Improving Ultrametrics Embeddings Through Coresets

Vincent Cohen-Addad [*1] **Rémi de Joannis de Verclos** [*2] **Guillaume Lagarde** [*3]

Abstract

To tackle the curse of dimensionality in data analysis and unsupervised learning, it is critical to be able to efficiently compute "simple" faithful representations of the data that helps extract information, improves understanding and visualization of the structure. When the dataset consists of d-dimensional vectors, simple representations of the data may consist in trees or ultrametrics, and the goal is to best preserve the distances (i.e.: dissimilarity values) between data elements. To circumvent the quadratic running times of the most popular methods for fitting ultrametrics, such as average, single, or complete linkage, Cohen-Addad et al. (2020) recently presented a new algorithm that for any $c \geq 1$, outputs in time $n^{1+O(1/c^2)}$ an ultrametric Δ such that for any two points u, v, $\Delta(u, v)$ is within a multiplicative factor of $5c$ to the distance between u and v in the "best" ultrametric representation. We improve the above result and show how to improve the above guarantee from $5c$ to $\sqrt{2}c + \varepsilon$ while achieving the same asymptotic running time. To complement the improved theoretical bound, we additionally show that the performances of our algorithm are significantly better for various real-world datasets.

1. Introduction

Simplifying, sparsifying or sketching datasets so as to retain the most crucial information and preserve the quality of the signal is a central question in machine learning, statistics and data analysis. In this context, reducing the size of the dataset, by for example reducing the dimension, has to be done in such a way that the essential properties of the dataset are preserved. An iconic example is the *principal component analysis* which leads to a representation of the dataset in a lower dimensional space that helps identify underlying cluster structures. Since the 50s, a large toolkit of metric embedding techniques have been developed in a variety of communities so as to "simplify" datasets while preserving some relevant structure of the input dataset.

In this paper, we aim at computing embedding of high-dimensional Euclidean data into *ultrametrics*[1]. Ultrametrics are simple hierarchical representations of metric spaces which are particularly useful in clustering settings when very little is known about the data (i.e.: if the target number of clusters is unknown). They could for example be used for finding hierarchical clustering, unveiling global hierarchical structure over data elements (such as phylogenic trees), or as visualization tools (the reader may refer to the extensive work of Carlsson & Mémoli (2010)).

The most popular approaches for computing ultrametrics are arguably the *linkage* algorithms, average-linkage, single-linkage, Ward's method and complete-linkage. These methods build a hierarchical clustering by iteratively merging the clusters at minimum distance, and only differ by their definitions of cluster distance. Unfortunately, for all the above algorithms, finding the closest clusters incurs a quadratic running time for somewhat high dimensional inputs (i.e.: $\Omega(\log n)$-dimensional inputs). This is a major roadblock even when dealing with moderate size datasets (as shown in the experiments, going above 50 000 points dataset is infeasible for regular computers) and so designing efficient ultrametric embeddings has been an important research problem.

In this paper, we follow the approach of Farach et al. (1995); Cohen-Addad et al. (2020) and measure the quality of an ultrametric by how much it *distorts* the distances between the input vectors. Namely, we aim at designing an ultrametric Δ such that for each pair of points a, b, $||a - b||_2 \leq \Delta(a, b) \leq \alpha||a - b||_2$ for the smallest possible α (we defer the reader to Section 2 for a formal definition). This indeed approximately preserves the hierarchical structure of the data: Given three points p_1, p_2, p_3, if $||p_1 - p_2||_2 < \alpha||p_1 - p_3||_2$, then $\Delta(p_1, p_2) \leq \Delta(p_1, p_3)$ and so the natural distance ordering is thus approximately preserved. The problem of computing the optimal ultrametric for the above objective function, which we refer

[*]Equal contribution [1]Google Research, Zurich [2]remi.de.joannis.de.verclos@ens-lyon.org [3]LaBRI, CNRS. Correspondence to: Guillaume Lagarde <guillaume.lagarde@labri.fr>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]An ultrametric (X, d) is a metric space where for each $x, y, z \in X$, $\Delta(x, y) \leq \max(\Delta(x, z)(, z, y))$

to as BUF_∞ following Cohen-Addad et al. (2020) notation, can be solved in time $O(n^2 d + n^2 \log n)$ Farach et al. (1995). Cohen-Addad et al. (2020) completed their result by showing that assuming some complexity hypothesis, no subquadratic algorithm exists. Unfortunately, a quadratic running time is also the main roadblock for all the current linkage algorithms. Hence, Cohen-Addad et al. (2020) turned to designing subquadratic approximation algorithms for the problem and described an algorithm running in time $O(nd + n^{1+1/c^2+o(1)})$ that returns a $5c$-approximation to the optimum ultrametric.

1.1. Our Results

We provide a simple algorithm, that for any $\gamma > 1$, returns a $(\sqrt{2}\gamma + \varepsilon)$-approximation for BUF_∞ in time $O(nd + n^{1+1/\gamma^2+o(1)})$, improving on the approximation guarantee by a factor at least 3.53 while achieving the same asymptotic running time.

Theorem 1. *For any $\gamma > 1$, $\varepsilon > 0$, there is an algorithm that produces a $(\sqrt{2}\gamma + \varepsilon)$-approximation in time*

$$O\left(n \cdot \left(n^{1/\gamma^2+o(1)} + d \cdot \left(\frac{\log 1/\varepsilon}{\varepsilon^2} + \frac{\log n}{\varepsilon}\right) + \frac{\log 1/\varepsilon}{\varepsilon^{4.5}}\right)\right)$$

for Euclidean instances of BUF_∞ of dimension d.

Empirical results To show that the improvement of the approximation guarantee by a factor $\frac{5}{\sqrt{2}+\varepsilon} \sim 3.535 - \varepsilon$ is indeed significant, we implemented our algorithm and analyzed its performances on four classic datasets (IRIS, SHUTTLE, MICE, PENDIGITS). We compared the results with classic linkage algorithms (single, complete, average), Ward's method from the Scikit-learn library Pedregosa et al. (2011) and from the FastCluster library Müllner (2013), the algorithm of Cohen-Addad et al. (2020), and the optimum algorithm of Farach et al. (1995). We show that our algorithm indeed produces a significantly better ultrametric (in terms of the BUF_∞ objective) than the algorithm of Cohen-Addad et al. (2020) (from 2 to 3 times better) while achieving a comparable running time on the largest dataset (i.e.: SHUTTLE). Our algorithm is also much faster than single linkage on datasets containing at least 10000 points.

1.2. Related Work

The landmark work of Carlsson & Mémoli (2010) on ultrametric embedding, hierarchical clustering and linkage algorithms has helped better understand the structure of the outputs produced by the linkage algorithms. The connection between ultrametric and hierarchical clustering, as phrased for example by Dasgupta (2015), has been thoroughly explored in Cohen-Addad et al. (2019; 2017). In particular, Cohen-Addad et al. (2019); Moseley & Wang (2017) proved that average-linkage yields a constant-factor approximate solution to the dual of Dasgupta's objective function. This line of research has been pushed forward in successive works (see e.g.:(Roy & Pokutta, 2016; Charikar & Chatziafratis, 2017; Cohen-Addad et al., 2017; Charikar et al., 2019; 2018; Alon et al., 2020; Chatziafratis et al., 2020), see also (Balcan et al., 2008; Chami et al., 2020)). A major difference between these works and ours is that we are focused on designing an ultrametric to approximate distances or dissimilarity between the input elements, namely finding a good ultrametric embedding, while they are interested in producing a hierarchical clustering tree with "good" clustering properties (measured in terms of a different objective function).

Related papers also include Cochez & Mou (2015) and Abboud et al. (2019) which aims at providing linkage algorithms that only approximately merge the closest clusters. These works are essentially focused on providing approximate guarantees on the distances between the clusters merged (e.g.: the clusters merged are at distance at most α times the closest distance). Unfortunately, these approaches are somewhat incomparable to ours since the output is not guaranteed to be an ultrametric.

1.3. Technical Overview

The approach of Farach et al. (1995); Cohen-Addad et al. (2020) consists in computing a Minimum Spanning Tree (MST) T and sorting the edges of the MST by their *cut weight* values (we will come back to this notion below). The edge e of T with the largest cut weight w will form the root of the ultrametric and points in different connected components of $T - e$ will be at distance w in the ultrametric. The algorithm then proceeds recursively on each connected component of $T - e$, hence defining an ultrametric.

There are two computational bottlenecks in the above algorithm description: (1) the MST computation, and (2) the computation of the cut weights of the edges of the MST. Computing the MST of a set of n points in Euclidean space of dimension $\Omega(\log n)$ is a major open problem and there is no subquadratic algorithm known. Hence, (Cohen-Addad et al., 2020) resorted to use an approximate minimum spanning tree (with some additional properties that we describe in the next sections) and we follow the same approach here. This approach leads to compute a γ-approximate MST in time $n^{1+1/\gamma^2+o(1)}$. To improve over the algorithm of (Cohen-Addad et al., 2020), both in theory and in practice, one must thus improve on the cut weight computation. Given an edge e of an MST T, the cut weight of e is given by the maximum distance between points of different connected components of $T - e$. This is essentially a *bichromatic diameter* problem – which cannot be solved exactly in subquadratic time assuming some popular

complexity hypothesis. (Cohen-Addad et al., 2020) show how to efficiently obtain a 5-approximation of this value by simply keeping track of an estimate of the diameter of each connected component and combining these estimates with the maximum distance from an arbitrary point of one side to the points of the other side. While better than 5-approximation linear-time algorithms for the bichromatic diameter are known, (Cohen-Addad et al., 2020) made the choice of using the above simple algorithm. Arguably, the main reason for this is that the entire algorithm requires to perform in total $n-1$ bichromatic diameter computations to compute the cut weights of all the edges of the MST and so, the algorithm must be able to re-use information from different bichromatic diameter computations to run in subquadratic time. In other words, a bichromatic diameter computation must run *in average* over all bichromatic diameter computation in time $O(n^{1/\gamma^2+o(1)})$ to match the claimed bound. We show how to achieve this while ensuring an approximation guarantee of $\sqrt{2}+\varepsilon$, matching the best known approximation bound for a single bichromatic computation, and improving over the 5-approximation of (Cohen-Addad et al., 2020).

Our key technical insight is that one can use constant-size core-sets (small summaries of the input that preserve some specific properties) for estimating the minimum enclosing ball (up to a $(1+\varepsilon)$ multiplicative factor on the radius of this ball). Equipped with this, we show how one can approximate the bichromatic diameter problem within a factor $\sqrt{2}+\varepsilon$ while maintaining the core-set efficiently over the different bichromatic diameter computations. This bound is the best known so far for the bichromatic diameter problem and so improving on this will require to come up with a significantly different algorithm.

2. Preliminaries

An ultrametric space (X,Δ) is a metric space where the triangle inequality is strengthened to the so-called ultrametric inequality:

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

If X is finite, a convenient way to represent such an ultrametric space is via a rooted tree T together with a weight function $w: T\to\mathbb{R}^+$ such that

- there is a one-to-one correspondence between the leaves of T and the set X,
- the weight of any leaf is zero,
- the weights are decreasing along any path from the root to a leaf.

The ultrametric space induced by (T,w) is given by

$$\Delta(x,y)=w(\mathrm{LCA}(x,y))$$

where $\mathrm{LCA}(x,y)$ is the least common ancestor of x and y in the tree T. It can be shown that any finite ultrametric space can be represented in this way.

2.1. Ultrametric embedding

We denote by $\ell_2:(x,y)\to\mathbb{R}^+$ the metric of a Euclidean space. If $e=(x,y)$ is an edge, then we will write $\ell_2(e)$ as a shortcut for $\ell_2(x,y)$. Our goal is to find interesting embeddings of a Euclidean space into an ultrametric space, meaning essentially that we want the distances in the ultrametric space to be as close as possible to the original distances. To this end, we consider the BEST ULTRAMETRIC FIT problem (BUF_∞), defined as follows:

- **INPUT:** (X,ℓ_2) a set of points in some Euclidean space.
- **OUTPUT:** an ultrametric (X,Δ) such that: $\forall x,y\in X, \ell_2(x,y)\leq\Delta(x,y)\leq\alpha\cdot\ell_2(x,y)$, for the absolute minimal value α.

An ultrametric $\widehat{\Delta}$ is said to be a γ-approximation of the best ultrametric if for any pair of points x,y we have $\ell_2(x,y)\leq\widehat{\Delta}(x,y)\leq\gamma\cdot\alpha\cdot\ell_2(x,y)$.

2.2. Approximation algorithm from (Cohen-Addad et al., 2020)

In this section, we overview an approximation algorithm from Cohen-Addad et al. (2020) which can be implemented to run in time $O(nd+n^{1+1/\gamma^2})$ and for which the output is guaranteed to be a $5\cdot\gamma$-approximation of the best ultrametric. We build on this result and improve it to get a $(\sqrt{2}+\varepsilon)\cdot\gamma$ approximation algorithm. Before diving into the algorithm, we need to define a few more notions.

Let T be a spanning tree over the set of points X. Without loss of generality assume[2] the edges have different weights. For an edge $e=(x,y)\in T$, we denote by $L(e)$ (respectively $R(e)$) the connected component containing x (respectively y) in $T\setminus\{e'\in T\mid \ell_2(e')\leq\ell_2(e)\}$.

The *cut weight* $\mathrm{CW}(e)$ of an edge $e\in T$ is defined as the maximal distance between a point in $L(e)$ and a point in $R(e)$; in symbols:

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

[2]Removing this assumption generates small technicalities that we prefer to avoid for clarity.

Given a spanning tree T together with a weight function $w : T \to \mathbb{R}$, a *Cartesian tree* of (T, w) is a weighted rooted tree defined as follows: if $|T| = 1$ then its Cartesian tree is the unique vertex associated with a zero weight; otherwise the root of the Cartesian tree corresponds to an edge e of maximal weight, and its two children are obtained inductively as Cartesian trees of the two connected components of $T \setminus \{e\}$.

Algorithm 1 is proven to output a $\gamma \cdot \delta$-approximation of the best ultrametric.

Algorithm 1 $\gamma \cdot \delta$-approximation for BUF_∞

1: Compute a γ-approximated MST T over the set of points
2: Compute a δ-estimate $\widehat{\mathrm{CW}}(\cdot)$ of the cut weights with respect to T.
3: Return a Cartesian tree of $(T, \widehat{\mathrm{CW}}(\cdot))$.

In (Cohen-Addad et al., 2020), the authors show how to implement, for Euclidean spaces

- Step 1 in time $O(nd + n^{1+1/\gamma^2})$ by constructing a γ-spanner via locality sensitive hash families (Har-Peled et al., 2013),
- Step 2 in time $O(nd + n \log n)$ for $\delta = 5$ by tweaking a disjoint-set data structure.

Step 3 is easy and can be done in time $O(n \log n)$ using again a disjoint-set data structure.

Here we build on this work and improve Step 2. We show how to improve the approximation factor from $\delta = 5$ to $\delta = \sqrt{2} + \varepsilon$ for this step. We keep the same structure of the algorithm and do not change Steps 1 and 3.

3. Improved approximation algorithm for the cut weights

We work with a set X of points in a Euclidean space of dimension d. Set $\Lambda = \frac{\max_{x,y \in X} \ell_2(x,y)}{\min_{\substack{x,y \in X \\ x \neq y}} \ell_2(x,y)}$. We denote by $\mathcal{B}(c, r)$ the ball of center c and radius r. Given a set of points S, the *minimum enclosing ball*, denoted by $\mathrm{MEB}(S)$, is the unique ball containing S with minimum radius. Given $\varepsilon > 0$, an *ε-core-set* of S is a set $C \subseteq S$ for which $S \subseteq \mathcal{B}(c, (1+\varepsilon)r)$ where $\mathcal{B}(c, r) = \mathrm{MEB}(C)$, meaning in particular that r is a $(1 + \varepsilon)$-approximation of the actual radius of $\mathrm{MEB}(S)$. Maybe surprisingly, every set has a core-set of size $O(1/\varepsilon)$; this quantity depends neither on the number of points nor on the space dimension. We recall the construction since we will need to modify it for our stronger result.

Algorithm 2 Core-set computation

1: Pick p arbitrarily in P.
2: Set $C = \{p\}$ and $\mathcal{B}(c, r) = \mathcal{B}(p, 0)$
3: **while** $\exists x \in P$ such that $\ell_2(c, x) > (1 + \varepsilon)r$ **do**
4: Find a farthest point $x \in S$ from c and add it to C
5: Compute $\mathcal{B}(c, r) = \mathrm{MEB}(C)$
6: **end while**
7: Return $\mathcal{B}(c, r)$ and C

It is proven in (Badoiu & Clarkson, 2003) that Algorithm 2 always outputs a core-set C of size at most $2/\varepsilon$ when the computations of the minimum enclosing balls $\mathrm{MEB}(C)$ are exact; it is shown in (Kumar et al., 2003) that a minimum enclosing ball $\mathrm{MEB}(C)$ can be computed with arbitrary precision ε' in time $O(|C|d + |C|^{3.5} \log(1/\varepsilon'))$ arithmetic operations. The while operation is iterated exactly $|C|$ times; each such iteration costs nd for checking the condition and at most $|C|d + |C|^{3.5} \log(1/\varepsilon')$ for computing the minimum enclosing ball. This leads to a running time of $O(|C|nd + |C|^2 d + |C|^{4.5} \log(1/\varepsilon'))$. Using $|C| \leq n$ and $|C| = O(\frac{1}{\varepsilon})$ gives a running time of $O(\frac{nd}{\varepsilon} + \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon})$ for Algorithm 2.

Their bound on the size of the core-set is based on the following property.

Lemma 1 ((Badoiu et al., 2002)). *At each iteration of Algorithm 2, the radius of* $\mathrm{MEB}(C)$ *increases by a factor at least* $(1 + \varepsilon^2/16)$.

3.1. A $(\sqrt{2} + \varepsilon)$-approximation algorithm

This section is devoted to the following theorem.

Theorem 2. *There exists an algorithm for computing a $(\sqrt{2} + \varepsilon)$-approximation of the cut-weights running in time*

$$O\left(n \cdot d \cdot \left(\frac{\log \Lambda}{\varepsilon^2} + \frac{\log n}{\varepsilon}\right) + n \cdot \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon}\right)$$

Here is a high-level description of Algorithm 3: It maintains a partition of X in a disjoint-set data structure $\mathcal{U}$. For each set $S \in \mathcal{U}$ of this, it keeps track of an ε-core-set $\textsc{core}(S)$ over S and its minimum enclosing ball $\mathcal{B}(c_S, r_S)$.

At the beginning of the process, it constructs a set for each point in X. Then, it consider the edges of the MST in increasing order of length (recall that we want to estimate the cut weights of these edges). When at edge $e = (x, y)$, it looks at the two sets S_x and S_y of $\mathcal{U}$ containing x and y respectively. Suppose w.l.o.g that S_x is bigger than S_y; then the cut weight of e is estimated using the following formula

$$\widehat{\mathrm{CW}}(e) = (1 + \varepsilon) \cdot r_{S_x} + \max_{z \in S_y} \ell_2(c_{S_x}, z).$$

Claim 1. *The above formula is such that*

$$\mathrm{CW}(e) \leq \widehat{\mathrm{CW}}(e) \leq (\sqrt{2} + \varepsilon) \cdot \mathrm{CW}(e)$$

Algorithm 3 Cut-weights approximation

for all $x \in X$ **do**
  Create the set $\{x\}$ with $c_{\{x\}} = x, r_{\{x\}} = 0$, $\textsc{core}(\{x\}) = \{x\}$
end for
for $e = (x, y) \in T$, taken in increasing order of $\ell_2(e)$ **do**
  $S_x \leftarrow \textsc{find}(x)$
  $S_y \leftarrow \textsc{find}(y)$
  If $|S_x| < |S_y|$, invert the roles of x and y
  $L \leftarrow \max_{z \in S_y} \ell_2(c_{S_x}, z)$
  $\widehat{\mathrm{CW}}(e) \leftarrow (1+\varepsilon) \cdot r_{S_x} + L$
  $S \leftarrow \textsc{union}(S_x, S_y)$
  $\textsc{core}(S) \leftarrow \textsc{core}(S_x)$
  $c_S, r_S \leftarrow c_{S_x}, r_{S_x}$
  if $L > (1+\varepsilon) \cdot r_{S_x}$ **then**
    $\textsc{Update}(S, x)$
  end if
end for
Return $(\widehat{\mathrm{CW}}(e))_{e \in T}$

Algorithm 4 $\textsc{Update}(S, x)$

if $|\textsc{core}(S)| > 20/\varepsilon$ **then**
  $\textsc{core}(S) \leftarrow \{x\}$
  $\mathcal{B}(c_S, r_S) \leftarrow \mathcal{B}(x, 0)$
end if
while $\exists z \in S$ with $\ell_2(c_S, z) > (1+\varepsilon) r_S$ **do**
  Add a farthest point $z \in S$ from c_S to $\textsc{core}(S)$
  Compute $\mathcal{B}(c_S, r_S) = \mathrm{MEB}(\textsc{core}(S))$
end while

Proof. Suppose p_1, p_2 are two points respectively in S_x and S_y with $\ell_2(p_1, p_2) = \mathrm{CW}(e)$. By the triangle inequality we have that $\ell_2(p_1, p_2) \leq \ell_2(p_1, c_{S_x}) + \ell_2(c_{S_x}, p_2)$. We then use the facts that $\ell_2(p_1, c_{S_x}) \leq (1+\varepsilon) \cdot r_{S_x}$ – since $\textsc{core}(S_x)$ is an ε-core-set – and that $\ell_2(c_{S_x}, p_2) \leq \max_{z \in S_y} \ell_2(c_{S_x}, z)$ to conclude for the first inequality. For the second inequality, let $q = \arg\max_{z \in S_y} \ell_2(c_{S_x}, z)$. Consider then the hyperplane H orthogonal to $\overrightarrow{c_{S_x} q}$ going through c_{S_x} and let V be the region of the space partitioned by H not containing q. By a basic property of the minimum enclosing ball (Kumar et al., 2003, Lemma 2), there is a point $q' \in S_x$ in the intersection of V and the boundary of $\mathrm{MEB}(\textsc{core}(S_x))$; q' is thus at distance r_{S_x} from c_{S_x}. By definition $\mathrm{CW}(e) \geq \ell_2(q, q')$. Since q' is in V we get that $\ell_2(q, q') \geq \sqrt{r_{S_x}^2 + \ell_2(c_{S_x}, q)^2}$ (with equality if q' lies in H). Overall, $\frac{\widehat{\mathrm{CW}}(e)}{\mathrm{CW}(e)} \leq \frac{(1+\varepsilon) \cdot r_{S_x} + \ell_2(c_{S_x}, q)}{\sqrt{r_{S_x}^2 + \ell_2(c_{S_x}, q)^2}}$. Finally, applying the Cauchy-Schwarz inequality to the numerator yields

$$\frac{\widehat{\mathrm{CW}}(e)}{\mathrm{CW}(e)} \leq \sqrt{(1+\varepsilon)^2 + 1^2} \leq \sqrt{2} + \varepsilon,$$

as desired. □

We then merge S_x and S_y using the standard disjoint-set algorithm. The core-set of $S = S_x \cup S_y$ is build by updating the core-set of S_x as follows: if every point in S_y is already in the ball $\mathcal{B}(c_{S_x}, (1+\varepsilon) r_{S_x})$, we set $\textsc{core}(S) = \textsc{core}(S_x)$ (and, consistently, $\mathcal{B}(c_S, r_S) = \mathcal{B}(c_{S_x}, r_{S_x})$). Otherwise, we start with $\textsc{core}(S) = \textsc{core}(S_x)$ and we iteratively add to this set the farthest point from the center of $\mathrm{MEB}(\textsc{core}(S))$ until we obtain an ε-core-set in S, as in Algorithm 2. An analysis of this algorithm in (Badoiu & Clarkson, 2003, Proof of Lemma 2.1) gives the following bound.

Lemma 2 ((Badoiu & Clarkson, 2003))**.** *The while loop in Algorithm 4 terminates after at most* $\frac{2}{\varepsilon}$ *iterations.*

In order to keep the size of the core-sets small enough, we further reset it to a singleton before the update if its size is more than a given threshold (namely $20/\varepsilon$). Together with Lemma 2, this ensures that the core-sets have size $O(1/\varepsilon)$.

Claim 2. *The overall running time of this algorithm is* $O\left(n \cdot d \cdot (\frac{\log \Lambda}{\varepsilon^2} + \frac{\log n}{\varepsilon}) + n \cdot \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon}\right)$

Proof. The complexity of the operations related to the disjoint-set data structure (namely "create a set", "find" and "union") has a total running time of $O(n \log n)$ when a standard implementation is used.

Sorting the edges of the tree T according to their length is done in $O(nd + n \log n)$ time. Computing L at a given iteration is done in time $d \cdot |S_y| = d \cdot \min(|S_x|, |S_y|)$ by querying the distances between c_{S_x} and every point in S_y; thus the contribution of this parts to the running time of the whole algorithm is $O(n \cdot \log n \cdot d)$.

Given an edge $e \in T$, denote by $k(e)$ the number of iterations of the while loop in Algorithm 4 when treating this edge (so $k(e) = 0$ if at this step $L \leq (1+\varepsilon) \cdot r_{S_x}$). Let $S_x(e)$, $S_y(e)$ and $S(e) = S_x(e) \cup S_y(e)$ be the corresponding sets at this iteration.

We know from Lemma 2 that $k(e) \leq 2/\varepsilon$, so the number of times $\mathrm{MEB}(.)$ is called is $\sum_{e \in T} k(e) = O(n/\varepsilon)$. Further, since $|C| = O(1/\varepsilon)$ each MEB can be computed in time $O(\frac{d}{\varepsilon} + \frac{1}{\varepsilon^{3.5}} \log \frac{1}{\varepsilon})$, The total time spent computing minimal enclosing balls is therefore $O(n \cdot (\frac{d}{\varepsilon^2} + \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon}))$

It remains to estimate the time spent to search for the farthest points in the while loop. The running time of this loop (excluding $\mathrm{MEB}(.)$) at Step e is of the order of $d \cdot |S(e)| \cdot$

$k(e)$, which in total gives:

$$d \cdot \sum_{e \in T} |S(e)| \cdot k(e) = d \cdot \sum_{z \in X} \sum_{e \in T | z \in S(e)} k(e) \quad (1)$$

We close the proof by showing that for every $z \in X$, the sum $k := \sum_{e \in T | z \in S(e)} k(e)$ is upper bounded by $O(\frac{\log \Lambda}{\varepsilon^2} + \frac{\log n}{\varepsilon})$.

To see this, let us fix $z \in X$ and consider $e_1, \dots, e_\ell$ the edges from $\{e \in T \text{ s.t. } z \in S(e)\}$, taken in order of appearance. We need to estimate $k := \sum_{i=1}^{\ell} k(e_i)$. Let r_i and C_i be the final values of $r_{S(e_i)}$ and $\textsc{core}(S(e_i))$ at the end of Step e_i.

For a given Step e_i, we consider three scenarios. (1) If $z \in S_x(e_i)$ and $|C_{i-1}| \leq 20/\varepsilon$, then it holds that $|C_i| - |C_{i-1}| = k(e_i)$. (2) If $z \in S_x(e_i)$ and at this step $\textsc{core}(S)$ is reset to $\{x\}$ (because $|\textsc{core}(S_x(e_i))| > 20/\varepsilon$), then $|C_i| - |C_{i-1}| \leq k(e_i) - |C_{i-1}| \leq -18/\varepsilon$ by Lemma 2. (3) If $z \in S_y(e_i)$, we use the general bound $|C_i| - |C_{i-1}| \leq 22/\varepsilon$.

Writing n_1, n_2 and n_3 the number of time each of these scenarios respectively happen, and k_1 the sum of $k(e_i)$ over the n_1 indices corresponding to scenario (1), we therefore have

$$0 \leq |C_{n-1}| - |C_0| \leq k_1 - n_2 \cdot \frac{18}{\varepsilon} + n_3 \cdot \frac{22}{\varepsilon}.$$

so $n_2 \leq k_1 \cdot \varepsilon/18 + 2n_3$. Note that Scenario (3) happens at most $\log n$ times since in this case $S(e_i)$ doubles (at least) in size because $S(e_{i-1}) = S_y(e_i)$ and $S(e_i) = S_x(e_i) \cup S_y(e_i)$ (and recall that $|S_y(e_i)| \leq |S_x(e_i)|$). It follows that

$$n_2 + n_3 \leq k_1 \cdot \varepsilon/18 + 3 \log n. \quad (2)$$

Let us now look at the variations of r_i. Since r_i is an ε-approximation of the radius of $\text{MEB}(S(e_i))$ and $S(e_{i-1}) \subseteq S(e_i)$, we always have $r_i/r_{i-1} \geq 1/(1+\varepsilon)$. In the case of scenario (1), we have instead $r_i/r_{i-1} \geq (1+\varepsilon^2/16)^{k(e_i)}$ by Lemma 1. In total, this yields

$$\frac{r_\ell}{r_2} = \prod_{i=3}^{\ell} \frac{r_i}{r_{i-1}} \geq \frac{(1+\varepsilon^2/16)^{k_1 - 1}}{(1+\varepsilon)^{n_2+n_3}}.$$

Bounding r_ℓ/r_2 by Λ and taking the log,

$$\log \Lambda \geq (k_1 - 1) \cdot \log(1 + \frac{\varepsilon^2}{16}) - (n_2 + n_3) \log(1+\varepsilon),$$

and further $k_1 \leq (n_2+n_3)(16/\varepsilon + o(1/\varepsilon)) + O(\log \Lambda/\varepsilon^2)$. Using (2) to bound $n_2 + n_3$, we obtain $k_1 = O(\log n/\varepsilon + \log \Lambda/\varepsilon^2)$.

We are now ready to estimate k. It follows from Lemma 2 that $k \leq k_1 + 2/\varepsilon \cdot (n_2 + n_3) \leq 10/9 \cdot k_1 + 6/\varepsilon \cdot \log n = O(\log n/\varepsilon + \log \Lambda/\varepsilon^2)$ by (2) and the inequality above.

□

Theorem 2 follows from Claim 1 and Claim 2.

3.2. Removing the dependency in Λ

In this section, we propose an algorithm whose complexity does not depend on Λ by proving the following theorem.

Theorem 3. *There exists an algorithm that computes a $(\sqrt{2}+\varepsilon)$-approximation of the cut-weights in time*

$$O\left(n \cdot d \cdot (\frac{\log 1/\varepsilon}{\varepsilon^2} + \frac{\log n}{\varepsilon}) + n \cdot \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon}\right).$$

The idea is to forget all points but one that were in the minimum enclosing balls of radius small enough. More precisely, we maintain the same data structure as before and we also keep track of a "threshold" $\textsc{th}(z)$ for any point $z \in X$ together with, for each set $S \in \mathcal{U}$ a *representative member* of S that we denote by $p(S)$. At the beginning, we set $\textsc{th}(x) = \infty$ and $p(\{x\}) = x$ for every $x \in X$. Thresholds and representative members are updated as follows: when S_x and S_y are merged into $S = S_x \cup S_y$, we set $\textsc{th}(p(S_y))$ to be the radius r_S and we define $p(S) := p(S_x)$.

For a set S of the disjoint-set data structure, we denote by S' the points of S that were removed, and let $S^* = S \setminus S'$. When the threshold is small enough compared to the radius of the minimum enclosing balls in which a point z is contained, then we "forget" the point, meaning that we remove this point from S^*. More precisely: If a point $z \in S$ is such that $\textsc{th}(z) \leq r_S \cdot \frac{\varepsilon}{4} \cdot \frac{1}{1+2\varepsilon}$ then we forget the point z. The main change in the algorithm lies in the update function (Algorithm 4) at Steps 5 and 6 where we look for z with $\ell_2(c_S, z) > (1+\varepsilon) \cdot r_S$ in S^* instead of S, thus giving a smaller running time.

Lemma 3. *At any moment of the algorithm, if $\mathcal{B}(c^*, r^*) = \text{MEB}(S^*)$, then for every $x \in S$, there is a point $y \in S^*$ with $\ell_2(x,y) \leq \varepsilon r^*$.*

Proof. Suppose we are at Step e_0 and that the lemma is satisfied for every earlier step. Let $E = \{e \in T | S(e) \subseteq S(e_0)\}$ and $M = \max\{r_{S(e)} | e \in E\}$.

Claim 3. *For any point $x \in S'$, there exists $y \in S^*$ such that*

$$\ell_2(x,y) \leq M \cdot \frac{\varepsilon}{2}$$

Let $S(e_1), \dots, S(e_\ell) = S$ be the successive sets from $\mathcal{U}$ containing x and define $y_i = p(S(e_i))$ their representative members, and $y_0 = p(S(e_0))$. Let j be the minimum index so that $p(S(e_j))$ is in S^* (observe that $j > 0$ because $x \in S'$ and j exists since $\textsc{th}(p(S)) = \infty$. Now consider the ball $B = \mathcal{B}(c_{S(e_j)}, (1+2\varepsilon) r_{S(e_j)})$, where $c_{S(e_j)}$ and $r_{S(e_j)}$ are the center and radius computed at Step e_j. Since $c_{S(e_j)}$ and $r_{S(e_j)}$ are constructed by Algorithm 4 using points of

S^*, we have $S^*(e_j) \subseteq \mathcal{B}(c_{S(e_j)}, (1+\varepsilon)r_{S(e_j)})$. Further, since the lemma is satisfied at the earlier Step e_j, we have $S(e_j) \subseteq B$. As a consequence, x and y_j are in B because they are in $S(e_j)$. So $\ell_2(x, y_j) \leq 2(1+2\varepsilon) \cdot r_{S(e_j)}$.

It remains to estimate $r_{S(e_j)}$. By construction, the threshold of $y_{j-1} = p(S(e_{j-1}))$ is $\text{TH}(y_{j-1}) = r_{S(e_j)}$. Since $y_{j-1} \in S'$, the threshold $\text{TH}(y_{j-1})$ is at most $\frac{\varepsilon}{4(1+2\varepsilon)} r_{S(e')}$ for some $e' \in E$, and therefore $r_{S(e_j)} \leq \frac{\varepsilon}{4(1+2\varepsilon)} M$ since $r_{S(e')} \leq M$. It follows that $\ell_2(x, y_j) \leq \frac{\varepsilon M}{2}$ and since $y_j \in S^*$, it concludes the proof of the claim.

It is clearly enough to prove the following claim to conclude for the lemma.

Claim 4. $M/2 \leq r^*$

To see this, let r the radius of the minimum enclosing ball of S. Observe that $r \geq M$ because M is the radius of the minimum enclosing of a subset of S. The previous claim shows that any point in S' is at distance $(\varepsilon/2)M \leq \varepsilon r$ from a point in S^*. Therefore, $\mathcal{B}(c^*, r^* + \varepsilon M/2)$ contains S, so $r \leq r^* + \varepsilon M/2$. It follows that $r^* \geq M(1-\varepsilon/2) \geq M/2$ if $\varepsilon \leq 1$. This concludes the proof of the claim.

□

Lemma 3 shows that the modified algorithm gives a $(1+\varepsilon)^2$ approximation of the minimum enclosing balls of S based on the core-set of S^* (we lose a factor $(1+\varepsilon)$ from the core-sets and the same factor from S to S^*). Therefore the cut weights are still approximated within a factor of $\sqrt{2}+\varepsilon'$ for a well chosen $\varepsilon' = 2\varepsilon + o(\varepsilon)$.

Claim 5. *The overall running time of this algorithm is* $O\left(n \cdot d \cdot (\frac{-\log \varepsilon}{\varepsilon^2} + \frac{\log n}{\varepsilon}) + n \cdot \frac{1}{\varepsilon^{4.5}} \log \frac{1}{\varepsilon}\right)$.

Sketch of the proof. The main difference with the previous section is that the cost of the update (1) is instead estimated by

$$d \cdot \sum_{e \in T} |S^*(e)| \cdot k(e).$$

We rewrite the sum as $|T| + \sum_{e \in T} |S^*(e) \setminus \{p(S)\}| \cdot k(e)$. The term $|T|$ is bounded by n so contribute to a factor of nd to the total running time. To give an upper bound on the sum, we follow the same strategy as in proof of Claim 2 and first rewrite it as $\sum_{z \in X} \sum_{e \in T | z \in S^*(e) \setminus \{p(S)\}} k(e)$. Now observe that a point z contributes to this sum from a Step e when $r := r_{S(e)}$ is assigned to $\text{TH}(z)$ (i.e., when z stops to be the representative member of its set) until it is removed from S^*, which happens when the radius of $r_{S(e')}$ for $S(e')$ containing z goes above $R := 4(1+2\varepsilon)\frac{\text{TH}(z)}{\varepsilon}$. Note that, $R/r = 4\frac{(1+2\varepsilon)}{\varepsilon}$. A similar analysis as in the proof of Claim 2, where $R/r = O(\frac{1}{\varepsilon})$ plays the role of Λ gives that

$$\sum_{e \in T | z \in S^*(e) \setminus \{p(S)\}} k(e) = O(\frac{\log n}{\varepsilon} + \frac{-\log \varepsilon}{\varepsilon^2}),$$

this is sufficient to conclude. □

Theorem 3 follows from Lemma 3 and Claim 5.

4. Experiments

We performed experiments to compare our implementation of Algorithm 1 using core-sets to standard agglomerative clustering algorithms (Ward, Single, Centroid). Our implementation is coded using the Cython extension for Python and relies on the C++ library miniball based on the algorithm in (Fischer et al., 2003) through its Cython binding cyminiball to compute MEBs.

4.1. Performance evaluations

We evaluated the algorithm's computation time and distortion. To make the distortions comparable, we rescaled the ultrametrics by the largest number so that the resulting ultrametrics is smaller or equal to the Euclidean distance on the points of the dataset. Practically, this boils down to estimating the distortion as $\max_{u \neq v} \frac{\Delta(u,v)}{\ell_2(u,v)} / \min_{u \neq v} \frac{\Delta(u,v)}{\ell_2(u,v)}$, where Δ stands for the ultrametrics distance and the $\min$ and the $\max$ are taken on all pairs of distinct points of the dataset. Note that for our algorithm, the $\min$ part is always equal to 1 since this ultrametrics is an upper bound on the Euclidean distance that is tight on sub-trees of size 2 or less.

Table 1. The datasets used in our experiments

	size	dimension
IRIS	150	4
MICE	1080	82
PENDIGITS	10992	16
SHUTTLE	58000	9

The running time and distortion on four classic datasets (IRIS, MICE, PENDIGITS, SHUTTLE, see Table 1 for a complete description, all datasets are from the UCI ML repository (Dua & Graff, 2017)) are reported on Table 2. The test have been made on a laptop with 8GB of memory and a processor Intel i5-8265U with frequency 1.60GHz. The distortion and time reported are an average over 100 measurements. **CoreSet** is our algorithm, using the parameter $\varepsilon = 0.2$ for core-sets. **CKL** is our implementation of Algorithm 1 in (Cohen-Addad et al., 2020), that therefore differs from **CoreSet** only on Step 2. **Ward** , **Single** , **Complete** , **Average** and **Centroid** are the implementation of the

Ward, single linkage, complete linkage, average linkage and centroid linkage algorithms in existing Python libraries. We first used the Scikit-learn library. The functions of Scikit-learn run out of memory on the bigger dataset (SHUTTLE), so we added the very well optimized fastcluster library to the experiment.

Our algorithm gives results of variable quality. It showed a standard deviation on the distortion of 13% (of the mean) for MICE, 11% for PENDIGITS and 22% for SHUTTLE. Note that the standard linkage algorithms are deterministic.

Table 2. Time and distortion comparison with classic clustering algorithms.

	MICE		PENDIGITS		SHUTTLE	
	$dist$	$T(s)$	$dist$	$T(s)$	$dist$	$T(s)$
CoreSet	15.13	0.056	27.26	0.41	106.8	5.44
CKL	38.20	0.022	82.58	0.25	379.9	4.11
Ward (scikit-learn)	59.30	0.043	433.78	4.63		-
Ward (fastcluster)		0.074		1.83	7311	25.06
Single (scikit-learn)	4.92	0.049	13.86	1.36		-
Single (fastcluster)		0.045		0.94	29.9	26.4
Centroid (fastcluster)	8.98	0.083	33.09	1.82	183	30.2
Complete (scikit-learn)	11.8	0.038	33.84	4.53		-
Average (scikit-learn)	9.7	0.037	27.52	4.70		-

As we can see on Table 2, our algorithm **CoreSet** is moderately slower than **CKL** while always producing a significantly better distortion, from 2 to 3 times better. On the largest dataset, SHUTTLE, our algorithm is only $\sim 25\%$ slower.

In order to evaluate further how the algorithms scale on larger datasets, we ran the algorithms over synthetic datasets sampled from a uniform distribution. Table 3 contains the running times for 10^5 points in dimension 100 and for 10^6 points in dimension 50. For these datasets, we did not compute the distortions since evaluating them takes unreasonably long computation times (quadratic time in the number of points when using constant-time LCA queries).

Table 3. Running time overs N random points in dimension d

	$N = 10^5, d = 100$	$N = 10^6, d = 50$
CKL	4.9s	42s
CoreSet	3.8s	58s
Ward (fastcluster)	2141s	$\geq 10h$
Single (fastcluster)	842s	$\geq 10h$
Centroid (fastcluster)	1364s	$\geq 10h$

4.2. Evaluation of the cut-weight algorithm alone

The result of Algorithm 1 is highly dependent on the quality of the γ-approximated MST computed in the first step, that is given by a stochastic algorithm with various possible implementations. In order to compare our new cut-weight estimation alone with existing cut-weight procedures, we measured the distortion given by Algorithm 2 when the exact MST is given on Step 1 instead of the γ-approximated MST. We performed the comparison on classic datasets (IRIS, MICE, PENDIGITS). The result are reported on Table 4. **ExactCoreSet** stands for this modified algorithm, where ε is the approximation parameters for the bounding balls, as used in Algorithm 2. **ExactCKL** stands for the algorithm described in (Cohen-Addad et al., 2020), where the approximated MST has also been replaced by an exact one. **Opt** is the algorithm giving the optimal distortion (Farach et al., 1995).

Table 4. Distortion when minimum spanning trees are used

	IRIS	MICE	PENDIGITS
ExactCKL	11.19	9.14	27.77
ExactCoreSet ($\varepsilon = 1$)	9.33	5.66	16.56
ExactCoreSet ($\varepsilon = 0.2$)	8.49	5.43	14.65
ExactCoreSet ($\varepsilon = 10^{-4}$)	8.49	5.35	14.20
Opt	8.07	4.92	13.86

References

Abboud, A., Cohen-Addad, V., and Houdrouge, H. Subquadratic high-dimensional hierarchical clustering. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 11576–11586, 2019.

Alon, N., Azar, Y., and Vainstein, D. Hierarchical clustering: A 0.585 revenue approximation. In Abernethy, J. D. and Agarwal, S. (eds.), *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pp. 153–162. PMLR, 2020.

Badoiu, M. and Clarkson, K. L. Smaller core-sets for balls. In *SODA*, volume 3, pp. 801–802, 2003.

Badoiu, M., Har-Peled, S., and Indyk, P. Approximate clustering via core-sets. In Reif, J. H. (ed.), *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pp. 250–257. ACM, 2002. doi: 10.1145/509907.509947. URL https://doi.org/10.1145/509907.509947.

Balcan, M.-F., Blum, A., and Vempala, S. A discriminative framework for clustering via similarity functions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 671–680. ACM, 2008.

Carlsson, G. and Mémoli, F. Characterization, stability and convergence of hierarchical clustering methods. *Journal of machine learning research*, 11(Apr):1425–1470, 2010.

Chami, I., Gu, A., Chatziafratis, V., and Ré, C. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

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

Charikar, M., Chatziafratis, V., Niazadeh, R., and Yaroslavtsev, G. Hierarchical clustering for euclidean data. *arXiv preprint arXiv:1812.10582*, 2018.

Charikar, M., Chatziafratis, V., and Niazadeh, R. Hierarchical clustering better than average-linkage. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2291–2304. SIAM, 2019.

Chatziafratis, V., Yaroslavtsev, G., Lee, E., Makarychev, K., Ahmadian, S., Epasto, A., and Mahdian, M. Bisect and conquer: Hierarchical clustering via max-uncut bisection. In Chiappa, S. and Calandra, R. (eds.), *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pp. 3121–3132. PMLR, 2020.

Cochez, M. and Mou, H. 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*, pp. 505–517. ACM, 2015.

Cohen-Addad, V., Kanade, V., and Mallmann-Trenn, F. Hierarchical clustering beyond the worst-case. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 6201–6209, 2017.

Cohen-Addad, V., Kanade, V., Mallmann-Trenn, F., and Mathieu, C. Hierarchical clustering: Objective functions and algorithms. *J. ACM*, 66(4):26:1–26:42, 2019. doi: 10.1145/3321386. URL https://doi.org/10.1145/3321386.

Cohen-Addad, V., S., K. C., and Lagarde, G. On efficient low distortion ultrametric embedding. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2078–2088. PMLR, 13–18 Jul 2020.

Dasgupta, S. A cost function for similarity-based hierarchical clustering. *arXiv preprint arXiv:1510.05043*, 2015.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Farach, M., Kannan, S., and Warnow, T. J. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13 (1/2):155–179, 1995. doi: 10.1007/BF01188585. URL https://doi.org/10.1007/BF01188585.

Fischer, K., Gärtner, B., and Kutz, M. Fast smallest-enclosing-ball computation in high dimensions. In Di Battista, G. and Zwick, U. (eds.), *Algorithms - ESA 2003*, pp. 630–641, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39658-1.

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

Kumar, P., Mitchell, J. S. B., and Yildirim, E. A. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM J. Exp. Algorithmics*, 8, 2003. doi: 10.1145/996546.996548. URL https://doi.org/10.1145/996546.996548.

Moseley, B. and Wang, J. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Advances in Neural Information Processing Systems*, pp. 3094–3103, 2017.

Müllner, D. fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. *Journal of Statistical Software*, 53(9):1–18, 2013. URL http://www.jstatsoft.org/v53/i09/.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Roy, A. and Pokutta, S. Hierarchical clustering via spreading metrics. In *Advances in Neural Information Processing Systems*, pp. 2316–2324, 2016.