first arrival of cuts in $(y_{(j-1)}, y_{k+1}]$ follows $\rho_t + z$, where $z \sim \exp(\lambda)$. Suppose $y_{k+1} > y_{(k)}$. Then, the time of the first arrival in this subsequence is $z \sim \exp(\lambda)$. Therefore, in the static algorithm, the first cut in $(y_{(j-1)}, y_{k+1}]$ has the exact same distribution as the new cut sampled in the data structure. If $y_{(j-1)} \ne m$, then the first cut in $(m, y_{(j-1)}]$ is the same in the data structure and the static algorithm. Combining two parts, the earliest cut that separates y_{k+1} returned by the data structure has the same distribution as that returned by the static algorithm. □

We now analyze the recourse and the update time of the dynamic algorithm with the above implementation.

Proof of Lemma B.3. We now analyze the recourse and the update time of the dynamic algorithm.

Recourse: Let $\mathcal{R}(t)$ be the recourse incurred by request t. We partition the requests into two sets: Let $S_1 \subseteq [k]$ be the set of requests for which the REBUILD operation is not called during the update due to this request. Let $S_2 = [k] \setminus S_1$ be the remaining requests where the REBUILD operation is called. We analyze each case separately.

Case 1 ($t \in S_1$): In this case, the request does not trigger a REBUILD operation, and the recourse is at most $\mathcal{R}(t) \leq 2$. This is because if the request is an insertion, at most two nodes are added to $\mathcal{T}_{t-1}$; if it is a deletion, at most two nodes are removed, i.e., the leaf that contains the center c and its parent in both cases. As a result, the total recourse over all such requests is bounded by

$$\sum_{t \in S_1} \mathcal{R}(t) \leq 2|S_1| \leq 2k. \tag{2}$$

Case 2 ($t \in S_2$): The REBUILD will only be called on one node u_t for each request t. Let C_{u_t} be the set of centers contained in the node u_t of $\mathcal{T}_{t-1}$, and let $k' = |C_{u_t}|$. Since REBUILD(u_t) is called, all $2k' - 1$ nodes in the subtree rooted at u_t are removed from $\mathcal{T}_{t-1}$. If the request t is an insertion of a center c, a new threshold tree is constructed at u_t using the updated center set $C_{u_t} \cup \{c\}$, which has size $k' + 1$. This results in inserting $2(k' + 1) - 1 = 2k' + 1$ nodes back into the tree. Therefore, the recourse is $\mathcal{R}(t) = 2k' - 1 + 2k' + 1 = 4k'$. If the request t is a deletion of a center c, the updated center set is $C_{u_t} \setminus c_t$ of size $k' - 1$, and the rebuilt threshold tree contains $2(k' - 1) - 1 = 2k' - 3$ nodes. The recourse in this case is $\mathcal{R}(t) = (2k' - 1) + (2k' - 3) = 4k' - 4$. In either case, we have the bound $\mathcal{R}(t) \leq 4k'$.

We now analyze the total recourse for S_2. Each node u on which the algorithm called a REBUILD stores an update counter Cnt_u. This update counter is initialized to zero when the node is rebuilt and is incremented by one each time an update (insertion or deletion) involves node u. This node u also stores the number of centers k_u in this node when it is rebuilt. Since the dynamic algorithm rebuilds this node u after $k_u/4$ updates, we have $k' \leq k_u + k_u/4$. Therefore, we have $\mathrm{Cnt}_{u_t} = k_{u_t}/4 \geq k'/5$. Hence, we have

$$\sum_{t \in S_2} \mathcal{R}(t) \leq \sum_{t \in S_2} 20 \cdot \mathrm{Cnt}(u_t). \tag{3}$$

The right-hand side of (3) is bounded by the total number of times any node's counter is incremented. According to the analysis in Lemma B.1, the dynamic algorithm guarantees that after the partition leaf call of a node u, each leaf has at most a $3/4$ fraction of the centers contained in u. Therefore, each update request is involved in at most $O(\log k)$ calls of INSERT_CENTER or DELETE_CENTER. Thus, the total number of times any node's counter is incremented is bounded by $O(k \log k)$. Combining this with (2) and (3), we conclude that $\sum_{t=1}^{k} \mathcal{R}(t) = O(k \log k)$ and thus the amortized recourse is $O(\log k)$.

Update Time: As in the amortized recourse analysis, let $S_1 \subseteq [k]$ be the set of time steps where REBUILD is called on some node u_t, and let $S_2 = [k] \setminus S_1$. We now split the analysis into two cases, depending on whether or not a rebuild is triggered.

Case 1 ($t \in S_1$): Suppose the request t is an insertion of center c_t. Let $u_1, u_2, \ldots, u_l$ be the nodes for which INSERT_CENTER(c_t, u_j) is called. Each such call on node u takes $O(d \log k)$ time. It takes

- $O(d \log k)$ time to update the d self-balancing binary search trees stored in u;
- $O(d \log k)$ time to compute the earliest cut through GET_EARLIEST_CUT(c);
- $O(\log k)$ time to locate this earliest cut and insert the center by searching the self-balancing binary search tree that maintains all cuts $(\omega'_1, \rho'_1), (\omega'_2, \rho'_2), \ldots, (\omega'_r, \rho'_r)$ currently used in the partition leaf call of u.

Since the center c_t can be involved in $O(\log k)$ INSERT_CENTER calls, the update time for an insertion request $t \in S_1$ is $\mathrm{Time}(t) = O(d \log^2 k)$. The same asymptotic bound holds for deletions, as finding the leaf that contains deletion center c_t takes $O(d \log^2 k)$, and the removal takes constant time. Thus, we have

$$\sum_{t \in S_1} \mathrm{Time}(t) = O(|S_1| \cdot d \log^2 k). \tag{4}$$

Case 2 ($t \in S_2$): Let u_t be the node that is rebuilt at request t. As in Case 1, the time to process the request before the rebuild is $O(d \log^2 k)$. If u_t contains k' centers at this request t, then REBUILD(u_t) takes $O(k' d \log^2 k)$ time.

Since when $\textsc{Rebuild}(u_t)$ is triggered, we have the update counter $\mathrm{Cnt}_{u_t} \geq k'/5$. Thus, we charge the rebuild time to the update counter. That is the update time $\mathrm{Time}(t) \leq O(\mathrm{Cnt}_{u_t} \cdot d \log^2 k)$. Therefore, we have

$$\sum_{t \in S_2} \mathrm{Time}(t) \leq O(d \log^2 k) \cdot \sum_{t \in S_2} \mathrm{Cnt}_{u_t}. \tag{5}$$

By the analysis in recourse, we have $\sum_{t \in S_2} \mathrm{Cnt}_{u_t} \leq O(k \log k)$. Combining (4) and (5), we obtain that the total update time is at most

$$\sum_{t=1}^{k} \mathrm{Time}(t) = O(kd \log^3 k)$$

and so the amortized update time is $O(d \log^3 k)$. □

We now prove the main theorem of the dynamic algorithm.

Proof of Theorem 4.1. By Corollary B.2 and Lemma B.3, we get the approximation guarantee, amortized recourse, and the amortized update time of the dynamic algorithm. □