A $(1+\epsilon)$-Approximation for Ultrametric Embedding in Subquadratic Time

Gabriel Bathie[1, 2], **Guillaume Lagarde**[1]

[1] LaBRI, Université de Bordeaux, France
[2] DI ENS, PSL Research University, Paris, France

Abstract

Efficiently computing accurate representations of high-dimensional data is essential for data analysis and unsupervised learning. Dendrograms, also known as ultrametrics, are widely used representations that preserve hierarchical relationships within the data. However, popular methods for computing them, such as *linkage* algorithms, suffer from quadratic time and space complexity, making them impractical for large datasets. The "best ultrametric embedding" (a.k.a. "best ultrametric fit") problem, which aims to find the ultrametric that best preserves the distances between points in the original data, is known to require at least quadratic time for an exact solution. Recent work has focused on improving scalability by approximating optimal solutions in subquadratic time, resulting in a $(\sqrt{2}+\epsilon)$-approximation (Cohen-Addad, de Joannis de Verclos and Lagarde, 2021).

In this paper, we present the first subquadratic algorithm that achieves arbitrarily precise approximations of the optimal ultrametric embedding. Specifically, we provide an algorithm that, for any $c \geq 1$, outputs a c-approximation of the best ultrametric in time $\tilde{O}(n^{1+1/c})$. In particular, for any fixed $\epsilon > 0$, the algorithm computes a $(1+\epsilon)$-approximation in time $\tilde{O}(n^{2-\epsilon+o(\epsilon^2)})$.

Experimental results show that our algorithm improves upon previous methods in terms of approximation quality while maintaining comparable running times.

1 Introduction

Clustering is a fundamental technique in data analysis that is used to group similar data points. It helps in uncovering underlying patterns, segmenting data into meaningful categories, and simplifying data representation. Applications of clustering span various domains, including for example bioinformatics, market segmentation, social network analysis, image processing, feature learning, spatial and geoscience, and many others; we refer to (Murtagh and Contreras 2017) for a comprehensive list of references.

While 'flat' clustering methods, such as k-means, can effectively partition data into distinct groups, they often fall short when dealing with complex data structures, in particular data points that exhibit multiscale structures. These methods typically assume a fixed number of clusters or rely on specific distance thresholds, which may not capture the true nature of the data. Consequently, they can struggle to reveal the intricate relationships between data points.

Hierarchical clustering addresses these limitations by building a multi-level hierarchy of clusters. This approach does not require specifying the number of clusters a priori and can reveal the nested structure of the data at all levels of granularity. Hierarchical clustering iteratively splits or merges data points based on similarity, forming a tree-like representation known as a dendrogram. This tree structure provides a comprehensive view of the data's organization, allowing for more nuanced interpretations and insights.

Dendrograms and hierarchical clusterings are formalized and quantified using the mathematical concept of *ultrametric*[1]. Ultrametrics provides a rigorous foundation for hierarchical clustering and allows the development and analysis of efficient algorithms to compute and analyze the hierarchical structure of data, see for example (Jain and Dubes 1988) or (Carlsson and Mémoli 2010).

The most popular methods for constructing an ultrametric are the *agglomerative algorithms*, such as single linkage, complete linkage, average linkage, and Ward's method. These algorithms work *bottom-up* and build an ultrametric by iteratively merging the closest clusters based on a distance metric. While widely used and successful in many applications, they suffer from two major limitations: first, it is not clear what objective functions these methods aim to optimize, making them difficult to analyze and lacking approximation guarantees; second, they have at least quadratic time and space complexity, making them impractical for handling large datasets.

In this paper, we consider the problem of computing the best ultrametric fit (BUF_∞), which is quantified by how well an ultrametric preserves the distances of the original metric using the notion of *maximum distortion*. Formally, given a set X of points in Euclidean space, our goal is to find an ultrametric Δ such that for all $x, y \in X$, $\|x-y\|_2 \leq \Delta(x,y) \leq c \cdot \|x-y\|_2$, where c is as small as possible. This problem was first introduced by Farach, Kannan, and Warnow (1995), who proved that it cannot be solved in sub-

[1] An ultrametric is a metric that satisfies a stronger triangle inequality: for any three points, the distance between any two points is at most the maximum of the distances between the other two pairs.

quadratic time and provided an exact algorithm matching this lower bound.

To overcome this quadratic time barrier, Cohen-Addad, Karthik C. S., and Lagarde (2020) and Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) initiated the development of faster algorithms for computing *approximations* of the optimal ultrametric. They proposed respectively algorithms achieving a $5c$-approximation and a $\sqrt{2}c$-approximation in time $\tilde{O}(n^{1+12/c^2})$, thus providing subquadratic algorithms to approximate BUF_∞ up to factors of ≈ 17.32 and ≈ 4.90, respectively.

1.1 Our Contribution

The fundamental question explored in this paper is whether **we can achieve arbitrarily precise approximations of the optimal ultrametric fit in subquadratic time**, or if there exists a theoretical barrier preventing this.

We answer this question positively by constructing the first subquadratic algorithm that achieves arbitrarily precise approximations of the optimal solution to BUF_∞. More precisely, we prove the following theorem:

Theorem 1. *For any $\gamma \geq 1$ and $\alpha > 1$, there exists an algorithm that computes a $\gamma \cdot \alpha$-approximation of BUF_∞ that runs in time $\tilde{O}(n^{1+1/\gamma^2} + n^{1+1/\alpha^2})$ and space $\tilde{O}(n^{1+1/\gamma^2} + n^{1+1/\alpha^2})$.*

Previously, the best known subquadratic time algorithm for BUF_∞, by Cohen-Addad, de Joannis de Verclos, and Lagarde (2021), could only handle approximation factors greater than $\sqrt{2} \cdot \sqrt{12} \approx 4.90$, while we can now achieve arbitrarily precise approximations. Our algorithm is based on two main components:

- An algorithm[2] for computing a γ-Kruskal Tree (abbreviated γ-KT) in time $\tilde{\mathcal{O}}(n^{1+1/\gamma^2})$. This algorithm might be of independent interest. Unlike previous algorithms for γ-KT, it does not require the construction of a γ-spanner, which cannot be built in subquadratic time for $\gamma < \sqrt{2}$ (Andoni and Zhang 2023).
- A new data structure that computes an α-approximation of the so-called cut weights in time $\tilde{\mathcal{O}}(n^{1+1/\alpha^2})$ (See Definition 3 for the definition of the cut weights). The previous best subquadratic time algorithm was only able to handle approximation factors greater than $\sqrt{2}$, see (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021). Our data structure is based on a dynamic version of the approximate farthest neighbor data structure developed by Pagh et al. (2017).

1.2 Experimental Results

To complement our theoretical results and to demonstrate the practical efficiency of our algorithm, we perform an extensive set of experiments. We measure the performance of our algorithm both in terms of approximation factor and running time on five classical and diverse real-world datasets, and evaluate its scalability on large synthetic datasets. We compare our algorithm with the state-of-the-art algorithm of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) and the widely used implementation of the `fastcluster` Python package. The results show that our algorithm yields better approximations than the algorithm of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) while maintaining a comparable running time and that it can scale to datasets containing millions of points.

1.3 Related Work

Carlsson and Mémoli (2010) established important foundations for hierarchical clustering, in particular through an in-depth study of ultrametrics that revealed the theoretical properties of hierarchical clustering algorithms.

Other works have explored the complexity of optimizing other distortion measures, such as the average distortion (ℓ_1 norm) and more generally, the ℓ_p norm for different values of p. The problem is NP-complete for $p = 1, 2$ and APX-hard for $p = 1$ (see (Wareham 1993) and (Agarwala et al. 1998)). Ailon and Charikar (2011) investigated the case of the ℓ_p norm for various values of p and provided polynomial-time algorithms to $O((\log n \log\log n)^{1/p})$-approximate both the best ultrametric and tree metric embeddings using an LP formulation and rounding techniques. These studies consider problems that are NP-hard (at least for $p = 1, 2$) and provide approximation algorithms, but do not focus on scalability: their algorithms have a complexity of $\Omega(n^4)$.

Subquadratic time algorithms in high-dimensional settings, such as those in our work, have been studied by (Gilpin, Qian, and Davidson 2013) and (Cochez and Mou 2015), who provide near-linear time algorithms in the best running case. However, these algorithms lack approximation guarantees for their outputs.

Another line of research focuses on different objective functions to quantify the quality of hierarchical clustering. For instance, Dasgupta (2016) introduced an objective function based on cluster properties. Since then, numerous efforts have been devoted to developing algorithms that optimize or approximate this and related metrics, see e.g. (Roy and Pokutta 2017), (Charikar and Chatziafratis 2017) and (Cohen-Addad, Kanade, and Mallmann-Trenn 2017).

Significant work has also been done to understand the guarantees provided by popular algorithms. Recently, (Cohen-Addad et al. 2019) and (Moseley and Wang 2023) proved that average linkage has a small constant approximation ratio for the dual of Dasgupta's objective function, while other methods, such as the bisecting k-means top-down approach, perform poorly for the same objective.

1.4 Organization of the Paper

First, we introduce the necessary definitions and notations in Section 2. In Section 3 we present the high-level algorithm of (Cohen-Addad, Karthik C. S., and Lagarde 2020) and (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021), on which our work is based, and highlight our improve-

[2] An algorithm was proposed in (Cohen-Addad, Karthik C. S., and Lagarde 2020) and (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021), but it operates in subquadratic time only when $\gamma \geq \sqrt{12}$

ments. Next, we describe in detail our algorithms to compute a γ-KT and an α-approximation of the cut weights for any $\gamma \geq 1, \alpha \geq 1$ in Section 4 and Section 5 respectively. Finally, we discuss our experimental results in detail in Section 6.

2 Preliminaries

Throughout this work, the $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in the input size.

For any dimension $d \in \mathbb{N}$, we denote by $\ell_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ the Euclidean distance between two points $x, y \in \mathbb{R}^d$. Given a set X, an ultrametric distance Δ on X is a distance function $\Delta : X \times X \to \mathbb{R}^+$ where the triangle inequality is replaced by the stronger ultrametric inequality:

$$\forall x, y, z \in X, \Delta(x, z) \leq \max(\Delta(x, y), \Delta(y, z)).$$

In this case, (X, Δ) is an ultrametric space. When X is a finite set, an ultrametric can be represented by a tree T together with a weight function $w : T \to \mathbb{R}^+$, such that

- each element of X is assigned to a leaf of T,
- the weight of each leaf is 0,
- the weights are decreasing along any path from the root to a leaf.

The ultrametric distance induced by T and w is denoted by Δ_T and is defined as $\Delta_T(u, v) = w(\text{LCA}(u, v))$, where $\text{LCA}(u, v)$ represents the least common ancestor of u and v in T. Using a tree representation of an ultrametric is useful for visualization and interpretation (see the full version of the paper for more details).

Distortion ratio. To quantify how well an ultrametric Δ preserves the distances of a metric ℓ, we use the standard concept of *distortion*. Given a metric space (X, ℓ) and an ultrametric Δ on X such that $\ell(x, y) \leq \Delta(x, y)$ for all $(x, y) \in X^2$, the distortion of Δ, denoted by DIST_∞,is the maximum ratio between the Euclidean distance and the ultrametric distance, i.e.,

$$\text{DIST}_\infty = \max_{(x,y) \in X^2, x \neq y} \frac{\Delta(x, y)}{\ell(x, y)}.$$

Best ultrametric fit. As defined by Farach, Kannan, and Warnow (1995), the best ultrametric fit problem (BUF_∞) consists in finding, given a metric space (X, ℓ), an ultrametric Δ on X that preserves the distances as well as possible. Formally, the problem is to find an ultrametric Δ such that:

- $\ell(x, y) \leq \Delta(x, y)$ for all $(x, y) \in X^2$,
- the distortion ratio $\text{DIST}_\infty(\ell, \Delta)$ is minimized.

We denote by DIST^*_∞ the (unique) distortion ratio of an optimal solution.

Δ is a *c-approximation* of the best ultrametric fit if the distortion of Δ is at most c times the distortion of the optimal ultrametric, or equivalently if for any pair of points $x, y \in X$, we have:

$$\ell(x, y) \leq \Delta(x, y) \leq c \cdot \text{DIST}^*_\infty \cdot \ell(x, y).$$

The scalar c is called the approximation factor of Δ. By extension, an algorithm is a c-approximation algorithm of BUF_∞ if it outputs an ultrametric which is a c-approximation of the optimal ultrametric embedding.

Working in Euclidean spaces. From now on, and as in (Cohen-Addad, Karthik C. S., and Lagarde 2020) and (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021), we consider the case where X is a set of n points in a Euclidean space $\mathbb{R}^d$ equipped with the Euclidean metric ℓ_2, which is one of the most natural settings for many applications in data analysis and unsupervised learning. Note that by the Johnson-Lindenstrauss lemma (Johnson, Lindenstrauss, and Schechtman 1986), the dimension d can always be reduced to $O(\log n)$ while preserving the distances between pairs of points up to a multiplicative factor of $(1 + \epsilon)$ for any fixed $\epsilon > 0$.

Omitted Proofs. To comply with page limit requirements, most of the proofs of the technical statements in this work, as well as the results of some experiments, have been omitted and can be found in the full version of the paper, which will be available online.

3 High Level Algorithm From (Cohen-Addad, Karthik C. S., and Lagarde 2020)

We build our approximation algorithm using the framework of Cohen-Addad, Karthik C. S., and Lagarde (2020), who provide a way to compute a $5 \cdot \gamma$-approximation of BUF_∞ in time $O(nd + n^{1+12/\gamma^2})$. This result was later improved by Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) who provide, for any fixed $\epsilon > 0$, a $(\sqrt{2}+\epsilon)\cdot\gamma$-approximation for the same asymptotic running time. We briefly recall the main ideas of their approach and pinpoint where we improve upon it. We first need a few more definitions.

Definition 2. *Let $G = (V, E, w)$ be a weighted graph and let $\gamma \geq 1$. A spanning tree $T = (V, E_T)$ of G is a γ-Kruskal tree (or γ-KT for short) of G if for every edge $e \in E \setminus E_T$, we have*

$$w(e) \geq \frac{1}{\gamma} \max_{e' \in P_T(e)} w(e'),$$

where $P_T(e)$ is the unique path in T from one endpoint of e to the other.

Let T be a spanning tree of the complete graph induced by the set of points X. For an edge $e = (x, y) \in T$, we denote by $L(e)$ (respectively $R(e)$) the connected component of $T \setminus \{e' \in T \mid \ell_2(e') \leq \ell_2(e)\}$ that contains x (respectively y).

Definition 3. *The* cut weight $\text{CW}(e)$ *of an edge e is defined as the maximal distance between a point in $L(e)$ and a point in $R(e)$:*

$$\text{CW}(e) = \max_{x \in L(e), y \in R(e)} \ell_2(x, y).$$

The function $\text{CW} : e \mapsto \text{CW}(e)$ for $e \in E_T$ is referred to as the *cut weights of T*. We say that ACW is an α-approximation of the cut weights of T if for every edge e of T, we have

$$\text{CW}(e) \leq \text{ACW}(e) \leq \alpha \cdot \text{CW}(e).$$

Given a spanning tree T and a edge-weight function $w : E_T \to \mathbb{R}^+$, the *Cartesian tree* of T with respect to w is a weighted binary tree defined inductively as follows:

- If T is a single node, then the Cartesian tree is a single node with weight 0.
- Otherwise, the root of the Cartesian tree corresponds to the edge e of T with the largest weight, and the left and right children are the Cartesian trees of the two connected components of $T \setminus \{e\}$.

Cohen-Addad, Karthik C. S., and Lagarde (2020) provide an high-level algorithm that achieves a $\gamma \cdot \alpha$-approximation of BUF_∞ by computing a γ-KT and its α-approximate cut weights. This algorithm is inspired by the one presented by Farach, Kannan, and Warnow (1995), who provide a quadratic time algorithm for computing the best ultrametric embedding of the data. The main difference is that in (Cohen-Addad, Karthik C. S., and Lagarde 2020), the first two steps are approximated rather than computed exactly. We outline this algorithm in Algorithm 1.

Algorithm 1: $\gamma \cdot \alpha$-approx. of the best ultrametric fit

Input: $X \subseteq \mathbb{R}^d$: set of points
Output: A $\gamma \cdot \alpha$-approximation of BUF_∞
1 $T \leftarrow \gamma$-KT T of the complete graph induced by X
2 ACW $\leftarrow \alpha$-approximation of the cut weights of T
3 $C_T \leftarrow$ Cartesian tree of T w.r.t. ACW
4 **return** *the ultrametric induced by* C_T

Theorem 4 ((Cohen-Addad, Karthik C. S., and Lagarde 2020, Theorem 3.1))**.** *For any $\gamma \geq 1$ and $\alpha \geq 1$, the output of Algorithm 1 is a $\gamma \cdot \alpha$-approximation of the best ultrametric fit.*

Furthermore step 3 of Algorithm 1 can be easily computed in $O(n \log n)$ time using a disjoint-set data structure (Farach, Kannan, and Warnow 1995). Hence, if there are algorithms that compute a γ-KT in time $f(n, d)$, and an α-approximation of the cut weights in time $g(n, d)$, then there is an algorithm that computes a $\gamma \cdot \alpha$-approximation of the best ultrametric fit in time $O(n \log n) + f(n, d) + g(n, d)$. In this work, we show that there are algorithms with $f(n, d) = \tilde{O}(nd + n^{1+1/\gamma^2})$ and $g(n, d) = \tilde{O}(nd + n^{1+1/\alpha^2})$. As mentioned in the preliminary section, we can assume w.l.o.g. $d = O(\log n)$, hence this term is asymptotically dominated by the other in both cases, and we drop it from now on.

Complexity of computing a γ-KT. Cohen-Addad, Karthik C. S., and Lagarde (2020) present an algorithm for computing a γ-KT in time $\tilde{O}\left(n^{1+O(1/\gamma^2)}\right)$. This is achieved by constructing a sparse γ-spanner using the LSH-based algorithm of Har-Peled, Indyk, and Sidiropoulos (2013) and then computing a minimum spanning tree of the γ-spanner. However, as explained by (Andoni and Zhang 2023), the $O(1/\gamma^2)$ term in the exponent contains a factor of 12, so this algorithm is only faster than the exact quadratic algorithm for $\gamma > \sqrt{12}$. Furthermore, Andoni and Zhang (2023) prove that γ-spanners cannot be constructed in subquadratic time for $\gamma < \sqrt{2}$, thus making spanner-based approaches inefficient for building γ-KTs when $\gamma < \sqrt{2}$. We address this limitation by introducing a new algorithm that computes a γ-KT in time $\tilde{O}\left(n^{1+1/\gamma^2}\right)$ without relying on spanners.

Complexity of approximating the cut weights. The best algorithm for approximating cut weights is by Cohen-Addad, de Joannis de Verclos, and Lagarde (2021), who provide an algorithm that computes a $(\sqrt{2} + \epsilon)$-approximation of the cut weights in time $O\left(nd \cdot \mathrm{poly}(\log n, 1/\varepsilon)\right)$.

The total asymptotic running time to compute the ultrametric is $\tilde{O}(nd + n^{1+12/\gamma^2})$, dominated by the time complexity of the algorithm that computes a γ-KT (or more precisely, a γ-spanner in these algorithms). This observation indicates that there is room to improve the approximation factor of the cut weights without increasing the total running time of the algorithm. However, the method presented in (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021) cannot leverage this observation due to its inherent (geometric) bottleneck, with an approximation factor of $\sqrt{2}$ for computing the cut weights.

Our Contributions The main contributions of this paper are as follows: first, we restore the claim of Cohen-Addad, Karthik C. S., and Lagarde (2020) by providing an algorithm that, for any $\gamma \geq 1$, computes a γ-KT in time $\tilde{O}(n^{1+1/\gamma^2} \log \delta)$ (Theorem 5), where δ is the *spread* of the input space X, defined as the ratio between the diameter and the minimum pairwise distance of X. Second, we present an algorithm that computes an α-approximation of the cut weights in time $\tilde{O}(n^{1+1/\alpha^2})$ for any $\alpha \geq 1$ (Theorem 6).

Theorem 5. *For any $\gamma > 1$, there is an algorithm that computes a γ-KT of a given n-points Euclidean space X in time and space $\tilde{O}(n^{1+1/\gamma^2} \log \delta)$, where δ is the spread of X.*

Theorem 6. *For any $\alpha > 1$, there is an algorithm that computes a α-approximation of the cut weights of an n-nodes tree T in time and space $\tilde{O}(n^{1+1/\alpha^2})$.*

Finally, these two results can be combined to obtain a c-approximation of BUF_∞, as shown in the following corollary.

Corollary 7. *For any $c \geq 1$, there exists an algorithm that computes a c-approximation of BUF_∞ in time $\tilde{O}(n^{1+1/c})$.*

Proof. Take $\gamma = \sqrt{c}$ and $\alpha = \sqrt{c}$ in Theorem 5 and Theorem 6, respectively. By using Theorem 4, we obtain a c-approximation of BUF_∞, and the total running time is $\tilde{O}(n^{1+1/c})$. □

For $c = (1 + \epsilon)$, this gives a $(1 + \epsilon)$-approximation of BUF_∞ in time $\tilde{O}(n^{2-\epsilon+o(\epsilon^2)})$, which remains subquadratic in n. For comparison, when $c = 1$ (no approximation), it is known that the best ultrametric fit problem cannot be solved in subquadratic time, see for example (Farach, Kannan, and Warnow 1995).

High-level overview of the techniques.

- **Theorem 5:** We achieve this result using a new method that avoids γ-spanner constructions. The algorithm operates essentially through a breadth-first traversal of the graph, guided by locality-sensitive hashing of the data.

- **Theorem 6:** This algorithm is built on a dynamic version of the approximate farthest neighbor data structure developed by Pagh et al. (2017), which supports queries in time $\tilde{O}(n^{1/\alpha^2})$ and space $\tilde{O}(n^{1+1/\alpha^2})$. We extend this data structure to work over a partition of the input space X, allowing approximate farthest neighbor queries within *clusters* (i.e., subsets of the partition) and efficient merging of clusters.

We now proceed to the details of these two algorithms.

4 Algorithm for γ-Kruskal tree

In this section, we present the algorithm to compute a γ-KT, as stated in Theorem 5, along with its full analysis.

Recall that the algorithm by Cohen-Addad, Karthik C. S., and Lagarde (2020) for computing a γ-KT operates in two steps: first, it computes a sparse γ-spanner of the complete graph induced by the set of points X, and then it computes a minimum spanning tree of the γ-spanner. To find a γ-spanner, they use the algorithm of Har-Peled, Indyk, and Sidiropoulos (2013), which runs in time $\tilde{\mathcal{O}}(n^{1+12/\gamma^2})$ and uses Locality-Sensitive Hashing. This algorithm is subquadratic only when $\gamma < \sqrt{12}$. While it might be possible to reduce this constant, Andoni and Zhang (2023) showed that spanners cannot be constructed in subquadratic time for $\gamma < \sqrt{2}$.

Instead of relying on spanners, we propose an algorithm that builds a γ-KT for any $\gamma \geq 1$ in time $\tilde{O}(n^{1+1/\gamma^2} \log \delta)$, essentially via a breadth-first traversal guided by LSH.

We first recall the definition of Locality-Sensitive Hash functions (LSH):

Definition 8. *Let (X, ℓ) be a metric space over n points, and let $c > 1$ and $R > 0$. A family $\mathcal{H}$ of functions is a (ρ, c, R)-LSH if, when choosing a function h uniformly at random from $\mathcal{H}$ we have, for every $x, y \in X$:*

- $\ell(x, y) \leq R \Rightarrow \mathbb{P}[h(x) = h(y)] \geq 1/n^{\rho}$
- $\ell(x, y) \geq cR \Rightarrow \mathbb{P}[h(x) = h(y)] \leq 1/n$

Andoni and Indyk (2006) showed that when X is a Euclidean space, i.e. ℓ is the ℓ_2 metric, then for every $c > 1$ and every $R > 0$, there exists a $(1/c^2, c, R)$-LSH family of functions $\mathcal{H}_{c,R}$, and the evaluation of a random function from $\mathcal{H}_{c,R}$ on all n points of X takes $\tilde{O}(n)$ time.

Given an LSH function h, the *buckets* of h are the equivalence classes of the relation $x \sim y \Leftrightarrow h(x) = h(y)$. Intuitively, points with the same hash value are put into the same bucket.

Finally, we introduce an important property, denoted (*), that will be useful in explaining the behavior of our algorithm.

Definition 9. *We say that a set E of edges satisfies the property (*) if for any pair $(u, v) \in X^2$, there is a path from u to v in E using only edges of weight at most $\gamma \cdot \ell_2(u, v)$.*

Overview of the algorithm. The algorithm works in two steps:

- First we compute a set E of $\tilde{O}(n^{1+1/\gamma^2} \cdot \log \delta)$ edges that satisfies (*). This is the purpose of Proposition 10.
- Then, we compute a minimum spanning tree of E. Lemma 11 shows that this tree is a γ-KT. This part is done in time $O(|E| \log |E|) = \tilde{O}(n^{1+1/\gamma^2})$ using the standard algorithm of Kruskal (1956).

The above algorithm, combined with the following proposition and lemma, yields the main result of this section, a proof of Theorem 5.

Proposition 10. *There is an algorithm that computes a set E of $\tilde{O}(n^{1+1/\gamma^2} \cdot \log \delta)$ edges that satisfies (*) in time $\tilde{O}(n^{1+1/\gamma^2} \log \delta)$.*

Lemma 11. *Let X be an n-points Euclidean space and let $\gamma > 1$. Let $E \subseteq X^2$ be a set of edges that satisfies (*). then running Kruskal's algorithm on E' yields a γ-KT of X in time $O(|E| \log |E|)$.*

5 Better Cut-Weights via Approximate Farthest Neighbors

The second step of Algorithm 1 is to compute the cut weights of the spanning tree obtained in the first step. Farach, Kannan, and Warnow (1995) provided a quadratic-time algorithm to compute the exact cut weights. More recently, (Cohen-Addad, Karthik C. S., and Lagarde 2020) and (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021) proposed a 5- and a $\sqrt{2}$-approximation algorithm that both operate in quasilinear time. However, their approximations are inherently limited due to their reliance on specific geometric properties of Euclidean spaces.

In this section, we introduce an α-approximation algorithm for the cut weights that works for any $\alpha > 1$. Our result is as follows:

Theorem 6. *For any $\alpha > 1$, there is an algorithm that computes a α-approximation of the cut weights of an n-nodes tree T in time and space $\tilde{O}(n^{1+1/\alpha^2})$.*

The core ingredient of our algorithm is a *dynamic* version of the data structure for approximate farthest neighbor of Pagh et al. (2017), which we present in the next subsection. The algorithm behind the Theorem 6 is described in the second part of this section.

5.1 Dynamic Approximate Farthest Neighbor

In order to obtain an α-approximation of the cut weights, we use the data structure of Pagh et al. (2017) for *approximate farthest neighbors* (AFN) in Euclidean spaces. Their data structure preprocesses a subset of a metric space, and can then find in that subset an α-approximate farthest neighbor of a given query point in time $\tilde{O}(n^{1/\alpha^2})$.

Definition 12 (AFN)**.** *Let (X, d) be a metric space and let $\alpha > 1$. A data structure $\mathcal{D}$ solves the α-AFN problem over a set $S \subseteq X$ if, given a point $q \in X$, it returns a point r that is an α-approximate farthest neighbor of q in S, i.e. it holds that*

$$d(q, r) \geq \frac{1}{\alpha} \max_{p \in S} d(q, p).$$

To fit our use case, we show that one can extend the data structure of Pagh et al. (2017) to be *dynamic*, i.e. given the

data structure for two disjoint subsets S, S' of (X, d), we can construct a data structure for $S \sqcup S'$ faster than the time needed to build it from scratch.

Theorem 13. *There is a data structure for α-AFN over a dynamic partition of a metric space (X, d) of n points that supports the following operations:*

- INITIALIZE(X, α)*: create an data structure containing a cluster $S_x = \{x\}$ for each x in X, in time $\tilde{O}(n^{1+1/\alpha^2})$.*
- QUERY$(\mathcal{D}, S, q)$*: given a cluster S in $\mathcal{D}$ and a query point $q \in X$, return an α-approximate farthest neighbor of q in S in time $\tilde{O}(n^{1/\alpha^2})$.*
- MERGE$(\mathcal{D}, S, S')$*: given two clusters S, S' in $\mathcal{D}$, add the cluster $S'' = S \sqcup S'$ to $\mathcal{D}$ in time $\tilde{O}(n^{1/\alpha^2} \cdot \min(|S|, |S'|, n^{1/\alpha^2}))$. This operation consumes the clusters S and S', i.e. they cannot be used in other operations afterward.*

This data structure uses $\tilde{O}(n^{1+1/\alpha^2})$ space. Furthermore, this construction is probabilistic (in the INITIALIZE *function), and for every q and S, the* QUERY *operation fails with probability at most $1/n^3$.*

5.2 Approximate Cut-weight Algorithm

We give a proof of Theorem 6, i.e. we give an algorithm that computes an α-approximation of cut weights of a tree in time and space $\tilde{O}(n^{1+1/\alpha^2})$, using the data structure of Theorem 13.

By augmenting the aforementioned data structure with a disjoint-set data structure, we can additionally support the following operations:

- FIND$(\mathcal{D}, q)$: return the (unique) cluster S in $\mathcal{D}$ that contains q, in time $O(\log^* n)$.
- ENUMERATE$(\mathcal{D}, S)$: iterate over all elements of a cluster S in $\mathcal{D}$, in total time $O(|S|)$.

The procedure to compute an α-approximation of the cut weights is given in Algorithm 2.

Algorithm 2: α-approximation of the cut weights

Input: $X \subseteq \mathbb{R}^d$: set of points,
T: spanning tree as a list of weighted edges sorted by non-decreasing weight,
$\alpha > 1$: approximation parameter

```
D ← INITIALIZE(X);
foreach edge e = (x, y) in increasing order of weights do
    S_x ← FIND(D, x); S_y ← FIND(D, y);
    if |S_x| > |S_y| then
        Swap S_x and S_y;
    ACW(e) ← α·max{w(z, QUERY(D, S_y, z)) | z ∈ ENUMERATE(D, S_x)};
    MERGE(D, S_x, S_y);
return ACW
```

We now turn to proving Theorem 6. We first analyze the complexity of Algorithm 2. The space complexity of $\tilde{O}(n^{1+1/\alpha^2})$ follows from that of the data structure $\mathcal{D}$ of Theorem 13. To obtain the desired time complexity, we crucially rely on the fact that, on Line 6 of Algorithm 2, we iterate over the smallest of S_x and S_y and query the other when computing ACW. This allows us to prove using a counting argument that the algorithm makes $O(n \log n)$ calls to QUERY, showing that the algorithm runs in time $\tilde{O}(n^{1+1/\alpha^2})$.

Lemma 14. *Algorithm 2 runs in time $\tilde{O}(n^{1+1/\alpha^2})$.*

We now show that the ACW function is an α-approximation of the cut weights with high probability. Intuitively, putting aside low-probability errors, QUERY$(\mathcal{D}, S_y, q)$ returns a point whose distance to q is between $d_{\max}/\alpha$ and $d_{\max}$, where $d_{\max}$ is the maximum distance between q and a point of S_y. Therefore, by taking the maximum over all points in S_x and multiplying by α, we obtain a value between $\mathrm{CW}(e)$ and $\alpha \cdot \mathrm{CW}(e)$.

Lemma 15. *With high probability, the function* ACW *returned by Algorithm 2 is an α-approximation of the cut weights* CW *of T. More precisely, with probability at least $1 - 1/n$, we have, for every $e \in T$:*

$$\mathrm{CW}(e) \leq \mathrm{ACW}(e) \leq \alpha \cdot \mathrm{CW}(e).$$

6 Experiments

We evaluate the performance of our algorithm on two types of datasets. First, as in (Cohen-Addad, Karthik C. S., and Lagarde 2020) and (Cohen-Addad, de Joannis de Verclos, and Lagarde 2021), we use five classic, diverse, real-world datasets to evaluate both the quality of the approximation and the runtime of the algorithms; see Table 2 for details on these datasets. Second, we use synthetic datasets to evaluate how the runtime of the algorithms scales with larger datasets. In this case, we cannot measure the quality of the approximation given by the algorithms, since this takes quadratic time, which is unreasonably long for large datasets. We compare our algorithm with the state-of-the-art algorithm of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) and the widely used implementation of the `fastcluster` Python package.

The experiments were conducted on identical nodes of the Grid'5000 cluster, running Debian GNU/Linux 5.10.0-28-amd64. The hardware configuration includes an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and 126GB of RAM. For the experiments, our algorithm is implemented in the Rust programming language, version 1.79.0 (`129f3b996`, 2024-06-10). Our code was compiled in `release` mode.

Experiment 1: Accuracy of the BUF_∞ algorithm. We measure the performance of our BUF_∞ c-approximation algorithm using our γ-KT and α-ACW algorithms with $\alpha = \gamma = \sqrt{c}$, which we call FASTULT. Based on the results of Experiment A (see full version of the paper), we modify the α-ACW algorithm to multiply distances by $\sqrt{\alpha}$ instead of α: in practice, this is sufficient to overestimate the cut weights and improves the approximation factor. We evaluate FASTULT for different values of c, and for each value we run the algorithm $t = 30$ times on each of the 5 datasets, for a total of 150 runs per value of c.

Algorithm	MICE apx.	MICE $T\,(s)$	PENDIGITS apx.	PENDIGITS $T\,(s)$	SHUTTLE apx.	SHUTTLE $T\,(s)$
FKW	1	$0.12s$	1	$8.16s$	1	$236.72s$
FastUlt ($c = 4$)	1.88	$0.12s$	1.53	$1.80s$	1.41	$21.39s$
FastUlt ($c = 9$)	3.67	$0.07s$	2.26	$0.79s$	1.85	$7.82s$
FastUlt ($c = 16$)	5.58	$0.04s$	2.79	$0.52s$	2.63	$4.69s$
CVL	3.27	$0.14s$	1.85	$0.76s$	2.41	$10.17s$

Table 1: Comparison of FastUlt with the state-of-the-art algorithm by Cohen-Addad, de Joannis de Verclos, and Lagarde (2021), denoted "CVL" in the table. The "apx." column reports the approximation factor, i.e. the distortion of the output ultrametric normalized by the distortion of the optimal ultrametric, given by quadratic-time algorithm by Farach, Kannan, and Warnow (1995), denoted "FKW". Each reported distortion or running time value is the average of 30 runs of the algorithm, all standard deviations are less than 10% for appoximation and 2% for runtimes.

Dataset	**size**	**dim.**	**optimal dist.**
IRIS	150	4	8.07
DIABETES	768	8	5.96
MICE	1080	77	4.92
PENDIGITS	10992	16	13.86
SHUTTLE	58000	9	29.72

Table 2: Description of the datasets used for evaluation. All datasets are publicly available on the UCI ML Repository (Kelly, Longjohn, and Nottingham 2024b), or Kaggle (Kelly, Longjohn, and Nottingham 2024a) for the DIABETES dataset.

The key takeaway from this experiment is that our algorithm performs significantly better than the worst-case approximation factor c. For example, if one wants a 2-approximation of the best ultrametric embedding of a dataset, they can run the algorithm with a larger parameter, e.g. $c = 9$ instead of 2. This approach will greatly reduce the running time and space usage from $\tilde{\mathcal{O}}(n^{1.5})$ to $\tilde{\mathcal{O}}(n^{1.11})$ while still providing a high-quality, low-distortion embedding.

Experiment 2: Comparison with previous methods. We compare the performance of FastUlt with the previous best known algorithm for BUF_∞ by Cohen-Addad, de Joannis de Verclos, and Lagarde (2021), both in terms of the quality of the approximation factor of the best ultrametric and the running time. The results are given in Table 1. These results show that FastUlt with $c = 4$ can achieve better ultrametric embeddings than the algorithm of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) for a comparable computational cost. Furthermore, by using $c = 9$ or 16, we can obtain embeddings with similar distortion but with a lower running time. Thus, another advantage of FastUlt is the ability to trade off approximation factor and running time by adjusting the parameter c, for any desired embedding quality.

Experiment 3: Scaling. Finally, to evaluate how well our algorithm scales to larger datasets, we create two synthetic datasets with up to one million data points. These datasets contain N uniformly random points from $[0, 1]^d$, for $(N, d) \in [(10^5, 100), (10^6, 10)]$. The running times are given in Table 3.

Algorithm	$N = 10^5$ $d = 100$	$N = 10^6$ $d = 10$
FastUlt ($c = 4$)	1m 39.89s	21m 58.50s
FastUlt ($c = 9$)	34.07s	5m 13.99s
FastUlt ($c = 16$)	19.24s	2m 34.97s
CVL	7.96s	1m 54.48s
Single Linkage	9m 55.36s	$\geq$ 10h

Table 3: Running time of the FastUlt algorithm, the algorithm of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021) and the *single linkage* algorithm from the `fastcluster` python package on datasets of N d-dimensional random points. Each reported running time is an average of 30 runs.

We observe that, while our algorithm is slower than that of Cohen-Addad, de Joannis de Verclos, and Lagarde (2021), FastUlt does not suffer from the same quadratic blow-up as classical algorithms such as single linkage, and maintains a reasonable running time that can be afforded for the analysis of large datasets.

Further, we empirically estimate the constant ρ_c such that our algorithm with parameter c runs in time $\tilde{O}(n^{\rho_c})$. To this end, we measure the running time of our program on random datasets of N points of dimension 20, for $N = 10^4$ to $N = 4 \cdot 10^5$, on 30 independent runs for each N. For $c = 4, 6, 9$, we obtain values of $\rho_c = 1.31, 1.27$ and 1.24 respectively. This confirms that our implementation runs in subquadratic time.

Acknowledgements

This work was partially supported by the SAIF project, funded by the "France 2030" government investment plan managed by the French National Research Agency, under the reference ANR-23-PEIA-0006.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER

and several Universities as well as other organizations (see https://www.grid5000.fr).

References

Agarwala, R.; Bafna, V.; Farach, M.; Paterson, M.; and Thorup, M. 1998. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM Journal on Computing*, 28(3): 1073–1085.

Ailon, N.; and Charikar, M. 2011. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM Journal on Computing*, 40(5): 1275–1291.

Andoni, A.; and Indyk, P. 2006. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 459–468.

Andoni, A.; and Zhang, H. 2023. Sub-quadratic $(1+\epsilon)$-approximate Euclidean Spanners, with Applications. *arXiv preprint arXiv:2310.05315*.

Carlsson, G. E.; and Mémoli, F. 2010. Characterization, Stability and Convergence of Hierarchical Clustering Methods. *J. Mach. Learn. Res.*, 11: 1425–1470.

Charikar, M.; and Chatziafratis, V. 2017. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 841–854. SIAM.

Cochez, M.; and Mou, H. 2015. Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*, 505–517.

Cohen-Addad, V.; de Joannis de Verclos, R.; and Lagarde, G. 2021. Improving Ultrametrics Embeddings Through Coresets. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, 2060–2068. PMLR.

Cohen-Addad, V.; Kanade, V.; and Mallmann-Trenn, F. 2017. Hierarchical clustering beyond the worst-case. *Advances in Neural Information Processing Systems*, 30.

Cohen-Addad, V.; Kanade, V.; Mallmann-Trenn, F.; and Mathieu, C. 2019. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4): 1–42.

Cohen-Addad, V.; Karthik C. S.; and Lagarde, G. 2020. On Efficient Low Distortion Ultrametric Embedding. *CoRR*, abs/2008.06700.

Dasgupta, S. 2016. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 118–127.

Farach, M.; Kannan, S.; and Warnow, T. J. 1995. A Robust Model for Finding Optimal Evolutionary Trees. *Algorithmica*, 13(1/2): 155–179.

Gilpin, S.; Qian, B.; and Davidson, I. 2013. Efficient hierarchical clustering of large high dimensional datasets. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 1371–1380.

Har-Peled, S.; Indyk, P.; and Sidiropoulos, A. 2013. Euclidean spanners in high dimensions. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, 804–809. SIAM.

Jain, A. K.; and Dubes, R. C. 1988. *Algorithms for clustering data*. Prentice-Hall, Inc.

Johnson, W. B.; Lindenstrauss, J.; and Schechtman, G. 1986. Extensions of Lipschitz maps into Banach spaces. *Israel Journal of Mathematics*, 54(2): 129–138.

Kelly, M.; Longjohn, R.; and Nottingham, K. 2024a. Diabetes Dataset - Kaggle. https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database. Accessed: 2024-06-23.

Kelly, M.; Longjohn, R.; and Nottingham, K. 2024b. The UCI machine learning repository. http://archive.ics.uci.edu. Accessed: 2024-06-23.

Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1): 48–50.

Moseley, B.; and Wang, J. R. 2023. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. *Journal of Machine Learning Research*, 24(1): 1–36.

Murtagh, F.; and Contreras, P. 2017. Algorithms for hierarchical clustering: an overview, II. *WIREs Data Mining Knowl. Discov.*, 7(6).

Pagh, R.; Silvestri, F.; Sivertsen, J.; and Skala, M. 2017. Approximate furthest neighbor with application to annulus query. *Information Systems*, 64: 152–162.

Roy, A.; and Pokutta, S. 2017. Hierarchical clustering via spreading metrics. *Journal of Machine Learning Research*, 18(88): 1–35.

Wareham, H. 1993. On the complexity of inferring evolutionary trees. *Technical Report Technical Report*, 9301.