Scaling Average-Linkage via Sparse Cluster Embeddings

Thomas Lavastida TLAVASTI@ANDREW.CMU.EDU
Carnegie Mellon University

Kefu Lu KLU@WLU.EDU
Washington and Lee University

Benjamin Moseley MOSELEYB@ANDREW.CMU.EDU
Carnegie Mellon University

Yuyan Wang YUYANW@ANDREW.CMU.EDU
Carnegie Mellon University

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Average-linkage is one of the most popular hierarchical clustering algorithms. It is well known that average-linkage does not scale to large data sets due to the slow asymptotic running time. The fastest known implementation has running time quadratic in the number of data points.

This paper presents a technique that we call cluster embedding. The embedding maps each cluster into a point in slightly higher dimensions. The pairwise distances between the mapped points approximate the average distance between clusters. By utilizing this embedding we scale the task of finding close pairs of clusters, which is a key step in average-linkage clustering. We achieve an approximate, sub-quadratic time implementation of average-linkage. We show theoretically the algorithm proposed in this paper achieves a near-linear running time and scales to large data sets. Moreover, its scalability empirically dominates average-linkage and typically offers 3-10x speed-up on large data sets.

Keywords: Hierarchical Clustering, Average-linkage, Scalable Clustering

1. Introduction

Hierarchical clustering is popular across many scientific disciplines, exemplified by its incorporation in many undergraduate curriculums and its inclusion in various popular machine learning libraries such as *scipy, scikit-learn* and *Spark MLLib* Jones et al. (2001–); Pedregosa et al. (2011); Meng et al. (2016).

Hierarchical clustering's input is a set of n data points with a similarity or dissimilarity function $D(x, y)$ between each pair of points x and y. The output is a *tree* with all n data points as leaves. Each internal node of the tree represents a cluster containing the data points of the leaves of the subtree rooted at the node.

If two data points are relatively similar, their least common ancestor (LCA) should be lower in the tree. If points are dissimilar, then their LCA should be higher in the tree. In this way, clusters corresponding to internal nodes become more refined at lower levels of the tree. Moreover, looking across the tree at any fixed depth will reveal a *partitioned* clustering[1] of all data points at that fixed level of detail.

There are many different types of hierarchical clustering algorithms. See Xu and Wunsch (2005) for an overview. This paper focuses on one of the most popular hierarchical clustering

1. In a partitioned clustering, each data point belongs to one cluster (e.g. the solution produced by k-means).

algorithms, *average-linkage*. This algorithm belongs to the class of agglomerative hierarchical clustering algorithms. Agglomerative methods initializes all data point to be singleton clusters, i.e. the leaves of the final tree. They then iteratively merge the most similar pair of clusters to produce new internal nodes in the hierarchy tree. The entire tree has been formed once all clusters have been merged into one. There are different ways to measure the similarity of pair of clusters at each step. Average-linkage calculates the average distance over all pairs of points across the clusters. The greater this average distance is, the less similar these two clusters are. This average distance is called the distance between the two clusters. Single-linkage and complete-linkage, two other agglomerative algorithms, define the distance between two clusters to be the smallest and greatest distance between all pair of points across the clusters, respectively.

Scalability of Average-Linkage: Average-linkage has scalability issues. Introductory guides on hierarchical clustering often state that average-linkage should only be used on small to modestly sized data sets due to the inherent large running time and the sequential nature of the algorithm. This limits the applicability of hierarchical clustering on large datasets Xu and Wunsch (2005). Therefore, there is interest in making average-linkage more scalable.

The fastest known implementation of average-linkage runs in $\Theta(n^2)$ time Mullner (2013). To compute the exact average-linkage output, this running time is intuitively necessary because the algorithm must inspect all of the $\Omega(n^2)$ pairwise distances between data points to find the first pair to merge.

This quadratic running time becomes quite prohibitive as data sets increase in size. Recently, there has been breakthroughs with improving the scalability of average-linkage. The work of Cochez and Neri (2015) and Abboud et al. (2019) broke through the quadratic barrier by relaxing the algorithm. In both works, the idea is to allow the algorithm to iteratively merge clusters that do not necessarily have the smallest average distance. Cochez and Neri (2015) found a sub-quadratic time algorithm for a restricted class of distance functions that includes Jaccard distance, but excludes all ℓ_p-norm distances. Despite being scalable in implementation, the methods in Cochez and Neri (2015) cannot be directly used for any ℓ_p-norm distances. Abboud et al. (2019) shows that for input with ℓ_1 norm distances, one can also break the quadratic bound by allowing the algorithm to merge clusters with average distance within a factor of $1+\epsilon$ of the average distance between the closest clusters. However, the algorithm design heavily relies on properties of the ℓ_1 norm and the work does not directly apply to other ℓ_p norms[2]. Further, the algorithm emphasizes the near-optimality of every merge performed, leveraging sophisticated data structures to quickly find clusters to merge. The construction and maintenance of such data structures incurs additional computational cost for this adaptation of average-linkage, which is likely to have significant overhead in practice.

Research Challenges: Previous work does not directly give a scalable, easily implementable adaptation of average-linkage for general ℓ_p norms. Nevertheless, it suggests the exciting possibility of developing such an adaptation with new algorithm designs. The new algorithm design should be supported by similar theoretical guarantees, i.e., it should iteratively choose two clusters to merge, whose distance is up to a constant of the smallest average distance among all clusters. In particular, an intriguing open question is to find an efficient adaptation of average-linkage for the ℓ_2-norm distance, the most popular distance function.

Results: This paper designs an algorithm that finds an approximation of average-linkage, which applies to ℓ_p norm distances with $p \in [1, 2]$, with sub-quadratic running time guarantees.

2. Note it is possible to embed ℓ_p norms into ℓ_1 though an additional data processing step which significantly increase the dimension of the data points.

Empirically, we give a scalable implementation with near-linear running time. Our results are as follows.

- The algorithm runs in near linear time. The algorithm requires $\tilde{O}(dn^{1+\rho})$ time and $\tilde{O}(dn^{1+\rho})$ space for arbitrarily small $\rho > 0$ when the input points belong to d-dimensional Euclidean space.[3]
- The tree constructed is a relaxation of average-linkage that ensures that any pair of clusters merged have average distances within a constant factor of the closest cluster pair.
- We perform experiments with our algorithm and compare it to the most popular implementations of average-linkage such as scikit-learn, sci-py, and fastcluster. These represent the fastest average-linkage packages used by data scientists across industry and academia. Our algorithm: (1) Constructs trees that are similar to what standard libraries achieve. In particular, the trees constructed have almost the same value for a popular hierarchical clustering objective. (2) Achieves asymptotically faster running times than the standard algorithms for large data sets, matching the theoretical guarantees. The running time of our implementation is near linear in the input size, as opposed to the quadratic baselines.
- The algorithm is easily parallelizable and can be used in parallel and distributed environments. There is much previous work on scaling clustering methods such as single-linkage (Bateni et al., 2017). However, fundamental differences in the problem prevent these methods from extending to average-linkage. In contrast, most recent work on fast average-linkage clustering, such as Abboud et al. (2019), do not extend easily into parallel and distributed settings.

A New Algorithmic Technique - Cluster Embeddings: This paper introduces a new technique that is of interest beyond average-linkage, cluster embeddings.

Consider a collection of clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. Each cluster is a subset of the input data points. A *cluster embedding* finds k points $v_1, v_2, \ldots, v_k$ where v_i corresponds to C_i. These points exist in some Euclidean space and the distance between v_i and v_j should approximate the average distance between points in C_i and C_j. Formally, the embedding should be low-distortion (Indyk, 2001) with respect to the average ℓ_p distance. Ideally, the points $v_1, v_2, \ldots, v_k$ are located in a space with dimension not much larger than d and the embedding can be computed quickly.

To see the applicability of cluster embeddings, consider a single step of average-linkage. At each step, there is a current collection of clusters $C_1, C_2, \ldots C_k$ and the goal is to find the clusters C_i and C_j that minimizes the average distance between C_i and C_j. That is, clusters whose points are closest on average. Notice that for any pair of clusters it can take quadratic time just to find their average distance depending on the sizes. However, with the embedding in place, we would only need to find the ℓ_p distance between the two embedded points.

The embedding allows us to leverage data mining techniques that apply to points in Euclidean space. Coupling the cluster embedding with *Locality-Sensitive Hashing (LSH)* functions yields a method to quickly find pairs of clusters with small average distance. Such a task is called *near cluster search*. Given a set of points in Euclidean space, LSH functions partitions the points into "buckets", where points in the same bucket are likely to be closer to each other, thus filtering out potential near neighbors without inspecting all pairwise distances between clusters. Such techniques leverage properties of ℓ_p distances in a Euclidean

3. The $\tilde{O}(\cdot)$ notation hides factors of $\log^c(n)$ for constant $c > 0$.

space and cannot be directly used to do near cluster search, since the cluster distance, defined to be the average distance over all pairs, is a more general metric where LSH may not exist. However, an embedding into Euclidean space makes such a technique applicable for near-cluster searches. Iteratively using the embedding coupled with LSH to find clusters that are close gives a new approach to efficiently approximate average-linkage.

Our cluster embedding has the following properties.

- The embedding is *sparse*. Clusters of points in d-dimensional Euclidean space are embedded into vectors with at most $d+1$ non-zero entries. Hence *only one additional non-trivial dimension* is required to store the embedding of each cluster.
- The embedding preserves the average distances well. The ℓ_p-norm distance between v_i and v_j is guaranteed to approximate the average distance between pairs of points in C_i and C_j. In particular, we prove this is bounded in the worst case. Experimentally, they are nearly the same.
- The embedding is *oblivious*, i.e. there is a function $\phi : \mathcal{C} \to \mathbb{R}^{d'}$ such that $v_i = \phi(C_i)$, where the computation of $\phi(C_i)$ depends only on the cluster C_i itself and no other cluster of $\mathcal{C}$. Insertion, deletion, and modification of other clusters doesn't affect $\phi(C_i)$. Here d' is the dimension of the embedded space.
- Constructing the embedding takes *linear time*, $O(nd)$. This is faster than comparing the average similarity between two (large) clusters (*quadratic time*).

Paper Organization: The paper is organized as follows. We begin by introducing preliminary definitions. Then in Section 3, we discuss the technique of cluster embedding - our main technique for speeding up average-linkage. We first showcase the application of cluster embeddings on the problem of near cluster search in Section 4. This is the key step in average-linkage. Afterwards, we give an overview of the average-linkage algorithm in Section 5. This overview gives a skeleton of the key algorithmic ideas. The full algorithm with strong theoretical guarantees is more technical and we defer its full analysis to Appendix A in the supplementary material. In Section 6 we give experimental evaluations. Appendix B in the supplementary material gives the proofs for technical lemmas introduced in Section 3.

Related Work: There has been consistent interest in finding efficient algorithms to cluster data in large databases Guha et al. (2001); Cochez and Neri (2015). Additionally, there has recently been increased interest in scaling hierarchical clustering algorithms Yaroslavtsev and Vadapalli (2018); Bateni et al. (2017); Abboud et al. (2019); Monath et al. (2019); Menon et al. (2019). Kell et al. Kull and Vilo (2008) and Cochez et al. Cochez and Neri (2015) speed-up algorithms by relaxing the criteria on when clusters can be merged. The work of Yaroslavtsev and Vadapalli (2018) and Lattanzi et al. (2019) considered parallelizing hierarchical clustering algorithms in the Massively Parallel Model of computation Karloff et al. (2010); Andoni et al. (2014). Other Spark and MapReduce algorithms have been developed Jin et al. (2015b,a). This prior work on scaling hierarchical clustering focused on other hierarchical clustering algorithms and not average-linkage.

As mentioned, recently there are breakthroughs in designing sub-quadratic adaptations of hierarchical clustering algorithms Cochez and Neri (2015); Abboud et al. (2019) which do not directly lead to a scalable average-linkage implementation for general ℓ_p norms. Method-wise both papers used Locality-Sensitive Hashing (LSH) and other related techniques. The work of Cochez and Neri (2015) was the first to show that one can speed up near cluster search with LSH, but the algorithm design assumes the input distance metric belongs to a restricted

class of distance functions. The work left open the problem of generalizing the ideas to other distances like ℓ_p-norm where LSH is known to exist. Abboud et al. (2019) designed an approximate average-linkage algorithm for the ℓ_1-norm distance. A crucial building block in Abboud et al. (2019) is to leverage properties of ℓ_1 to construct a sophisticated data structure based on LSH functions, where every cluster is represented as a multi-dimensional point. This is similar to our work, except their representation of a cluster increases the dimension by a factor of $\Omega(\frac{1}{\epsilon^6}\log^3 n)$ for constant $\epsilon > 0$ and critically uses properties of ℓ_1 that do not hold for other ℓ_p norms. The original paper does not include an average-linkage implementation. There remains the question of finding a more concise way of embedding clusters into potentially low-dimensional points and designing a practical adaptation of average-linkage based on this embedding.

Embeddings are widely used and there are many types with different properties. See Goodman and O'Rourke (1997) for reference to different types of embedding. Much prior work has focused on low-distortion embeddings Indyk (2001). These embeddings are orthogonal to that considered in this paper. These are used for reducing dimension and ours is used to represent average cluster distances.

This paper uses an objective function for hierarchical clustering Cohen-addad et al. (2019) to evaluate our algorithm. There has been recent interest in developing good objectives for hierarchical clustering Dasgupta (2016); Wang and Moseley (2017). The objective in Cohen-addad et al. (2019) is the most suitable for the setting we are considering - a point set with pairwise distance metric. Average-linkage is a constant approximation algorithm for the objective in Cohen-addad et al. (2019) when pairs of points have distances representing dissimilarity scores and the objective considered in Wang and Moseley (2017) when pairs of points have similarity scores.

2. Preliminaries

Here we define cluster embeddings, average-linkage, near cluster search, hierarchical clustering objective function and Locality-Sensitive Hashing (LSH).

Notation: We use lower case letters e.g. x, y to denote points in Euclidean space and use subscripts to index their coordinates e.g. x_i, y_i. We will use $\|x\|_p := (\sum_i x_i^p)^{1/p}$ to denote the ℓ_p-norm of x. $\|x\|$ will be shorthand for the ℓ_2-norm. For a finite set X in d-dimensional Euclidean space $\mathbb{R}^d$, let $\Delta(X)$ and $\delta(X)$ be the maximum and minimum distances within X, that is, $\Delta(X) := \max_{x,x' \in X} \|x - x'\|$ and $\delta(X) := \min_{x,x' \in X} \|x - x'\|$. If X is a finite set of points let $\mu(X) := \sum_{x \in X} x/|X|$ be the centroid of X and let $\text{gm}(X) := \arg\min_{y \in \mathbb{R}^d} \sum_{x \in X} \|y - x\|/|X|$ be the geometric median of X.

Average-Linkage: Let A, B be disjoint finite sets of points in a metric space with distance $D(\cdot, \cdot)$. The average-linkage between A and B is defined as follows: $\text{Avg}(A, B) := \frac{1}{|A| \cdot |B|} \sum_{x \in A} \sum_{y \in B} D(x, y)$. If $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a collection of pairwise disjoint finite sets of points, then $\text{Avg}(\cdot, \cdot)$ induces a metric on $\mathcal{C}$ where we take $\text{Avg}(C_i, C_i) := 0$. In our setting we have each $C_i \subset \mathbb{R}^d$ and $D(x, y) = \|x - y\|_p$ for some $p \in [1, 2]$.

Cluster Embeddings For Average-Linkage: Let $S \subset \mathbb{R}^d$ be a point set and let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a clustering of S. Let $\phi : \mathcal{C} \to \mathbb{R}^{d'}$ be an embedding of $\mathcal{C}$ into d'-dimensional Euclidean space. We say that ϕ is an embedding of $\mathcal{C}$ into $\ell_p^{d'}$ with *distortion* α if the following holds for all $C_i, C_j \in \mathcal{C}$: $\text{Avg}(C_i, C_j) \leq \|\phi(C_i) - \phi(C_j)\|_p \leq \alpha \cdot \text{Avg}(C_i, C_j)$. Ideally we want to embed into a space of dimension $\approx d$. In lieu of that, we ask for a sparse embedding in a high dimensional space. We say that an embedding ϕ is s-sparse if the number of non-zero entries in $\phi(C_i)$ is at most s for all $C_i \in \mathcal{C}$.

Near Cluster Search: Given a set of clusters $\mathcal{C}$ and a query cluster Q, the near cluster search problem with respect to average-linkage is to find a cluster $C \in \mathcal{C}$ that is the closest to Q in average-linkage. The α-approximate near cluster search problem is to find a cluster $C \in \mathcal{C}$ such that $\text{Avg}(Q, C) \leq \alpha \cdot \min_{C' \in \mathcal{C}} \text{Avg}(Q, C')$.

Approximate Average-linkage: Our algorithms make approximate decisions so we define the notion of approximation here. Sets A, B are α-approximate for clustering $\mathcal{C}$ if $\text{Avg}(A, B) \leq \alpha \cdot \min_{A', B' \in \mathcal{C}} \text{Avg}(A', B')$. An algorithm is α-approximate if it only merges α-approximate clusters.

Global Objectives for Hierarchical Clustering: Recently there has been work on objective functions for hierarchical clustering Dasgupta (2016); Roy and Pokutta (2017); Charikar and Chatziafratis (2017); Cohen-addad et al. (2019); Wang and Moseley (2017); Charikar et al. (2019a,b). We consider the objective defined in Cohen-Addad et al. Cohen-addad et al. (2019) for dissimilarity-based inputs. A slight modification of the proof of Theorem 4.6 in Cohen-addad et al. (2019) shows that any α-approximate algorithm for average linkage is an $O(\alpha)$-approximation for this objective. We will use this objective to empirically assess the quality of the clustering.

The formal definition of the objective in Cohen-addad et al. (2019) is as follows. Let T be a hierarchical clustering tree. For data points x, y let $\text{leaves}(x, y)$ be the number of leaves in the subtree rooted at the least common ancestor of x, y in T. For each internal node of T let $A \to (A_1, A_2)$ be the split into two clusters at that node. Given dissimilarities $D : S \times S \to \mathbb{R}_+$ defined on each pair of input points and a hierarchical clustering tree T, define the objective $\text{rev}(T)$ as follows:

$$\text{rev}(T) := \sum_{A \to (A_1, A_2)} |A| \cdot D(A_1, A_2) = \sum_{x,y \in S} D(x, y) \cdot \text{leaves}(x, y) \tag{1}$$

where $D(A_1, A_2) := \sum_{x \in A_1, y \in A_2} D(x, y)$. The goal is to find a tree T maximizing $\text{rev}(T)$. Solving this maximization problem is NP-hard (Dasgupta, 2016; Cohen-addad et al., 2019), but average-linkage is $\frac{2}{3}$-approximate (Cohen-addad et al., 2019).

Locality Sensitive Hashing (LSH): LSH is used in our algorithm to construct ANN queries of the embedded points. Thus, this will be used to find pairs of clusters to merge. Intuitively, LSH partitions a space into buckets so that nearby points are more likely to lie in the same bucket and far away points are less likely to land in the same bucket. Let X be a metric space with distance function D and $\mathcal{U}$ a universe of buckets. A function family $\mathcal{H} = \{h : X \to \mathcal{U}\}$ is said to be (r_1, r_2, p_1, p_2)-sensitive for D if for any $x, y \in X$ and h drawn uniformly at random from $\mathcal{H}$: $D(x, y) \leq r_1 \implies \Pr_{\mathcal{H}}[h(y) = h(x)] \geq p_1$, and $D(x, y) > r_2 \implies \Pr_{\mathcal{H}}[h(y) = h(x)] \leq p_2$. We require $p_1 > p_2$ and $r_1 < r_2$. A family can also be defined by giving a random process which outputs a function from $\mathcal{H}$. LSH families have been constructed for distances induced by ℓ_p norms for all $p \in (0, 2]$ (Datar et al., 2004).

Technical Assumptions: We assume that $\frac{\Delta(S)}{\delta(S)} = O(\text{poly}(n))$, i.e. the ratio of the maximum to minimum distance is bounded by a polynomial in the number of points. Up to scaling, we may also assume that $\delta(S)$, the minimum distance, is 1.

3. Sparse Cluster Embeddings for Average Distance

In this section we present our embedding of clusters to points in Euclidean space such that the distance between embedded points approximates the average ℓ_p-distance between clusters. The embedding enables the algorithm to approximately calculate the average distance between

any two clusters in $O(d)$ time regardless of their sizes. Later in the discussion on algorithm design, we will show how to use it to facilitate the search for clusters with small average distances. We focus on the case when $p = 2$ and show the following theorem.

Theorem 1 *Given a clustering $\mathcal{C}$ of $S \subset \mathbb{R}^d$, there is an embedding $\phi : \mathcal{C} \to \mathbb{R}^{d'}$ into $\ell_2^{d'}$ with distortion α. We have $d' = d + |\mathcal{C}|$, $\alpha = 5\sqrt{3}$ and ϕ is $(d+1)$-sparse.*

We give a formula which approximates the average-linkage between two clusters A and B. The formula is written in terms of the distance between the centroids of A and B plus two correction terms which only depend on A and B, respectively. This will be used to define the embedding.

Squared Euclidean Distance: We first examine a simpler case to motivate our ideas. Suppose that dissimilarities are given $D(x, y) = \|x - y\|^2$. Thus $\mathrm{Avg}(A, B) = \sum_{x\in A, y\in B} \frac{\|x-y\|^2}{|A|\cdot|B|}$. The goal is to express the average-linkage between clusters A, B as the distance between their centroids plus correction terms. Since $\|\mu(A) - \mu(B)\|^2$ can potentially be small relative to $\mathrm{Avg}(A, B)$, we need the correction terms to capture the rough *variance* in each cluster. For a finite set of points X define $\mathrm{Var}(X) := \sum_{x\in X} \frac{\|x-\mu(X)\|^2}{|X|}$, resembling the definition of "variance" in statistics. The next proposition is the formula whose proof can be found in Appendix B in the supplementary material.

Proposition 2 *For any clusters A and B,* $\mathrm{Avg}(A, B) = \|\mu(A) - \mu(B)\|^2 + \mathrm{Var}(A) + \mathrm{Var}(B)$.

Euclidean Distance: Now we consider dissimilarities given by the Euclidean distance between points. Recall that $\mathrm{Avg}(A, B) = \sum_{x\in A, y\in B} \frac{\|x-y\|}{|A|\cdot|B|}$. We would like a decomposition similar to Proposition 2 expressing $\mathrm{Avg}(A, B)$ as the distance between centroids and some correction terms.

Let $\mathrm{Dev}(X) := \sum_{x\in X} \frac{\|x-\mu(X)\|}{|X|}$ be the average deviation of X from its centroid. It is perhaps intuitive that $f(A, B) := \|\mu(A) - \mu(B)\| + \mathrm{Dev}(A) + \mathrm{Dev}(B)$ should suffice based on the results above. However, there are examples where $f(A, B) \neq \mathrm{Avg}(A, B)$[4]. Instead, we show that the two are always within a constant factor of each other.

Lemma 3 *For any two clusters A and B we have* $\mathrm{Avg}(A, B) \leq f(A, B) \leq 5\,\mathrm{Avg}(A, B)$.

The lower bound $\mathrm{Avg}(A, B) \leq f(A, B)$ is a simple application of the triangle inequality. To show the upper bound we must relate $\mathrm{Dev}(A)$ and $\mathrm{Dev}(B)$ to $\mathrm{Avg}(A, B)$. Recall that for a cluster X, its geometric median $\mathrm{gm}(X)$ minimizes $R_X(y) := \frac{1}{|X|}\sum_{x\in X} \|y - x\|$. The value of $R_X(y)$ is the average distance of points in X to y. We show that for any two clusters A, B, both $R_A(\mathrm{gm}(A))$ and $R_B(\mathrm{gm}(B))$ lower bound $\mathrm{Avg}(A, B)$. Then we show that for any cluster X, $\mathrm{Dev}(X) \leq 2R_X(\mathrm{gm}(X))$. Combining these two facts with $\|\mu(A) - \mu(B)\| \leq \mathrm{Avg}(A, B)$ implies the upper bound. The full proof of this lemma is in Appendix B.

Constructing the Embedding: We now define the embedding and prove Theorem 1. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a clustering of a point set $S \subset \mathbb{R}^d$. We define the embedding $\phi : \mathcal{C} \to \mathbb{R}^{d+|\mathcal{C}|}$ for each $C_i \in \mathcal{C}$ as follows.

$$\phi(C_i) := \sqrt{3}(\mu(C_i), 0, \ldots, \underbrace{\mathrm{Dev}(C_i)}_{d+i\text{'th coordinate}}, \ldots, 0) \tag{2}$$

4. Consider the corners of a rectangle with width w and height h, letting A be the left side points and B the right side points. Then $\mathrm{Avg}(A, B) = (w + \sqrt{w^2 + h^2})/2 \neq w + h = f(A, B)$

In other words we set the first d coordinates to be the centroid of C_i, the $d+i$'th coordinate to be $\mathrm{Dev}(C_i)$ and all other coordinates to be 0. Observe that the embedding is oblivious, meaning that computing $\phi(C_i)$ only depends on the cluster C_i and none of the other clusters. Given this embedding we have that the distance between the embedding of clusters C_i and C_j, $\|\phi(C_i) - \phi(C_j)\|$, is equal to

$$\sqrt{3}\sqrt{\|\mu(C_i) - \mu(C_j)\|^2 + \mathrm{Dev}(C_i)^2 + \mathrm{Dev}(C_j)^2} \tag{3}$$

To finish the proof, we use some easy to verify inequalities.

Proposition 4 *For all $a, b, c \in \mathbb{R}_+$, we have: $(1/\sqrt{3})(a+b+c) \leq \sqrt{a^2+b^2+c^2} \leq a+b+c$*

Proof [Proof of Theorem 1] Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a clustering of S and consider the embedding $\phi : \mathcal{C} \to \mathbb{R}^{d+|\mathcal{C}|}$ defined by (2). The distance between embedded clusters $\|\phi(C_i) - \phi(C_j)\|$ is given by (3). Combining this with the respective lower/upper bounds from Proposition 4 and Lemma 3, we have: $\|\phi(C_i) - \phi(C_j)\| \geq f(C_i, C_j) \geq \mathrm{Avg}(C_i, C_j)$ and $\|\phi(C_i) - \phi(C_j)\| \leq \sqrt{3} f(C_i, C_j) \leq 5\sqrt{3}\,\mathrm{Avg}(C_i, C_j)$. Finally, ϕ being $(d+1)$-sparse follows directly from (2). ■

The main idea for the proof can be easily extended to work for general ℓ_p-norm where $p \in [1, \infty]$, where the distortion will be $5 \cdot 3^{1-1/p}$. See Appendix B for complete arguments.

4. Near Cluster Search

In the section we consider the key subproblem solved in average-linkage to showcase the cluster embedding idea. At each step, average-linkage finds the pair of cluster that are the closest. We abstract this to the following *near cluster search* problem. The goal is to find an approximately nearest cluster in a clustering $\mathcal{C}$ to a query cluster Q with respect to the average distance metric. This subproblem is of its own interest, because data scientists often seek to find pairs of clusters that are very similar.

The naive approach for finding the cluster $C \in \mathcal{C}$ minimizing $\mathrm{Avg}(Q, C)$ requires $\Theta(d|Q| \cdot \sum_i |C_i|)$ time. Our cluster embedding can be used to give an asymptotically faster but approximate algorithm for this task, taking $O(d(|Q| + \sum_i |C_i|))$ time.

Theorem 5 *There exists a data structure supporting $5\sqrt{3}$ approximate near cluster search queries in time $O(d(|\mathcal{C}| + |Q|))$. Construction takes time $O(d(|Q| + \sum_i |C_i|))$.*

Proof Let $\mathcal{C}' = \mathcal{C} \cup \{Q\}$ and let ϕ be the embedding given by Theorem 1 for $\mathcal{C}'$. Start by computing $\phi(C_i)$ for each $C_i \in \mathcal{C}$, taking time $O(d \sum_i |C_i|)$. Note that ϕ is oblivious so this step can be done without knowing Q. The data structure stores the embedded points. To answer a query Q, compute $\phi(Q)$ and then compute the distance from $\phi(Q)$ to $\phi(C_i)$ for each $C_i \in \mathcal{C}$ and return C_i with the smallest embedded distance. Since ϕ has distortion $5\sqrt{3}$ the approximation guarantee follows. Since ϕ is $(d+1)$-sparse, answering the query takes time $O(d|Q| + d|\mathcal{C}|)$. ■

Theorem 5 speeds up near cluster search for any cluster Q enormously, as is shown in Section 6. Further, coupling the embedding with ANN techniques allows us to query Q in *sublinear* time, while only losing an additional constant factor in the approximation.

We consider the LSH family $\mathcal{H}$ defined by Datar et al. (2004). When points are in $\ell_2^{d'}$, i.e. distances are given by the ℓ_2-norm, a hash function $h \in \mathcal{H}$ is constructed by the following procedure. Let $r > 0$ be given. Sample a vector $g \in \mathbb{R}^{d'}$, where each g_i is an i.i.d. standard Gaussian, and a real number b uniformly from $[0, r]$. For a point $x \in \mathbb{R}^{d'}$, its hash is $h(x) = \lfloor (\langle g, x \rangle + b)/r \rfloor$. It's known that this family of hash functions is (R, cR, p_1, p_2)-sensitive for any $c > 1$, where p_1, p_2 depend on c. These hash functions can be combined using well-known techniques to amplify collision probabilities and increase accuracy. See Datar et al. (2004) for details. Here we summarize this result.

Definition 6 (ANN Query Datar et al. (2004)) *Let S be a set of points and D a distance function on S. Given a new point q, the query either returns a point y with $D(q, y) \leq cR$ or reports that there is no point y' with $D(q, y') \leq R$.*

Theorem 7 (Theorem 1 in Datar et al. (2004)) *Let $\mathcal{H}$ be a (R, cR, p_1, p_2)-sensitive family of hash functions for distance function D. Given a set S of n points there exists an algorithm which constructs a data structure L by sampling multiple times from $\mathcal{H}$ which supports (R, c)-ANN queries. The time to construct L and to answer a query from L is dominated by $\tilde{O}(n^{1+\rho})$ distance computations and $\tilde{O}(n^{\rho})$ distance computations, respectively. Here $\rho = \frac{\ln 1/p_1}{\ln 1/p_2} \in (0, 1)$ depends on c. The bigger c is, the closer ρ is to 0.*

Using this, we can construct the point set S' consisting of the embedded points $\phi(C_i)$ for all $C_i \in \mathcal{C}$. We apply the above to obtain a data structure L supporting efficient approximate near neighbor search queries in the embedded space. The sparse nature of the embedding also implies that L can be computed very efficiently in practice. When given query Q, we compute $\phi(Q)$ and then apply the algorithm from Theorem 7 to process the query. We get the following result.

Theorem 8 *For any constant $\epsilon > 0$, there exists a data structure which supports $5\sqrt{3}c$-approximate near cluster search queries in time $\tilde{O}(d(|C|^{\rho} + |Q|))$. Constructing the data structure takes time $\tilde{O}(d(\sum_i |C_i| + |\mathcal{C}|^{1+\rho}))$. Here $\rho \in (0, 1)$ is a constant depending on c.*

5. Fast Approximate Average-Linkage

The main application of our cluster embedding is to get fast approximate algorithms for average-linkage. In this section we give the main structure for our fast implementation of average-linkage that highlights the key ideas. The idea is to embed the current set of clusters into points and then partition the embedded points into buckets (sets) using LSH. The number of buckets is not fixed. We can use the hash family described in Section 4 for this.

Say we want to merge all pairs of clusters with average distance approximately δ. The algorithm draws hash function h randomly from $\mathcal{H}$, where $\mathcal{H}$ is the LSH family given in Datar et al. (2004), and applies it to $\phi(A)$ for each cluster $A \in \mathcal{C}$. Let the i'th bucket be $B_i = \{A \in \mathcal{C} \mid h(A) = i\}$[5]. Suppose bucket B_i is small: $|B_i| \leq O(n^{\rho})$ for some $\rho \in (0, 1)$. We run vanilla average-linkage inside B_i to the point where there is no pair of clusters with average distance smaller than $(1 + \epsilon)\delta$. Here we crucially treat each cluster in a bucket B_i as a single point via the cluster embedding, which allows us to replace computing the average-linkage between pairs of clusters with a distance computation in the embedding.

5. We use $h(A)$ as a shorthand for $h(\phi(A))$ since we will always be implicitly working in the embedded space

This takes $\tilde{O}(dn^{2\rho})$ time. Moreover, assuming all buckets are small, the cost over all buckets is $\tilde{O}(dn^{1+\rho})$. We call this the "local merging" step.

Once all buckets have completed their "local merging" step, the algorithm repeats the above steps with the given value of δ for $O(\log n)$ iterations. Provably, no two clusters will have average-linkage $\leq (1+\epsilon)\delta$ with high probability. At this point the algorithm recurses with $\delta \leftarrow (1+\epsilon)\delta$. The number of such iterations is bounded by the number of δ's considered between the minimum and maximum possible distances, which is $O(\log n)$ by our technical assumption. See Algorithm 1 for pseudocode. Given the above discussion, we see that the running time should be $\tilde{O}(dn^{1+\rho})$, near linear time for small constant ρ.

Algorithm 1 Fast Approximate Average-Linkage Algorithm

```
FastAverageLinkage(S, α, ε)
C ← {{x} | x ∈ S} {Make leaf clusters}
for k = 1, 2, ..., Θ(log n) do
    δ ← (1 + ε)^{k−1}
    C_final ← ∅ {Tracks clusters that won't get merged again for this value of δ}
    for t = 1, 2, ..., Θ(log n) do
        h ← random hash function from H
        Compute h(A) for each A ∈ C
        Let B_i = {A ∈ C | h(A) = i}
        if |B_i| ≤ O(n^ρ) for all i then
            for each B_i do
                Run vanilla average-linkage on B_i until no merges w/ average-linkage ≤ α(1+ε)δ
            end for
            Update the embeddings
        else
            C, C_final ← RobustMerging(C, C_final, δ, α, ε)
        end if
    end for
    C ← C_final
end for
Return the resulting tree
```

There are two unaddressed challenges. First, what should be done when there are buckets B_i with $|B_i| > n^\rho$. Second, maintaining the embedding can be expensive if done in a naive manner. We briefly address these challenges here, but note that a variant of the above procedure works in experimentation.

Challenge 1 - Large Buckets: In the worst case, it is possible that LSH can produce buckets of size larger than n^ρ, resulting in a poor running time for the local merges. Notably, empirically most buckets have small sizes after LSH. For the few buckets whose sizes are large, we use a more robust merging procedure that uses the ANN search method mentioned in Section 4. This procedure is described in Appendix A. For any cluster, this technique only checks $\tilde{O}(n^\rho)$ clusters that got hashed into the same bucket and gives theoretical performance guarantees.

Challenge 2 - Updating the Embedding: If one wants to update the embedding after a single merge, this requires computing the distance of each point to the new centroid. This is potentially a complex and expensive operation we wish to avoid. To do so we carefully make additional merges before updating the embedding and applying LSH again. This is

	Blog		Shuttle		Covertype		Higgs (L)		Susy (L)	
k	Embed	Base	Embed	Base	Embed	Base	Embed	Base	Embed	Base
128	0.17±0.003	25.1 ±28.2	0.09±0.002	34.2±44.7	0.11±0.001	32.5±9.1	0.11±0.001	22.3±5.4	0.10±0.001	24.5±10.5
256	0.17±0.002	9.5±9.7	0.09±0.001	18.6±17.8	0.11±0.002	10.5±4.5	0.11±0.001	11.5±2.2	0.10±0.002	11.2±4.2
512	0.17±0.005	14.2±33.4	0.09±0.002	6.3±6.9	0.12±0.004	6.3±2.4	0.11±0.002	4.8±1.9	0.10±0.002	4.9±3.3
1024	0.18±0.01	3.2±3.4	0.09±0.002	2.0±1.8	0.12± 0.002	2.9±1.2	0.12±0.002	3.3±0.7	0.11±0.002	2.9±1.0
2048	0.18±0.02	1.8±2.8	0.11±0.012	0.79±0.5	0.13±0.02	1.3±0.6	0.13±0.01	1.4±0.5	0.11±0.002	1.0±0.6
4096	0.19±0.003	0.82±0.3	0.12±0.01	0.45±0.21	0.14±0.003	0.62±0.3	0.15±0.01	0.53±0.2	0.13±0.002	0.52±0.4

Table 1: Run time results for Near Cluster Search in seconds. We report the results as $\mu \pm \sigma$, where μ is the average run time at each value of k and σ is the standard deviation. Note that our technique consistently has a small run time with minuscule variance compared to the highly variable baseline.

Dataset	Approx. ratio	Embedding ratio
Shuttle	1.003	1.072
Blog	1.009	1.053
Covertype	1.022	1.059
Susy (L)	1.019	1.032
Higgs (L)	1.012	1.016

Table 2: Approximation Results for Cluster Search. Note that these ratios tend to be very close to 1, indicating the accuracy of our algorithm.

Algorithm	Blog 32768	Blog 52397	Shuttle 32768	Shuttle 43500
scipy	221.76	732.43	125.27	307.43
sklearn	247.88	754.16	127.55	305.45
fastcluster	245.49	689.19	96.43	228.02
Our method	129.88	205.70	48.98	78.76

Table 3: Run time results for average-linkage clustering. Each column is labeled by the data set and input size considered. Our algorithm consistently sees a speed-up at moderate input sizes over all baselines tested.

done to ensure that enough clusters have been involved in a merge so that the total cost due to updating the embedding is small. The robust merging procedure alluded to in Algorithm 1 is also designed to handle this. See Appendix A for details. In practice, we use reservoir sampling to reduce the cost of updating the embedding by approximating it using a set of $O(\log n)$ samples maintained during the merges.

6. Experiments

We evaluate the performance of the cluster embedding for two applications: near cluster search and average-linkage hierarchical clustering. The goal is to establish the following:

- Our algorithms which use cluster embeddings are scalable for both applications. Their running time drastically outperforms standard implementations on large data sets.
- The running time of our average linkage implementation is only slightly super-linear in the input size.
- Distances in the embedding closely approximate the average distance for all data sets.
- Our algorithm for near cluster search returns a cluster with approximately the smallest average distance. In fact, it returns the exact best cluster at least 90% of the time.
- Our average-linkage hierarchical clustering algorithm closely approximates the standard average-linkage algorithm. The vast majority of the merges made by the algorithm have average distances below a factor 1.5 of the minimum average distance between clusters.
- The objective value of trees produced by our new average-linkage algorithm has negligible loss compared to baseline algorithms.

Experiment Specifications: We implemented our algorithms in Python and used 3 baseline implementations of average linkage available in the SciPy, Scikit-Learn, and Fastclustering libraries. Experiments utilized Google Cloud Platform virtual machines (VMs), specifically `n1-highmem-4` type VMs, each with 4 virtual CPUs and 26 GB of memory.

We further compare to the work of Abboud et al. (2019). For this comparison, we used the ℓ_1-norm distance, a requirement of their algorithm. Our method and the baselines were superior in all experiments. While their algorithm has near linear run time like ours, their method runs in time $O(\frac{1}{\epsilon^6} n^{1+\rho} \log^3 n)$. The $\frac{1}{\epsilon^6} \log^3 n$ factor makes this algorithm slow in practice. See Table 10 in Appendix C.

Datasets: We use datasets from the UCI ML repository Lichman (2013) : Shuttle, Blog, Covertype, Susy, and Higgs. Number of dimensions range from 8 to 281. Datasets are listed in Table 4. Some datasets contained extra features which are a function of the other features. These were preprocessed to create two datasets: "Large" (L) with all features and "Small" (S) without the extra features. If the dataset contained a class column for supervised learning, that column was removed. Euclidean distance was used as the dissimilarity between datapoints.

Implementation Details: For near cluster search, we implemented the algorithm based on Theorem 5. Our implementation of average-linkage is similar to Algorithm 1 but include a few minor changes for practical efficiency.

Merging procedure for large buckets: in practice, if the partition induced by a round of LSH has a large bucket with more than $\sqrt{n}$ points, we split it into several small buckets of size at most $\sqrt{n}$. Then we run average linkage within each small bucket.

Dataset	Number of Points	Number of Features	Pre-processed?
Shuttle	43500	8	N
Blog	600021	281	N
Covertype	581012	54	Y
Susy (L/S)	5000000	18/8	Y
Higgs (L/S)	11000000	28/21	Y

Table 4: Data sets tested

Reservoir sampling to approximate deviation terms: we estimate the deviation $\text{Dev}(A)$ term in the embedding $\phi(A)$ by sampling a small set of points uniformly from the points in this cluster: if x is a uniformly random point from A, then $\mathbb{E}[\|x - \mu(A)\|] = \text{Dev}(A)$. A set of $\Theta(\log n)$ samples is maintained for every cluster, and updated using *reservoir sampling* when merging two clusters. This makes it easy maintain estimates of the deviation term.

Algorithm	Covertype 65536	Covertype 262144	Higgs (L) 65536	Higgs (L) 262144	Higgs (S) 65536	Susy (L) 65536	Susy (L) 262144	Susy (S) 65536
fastcluster	584.06	-	667.10	-	673.86	649.74	-	610.90
our method	465.15	3220.69	233.80	1684.17	230.89	112.87	799.91	63.82

Table 5: Running time results in seconds for average-linkage clustering. Columns indicate data set and input size considered. Missing entries indicate a failure due to a memory error on this input size. All other baseline algorithms fail on these large sizes.

Near Cluster Search: To test our near cluster search algorithm we considered the following setup. First, for each dataset we constructed a clustering using a subsample 50k points[6]. To get a collection of clusters and a query, we ran $(k+1)$-means and took one of the clusters to be the query cluster Q and the remaining k clusters to be the clustering $\mathcal{C}$. For our algorithm we measured running time of constructing the embedding plus answering the query. This is compared with the running time of the baseline method which answers a query by computing the average distance of Q to each cluster. We compared the quality of the cluster returned by our algorithm to the cluster returned by the baseline algorithm. Finally, we compared

6. The Shuttle dataset was used in its entirety as it has less than 50k points

Data Set	Average	95-percentile	Max
Shuttle	1.079	1.268	1.894
Blog	1.070	1.238	2.515
Covertype	1.081	1.330	3.535
Susy (L)	1.143	1.481	2.698
Susy (S)	1.132	1.444	2.465
Higgs (L)	1.194	1.652	8.260
Higgs (S)	1.188	1.632	9.859

Table 6: Embedding ratio stats, with Dev$(\cdot)$ approximated with reservoir sampling, at sample size 16384

Data Sizes	64	128	256	512	1024	2048	4096	8192	16384	32768	43500
Shuttle	99.79	99.61	99.78	99.67	99.63	99.76	99.81	99.79	99.62	99.67	99.79
Blog	92.74	95.72	95.84	96.96	97.88	98.32	99.19	99.28	99.59	99.74	/
Covertype	89.67	89.95	92.58	94.52	95.56	96.72	97.65	98.63	99.48	99.57	/
Susy (L)	98.98	98.92	99.26	99.11	98.46	98.47	98.56	98.56	98.32	98.08	/
Susy (S)	98.81	99.23	99.26	98.80	99.00	98.65	98.69	98.41	98.56	98.41	/
Higgs (L)	98.61	98.36	98.32	97.67	97.69	97.78	97.65	97.75	97.21	97.37	/
Higgs (S)	98.21	97.99	98.18	98.04	98.00	97.91	97.57	97.56	97.73	97.48	/

Table 7: Mean objective approximation ratio of average-linkage, in percentage

the embedded distance of the returned cluster to the true average distance. The parameter k was ranged in powers of 2 from 128 to 4096, and 10 trials were done for each value of k.

The running times of cluster search are in Table 1. Our algorithm utilizing the embedding outperforms the baseline method in both average run time and variance. The run time of the naive method varies drastically across different instances, being quadratically dependent on the size of the query cluster and the number of points. Our method has very stable run times, consistent with the theoretical justification, as the run time only depends linearly on the size of the query cluster and the number of given clusters.

We display the accuracy of our near cluster search algorithm in Table 2. Here, *Approx. ratio* refers to the ratio of the average-linkage distance of the returned cluster for each query to that of the actual closest cluster. *Embedding ratio* is the ratio, for the returned cluster, of its distance from the query cluster in the embedding compared to their actual average distance. We report the *worst* such ratio encountered in all instances we tested. Our method has an error rate of about 2% for all data sets, and we observe less than 10% difference in the embedded distances. Near cluster search with the embedding yields very accurate results, much better than the theoretical bounds.

Average-linkage Hierarchical Clustering: Running time wise, our average-linkage algorithm is faster than all the baseline algorithms on large datasets. See Tables 3 and 5 for details. All baseline algorithms also begin to fail once sizes reach 32k to 64k points due to memory requirements. Recall that they require $\Omega(n^2)$ memory for memoization, whereas we need slightly super-linear memory.

Figure 1 compares the growth rate of running time versus input size for our implementation and the three baselines. Both running time and input size are plotted on a logarithmic scale. Notice that any polynomial function $y = cx^\rho$ becomes a linear function on the log-log plot, thus the *slope* of a curve equals the exponent ρ. Our algorithm has a slope that is close to 1, showing it to have near linear running time. The three baselines share a similar growth rate.

Dataset	Mean	95^{th} percentile	Max
Shuttle	1.008	1.031	1.090
Blog	1.008	1.032	1.123
Covertype	1.008	1.032	1.101
Susy (L)	1.005	1.019	1.088
Susy (S)	1.006	1.020	1.075
Higgs (L)	1.003	1.012	1.058
Higgs (S)	1.003	1.012	1.077

Table 8: Embedding ratio stats using true Dev$(\cdot)$ term at size 16384

Dataset	Mean	95^{th} percentile	Max
Shuttle	1.13	1.33	1.58
Blog	1.30	1.85	4.12
Covertype	1.56	2.19	3.65
Susy (L)	1.28	1.61	2.02
Susy (S)	1.25	1.52	1.87
Higgs (L)	1.33	1.65	2.01
Higgs (S)	1.34	1.66	1.99

Table 9: Closeness ratio stats, size 1024

The curve "FastCluster Regression Model" is a linear regression fitted to the running times of FastCluster, the fastest of the baselines. We report the fitted slope to be near 2, showing all three baselines to have quadratic running time. Figure 1 focused on Susy (L) but the results are similar for all datasets.

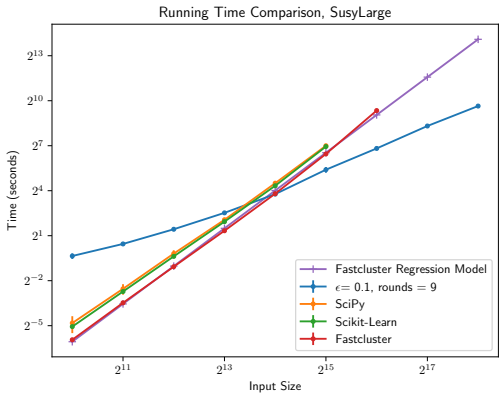

Figure 1: Comparison of how running time grows with input size, both axis in log scale, on data set Susy(L)

The embedding gives very accurate estimates of the average distance between clusters. See Table 8 for detailed statistics on the embedding ratio. Further, Table 6 shows negligible loss in using sampling to approximate the deviation term in the cluster embedding (2).

Closeness ratio: to test accuracy, in each step of the algorithm, we measure how well the merge our algorithm makes approximate the current minimum average-linkage. The approximation ratio is the ratio of the average distance of the chosen pair of clusters to the minimum average distance. To measure this, we find the minimum average distance δ^* only at the start of each round of LSH. Then, for each merge of A and B in this round we compute $\text{Avg}(A, B)/\delta^*$, giving an *upper bound* on the approximation ratio, named the closeness ratio.

Table 9 includes more detailed statistics on all subsamples of size 1024. The merges picked by our algorithm are close to the minimum.

We compare our implementation and the standard implementation of average-linkage on the objective in Cohen-addad et al. (2019). See Table 7. We observe a negligible loss in overall tree quality compared to vanilla average-linkage.

7. Conclusions and Future Work

This paper introduces a sparse cluster embedding that approximately preserves the average distance between clusters. The embedding can be computed in linear time and, after the embedding, pairwise average cluster distances can be computed in $O(d)$ time.

The embedding enables a sub-quadratic average-linkage algorithm. In experiments, the algorithm scales better than current popular implementations of average-linkage.

An interesting direction for future work is to explore cluster embeddings further and determine whether they can be leveraged in other data mining tasks.

References

Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrouge. Subquadratic high-dimensional hierarchical clustering. In *NeurIPS*, pages 11576–11586, 2019.

A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, 2014.

MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *NeurIPS*, 2017.

Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *SODA*, 2017.

Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *SODA*, 2019a.

Moses Charikar, Vaggos Chatziafratis, Rad Niazadeh, and Grigory Yaroslavtsev. Hierarchical clustering for euclidean data. In *AISTATS*, 2019b.

Michael Cochez and Ferrante Neri. Scalable hierarchical clustering: Twister tries with a posteriori trie elimination. In *SSCI*, pages 756–763, 2015.

Vincent Cohen-addad, Varun Kanade, Frederik Mallmann-trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *J. ACM*, 66(4), 2019.

Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *STOC*, 2016.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 253–262, 2004. doi: 10.1145/997817.997857. URL http://doi.acm.org/10.1145/997817.997857.

Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., USA, 1997. ISBN 0849385245.

Google Cloud Platform. https://cloud.google.com/.

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001. doi: 10.1016/S0306-4379(01)00008-4. URL https://doi.org/10.1016/S0306-4379(01)00008-4.

P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS*, 2001.

Chen Jin, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. Incremental, distributed single-linkage hierarchical clustering algorithm using mapreduce. In *HPC*, 2015a.

Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. A scalable hierarchical clustering algorithm using spark. In *Big Data Computing Service and Applications*, 2015b.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online].

Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, 2010.

Meelis Kull and Jaak Vilo. Fast approximate hierarchical clustering using similarity heuristics. *BioData Mining*, 1, 2008. doi: 10.1186/1756-0381-1-9.

Silvio Lattanzi, Thomas Lavastida, Kefu Lu, and Benjamin Moseley. A framework for parallelizing hierarchical clustering methods. In *ECML*, 2019.

M. Lichman. UCI ml repository, 2013. URL http://archive.ics.uci.edu/ml.

X. Meng, J. Bradley, B. Yavuz, Evan Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, 17(1):1235–1241, January 2016. ISSN 1532-4435.

A. Krishna Menon, A. Rajagopalan, B. Sumengen, G. Citovsky, Q. Cao, and S. Kumar. Online hierarchical clustering approximations. *CoRR*, abs/1909.09667, 2019.

N. Monath, A. Kobren, A. Krishnamurthy, M. R. Glass, and A. McCallum. Scalable hierarchical clustering with tree grafting. In *SIGKDD*, 2019.

Daniel Mullner. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 53(9), 2013.

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

Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. *J. Mach. Learn. Res.*, 18:88:1–88:35, 2017. URL http://jmlr.org/papers/v18/17-081.html.

Joshua Wang and Benjamin Moseley. Approximation bounds for hierarchical clustering: Average-linkage, bisecting k-means, and local search. In *NIPS*, 2017.

Rui Xu and Donald C. Wunsch. Survey of clustering algorithms. *IEEE Trans. Neural Networks*, 16(3):645–678, 2005. doi: 10.1109/TNN.2005.845141. URL https://doi.org/10.1109/TNN.2005.845141.

Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under lp distances. In *ICML*, 2018.

Appendix

Appendix A. Analysis of Fast Average-Linkage

In this section we show the robust merging procedure referred to in Algorithm 1 and the algorithm analysis. The purpose of this algorithm and analysis is to extend the algorithm in Section 5 to ensure that the algorithm runs in near-linear time in the worst-case and approximates average-linkage.

In Algorithm 2, $\alpha = 5\sqrt{3}$ is the distortion factor in Theorem 1. The goal of this section is to show the following theorem.

Theorem 9 *With high probability, Algorithm 1 along with Algorithm 2 is $O(1)$-approximate for average-linkage and runs in time $\tilde{O}(dn^{1+\rho})$ for some constant $\rho \in (0, 1)$.*

The following lemma will be quite useful in the analysis and will guide the development of the merging procedure. The proof is easy and follows from a simple averaging argument.

Lemma 10 *For a cluster A, if $\text{Avg}(A, B) \geq \delta$ and $\text{Avg}(A, C) \geq \delta$, then $\text{Avg}(A, B \cup C) \geq \delta$.*

In other words this lemma states that if after some merge the minimum average distance between clusters is at least δ, we are guaranteed that it will not fall below δ during any future merges. Intuitively, we want to merge clusters an increase δ over time. This lemma ensures that we do not undo progress.

A.1. The Robust Merging Procedure

Recall that one of the main challenges discussed is that LSH may produce large buckets. To handle this, we employ the Approximate Near Neighbor (ANN) search technique from Datar et al. (2004). Recall Definition 6 and Theorem 7 from earlier in the paper. We will use these tools in our analysis.

The robust merging procedure leverages (R, c)-ANN queries in the following way. First, we embed the clusters into points. By Theorem 1, this preserves the average-linkage up to the constant distortion factor α. We then build the data structure referenced above and use it to query every cluster.

The robust merging procedure is given in Algorithm 2. This can be described as follows. We say two clusters are "neighbors", if querying one of them returns the other. For each $A \in \mathcal{C}$, let $Q(A)$ denote the set of its neighbor clusters. If $Q(A) = \emptyset$, with high probability A has average distance of at least $(1 + \epsilon)\delta$ from any other cluster. We remove A from $\mathcal{C}$ and add it to C_{final}. Otherwise, if set $Q(A)$ has valid clusters that are not yet merged with any other cluster, we iteratively merge A with these clusters in any arbitrary order. However, if set $Q(A)$ is non-empty but all clusters are already merged with their other neighbors, we pick any neighbor from $Q(A)$ and merge A with the current cluster containing it.

We give intuition for why the algorithm is approximate and has good running time. The formal proof is given in the next section.

Intuition for Proving the Algorithm is Approximate: The following lemma asserts that every merge done by Algorithm 2 is approximate. Intuitively, this is true because all the clusters that are merged into one in Algorithm 2 would have a common "center" cluster that is close to all of them.

Lemma 11 *Every cluster $A \in \mathcal{C}$ such that $Q(A)$ is non-empty is merged with some other clusters, and every merge done by Algorithm 2 has average distance at most $4c\alpha(1 + \epsilon)\delta$.*

Algorithm 2 Pseudocode for Robust Merging Procedure

$L \leftarrow (\alpha(1+\epsilon)\delta, c)$-ANN data structure from Theorem 7 {Construct it for point set $\{\phi(A) \mid A \in \mathcal{C}\}$}
for $A \in \mathcal{C}$ **do**
  $Q(A) \leftarrow \emptyset$ {Will be set of discovered "neighbors" of A}
end for
for $A \in \mathcal{C}$ **do**
  $B \leftarrow$ result of querying $\phi(A)$ in L
  if B is a cluster **then**
    $Q(B) \leftarrow Q(B) \cup \{A\}$, $Q(A) \leftarrow Q(A) \cup \{B\}$
  end if
end for
for $A \in \mathcal{C}$ such that $Q(A) = \emptyset$ **do**
  Drop A from $\mathcal{C}$ and add it to $\mathcal{C}_{final}$. Delete $\phi(A)$ from L. {A won't be considered for merging again at the current δ}
end for
for $A \in \mathcal{C}$ such that $Q(A)$ is not empty **do**
  if $Q(A)$ has clusters that are not yet merged with any other cluster **then**
    Merge all unmerged clusters in $Q(A) \cup \{A\}$ with A in any order
  else
    Let B be arbitrary cluster in $Q(A)$
    Merge A with the cluster containing B
  end if
end for
Remove all the invalid clusters from $\mathcal{C}$ and add all the new clusters into $\mathcal{C}$
Return $\mathcal{C}, \mathcal{C}_{final}$

Proof In any iteration of robust merging, a cluster A, if not merged with other clusters yet, is merged with its unmerged neighbors in $Q(A)$, or it gets added to a set of clusters which are already merged. We call the first type of cluster "core" clusters.

Pick any core cluster A in $\mathcal{C}$ at the beginning of Algorithm 2. Any other cluster in $\mathcal{C}$, if not put into $\mathcal{C}_{final}$, must be merged with one such cluster. We show that by the end of robust merging, for every cluster B that is merged with core A, $\text{Avg}(A, B) \leq 2c\alpha(1+\epsilon)\delta$.

Pick any cluster B that is in the same cluster with A at the end of Algorithm 2. By Theorem 1, we have $\text{Avg}(A, B) \leq \|\phi(A) - \phi(B)\|$. If $B \in Q(A)$, we have $\|\phi(A) - \phi(B)\| \leq c\alpha(1+\epsilon)\delta$ since B is returned by doing $(\alpha(1+\epsilon)\delta, c)$-NN query on A. Otherwise, it must be the case that some other cluster $C \in Q(B)$ is in the same cluster as A. If $C \notin Q(A)$, when robust merging comes to C, it would have merged C with B since $B \in Q(C)$ and B hasn't been merged with any cluster yet, contradicting the assumption that B is merged with A. Therefore $C \in Q(A)$, and $\text{Avg}(A, B) \leq \|\phi(A) - \phi(B)\| \leq \|\phi(A) - \phi(C)\| + \|\phi(C) - \phi(B)\| \leq 2c\alpha(1+\epsilon)\delta$. Now by triangle inequality any two clusters marked with A has average-linkage at most $4c\alpha(1+\epsilon)\delta$. ■

This is the key lemma to show that the algorithm is approximate. It highlights the reason the algorithm is correct. To complete the analysis one has to show that the minimum average distance is close to δ this will be done in the next sections.

Intuition for Proving the Algorithm has Efficient Run Time: The goal is to show that in every round of robust merging makes reasonable progress. In particular, in the total number of clusters remaining in $\mathcal{C}$ shrinks by a constant factor. For fixed δ, every time robust merging is executed, any cluster in $\mathcal{C}$ is either put into $\mathcal{C}_{final}$ or merged with another cluster. Therefore, $|\mathcal{C}|$ shrinks by at least $\frac{1}{2}$. Thus for the same threshold δ, after $\Theta(\log n)$ rounds of robust merging there will be no pairs of (A, B) such that $\text{Avg}(A, B) \leq (1+\epsilon)\delta$. This is formalized in the next section.

A.2. Running Time Analysis for Algorithm

We now complete the analysis of Algorithm 1. We start by analyzing the run time. First we give a lemma showing there exists an estimation $f(A, B) \approx \text{Avg}(A, B)$ update formula which preserves the approximation ratio for $\text{Avg}(A, B)$.

Lemma 12 *Let $f(A, B)$ be estimates such that $\text{Avg}(A, B) \leq f(A, B) \leq \alpha \,\text{Avg}(A, B)$ for all $A, B \in \mathcal{C}$. Suppose we merge $A, A' \in \mathcal{C}$ and set $f(A \cup A', B) = \frac{|A|}{|A|+|A'|} f(A, B) + \frac{|A'|}{|A|+|A'|} f(A', B)$, then $\text{Avg}(A \cup A', B) \leq f(A \cup A', B) \leq \alpha \,\text{Avg}(A \cup A', B)$ for all $B \in \mathcal{C} \setminus \{A, A'\}$.*

Lemma 12 follows easily by an averaging argument using $\text{Avg}(A \cup A', B) = \frac{|A|\,\text{Avg}(A,B) + |A'|\,\text{Avg}(A',B)}{|A|+|A'|}$. This implies that we can simulate vanilla average-linkage within each bucket in Algorithm 1, using the update formula given in this lemma to simulate the updates in average-linkage. Combining this with current average-linkage implementations, we can approximately simulate the vanilla average linkage algorithm in bucket B_i in time $\tilde{O}(|B_i|^2)$, until there are no pairs of clusters (A, B) such that $f(A, B) \leq \alpha(1+\epsilon)\delta$. Thus, there are no pairs of (A, B) such that $\text{Avg}(A, B) \leq (1+\epsilon)\delta$.

We will now show the run time.

Theorem 13 *Algorithm 1 has a run time of $\tilde{O}(dn^{1+\rho})$ for some constant $\rho \in (0, 1)$.*

Proof Let $\rho = \ln(1/p_1)/\ln(1/p_2) \in (0, 1)$, where p_1, p_2 are the probabilities for the LSH family in Datar et al. (2004). Note that it is possible to take p_1, p_2 as constants. Since the $\tilde{O}(\cdot)$ notation hides factors of $O(\log(n))$, it suffices to analyze the cost of the work done in the innermost loop. This is because by definition of the algorithm, the steps in the innermost loop are repeated at most $O(\log^2 n)$ times. We first show that trivially, the time spent on updating the embedding $\phi(\cdot)$ is $\tilde{O}(n)$. This comes from the fact that updating centroid takes $O(1)$ time every merge, and there are $n-1$ merges in total. For deviation term, we do $\log^2(n)$ rounds of merging. During every round of merging, every point in the original cluster S is calculated in the deviation terms only once, thus giving $\tilde{O}(n)$ time spent on updating. Thus we want to bound the cost of running a vanilla algorithm on each bucket when all bucket sizes are bounded by $O(n^\rho)$ and the cost of running the robust merging procedure.

Suppose that we have buckets B_i with $|B_i| \leq O(n^\rho)$ for all i. Note that $\sum_i |B_i| = |\mathcal{C}|$ since the buckets partition the current set of clusters Then by Lemma 12 running a vanilla average-linkage algorithm on each bucket costs at most $O(|B_i|^2)$ per bucket. Thus the overall run time cost is $O(\sum_i |B_i|^2) = O(n^\rho \sum_i |B_i|) = O(n^\rho |\mathcal{C}|) = O(n^{1+\rho})$ since $|\mathcal{C}| \leq n$.

Now consider the cost of running the robust merging procedure. By Theorems 1 and 7 the data structure L can be constructed in time $\tilde{O}(|\mathcal{C}|^{1+\rho}) = \tilde{O}(n^{1+\rho})$. Again by Theorem 7, querying each cluster takes time $\tilde{O}(|\mathcal{C}|^{1+\rho}) = \tilde{O}(n^{1+\rho})$ distance computations in total. Finally,

the merging step takes time $O(|\mathcal{C}|) = O(n)$, yielding an overall cost of $\tilde{O}(n^{1+\rho})$ for the robust merging procedure. ■

A.3. Approximation and Correctness

Now we move to analyzing the correctness of our algorithm. This encompasses two different claims, one concerning how well the algorithm approximates average-linkage and another showing that the algorithm always produces a valid hierarchical clustering tree. The second claim is important since it is not immediately obvious from the definition of Algorithm 1. In particular we need to show that the algorithm will perform enough merges and does not terminate before a tree is completed. The following theorems formalize these claims.

Theorem 14 *Assuming the minimum pairwise distance to be lower-bounded by* 1*, Algorithm 1 is* $O(1)$*-approximate for average-linkage with high probability.*

Theorem 15 *With high probability, Algorithm 1 returns a valid hierarchical clustering tree upon termination.*

Theorem 14 establishes the fact that Algorithm 1 is an $O(1)$-approximation for average-linkage. Note that Lemma 11 only establishes this for the robust merging procedure, so to complete this analysis we need to look at Algorithm 1, which uses the robust merging procedure. The proof of Theorem 15 is also tied to the proof of Theorem 14, so we will do the analysis for both here.

To prove the above theorems we make use of the following key lemmas. These guarantee that for each threshold value δ, by the end of the $O(\log n)$ rounds of merging, there are no good pairs left to merge for the current threshold, and every merge we do has average distance close to $(1+\epsilon)\delta$.

Lemma 16 *For threshold* δ *after merging with high probability, there are no pair of clusters with distance at most* $(1+\epsilon)\delta$ *left.*

Lemma 17 *For a threshold value* δ*, every merge done by Algorithm 1 has average distance at most* $4c\alpha(1+\epsilon)\delta$*.*

Before proving Lemmas 16 and 17, we first show that they directly give us Theorems 14 and 15. To simplify the language in our arguments, we define $(1+\epsilon)\delta$-good clusters as follows.

Definition 18 *During any stage of Algorithm 1, a cluster is called* $(1+\epsilon)\delta$*-good, if there exists one cluster in the current clustering with average distance at most* $(1+\epsilon)\delta$*. We say a pair is* $(1+\epsilon)\delta$*-good if the average distance is at most* $(1+\epsilon)\delta$*.*

Notice that with the definition of $(1+\epsilon)\delta$-goodness, Lemma 16 is equivalent to claiming that by the end of merging there are no $(1+\epsilon)\delta$-good points left, or no $(1+\epsilon)\delta$-good pairs left. We are now ready to prove the theorems.

Proof of [Theorem 14 and Theorem 15**]** For Theorem 14, given any threshold δ, take any merge $(A, B) \to A \cup B$ done in any round for δ, by Lemma 17 we have $\text{Avg}(A, B) \leq 4c\alpha(1+\epsilon)\delta$. However, $\min_{\{S_1,S_2\}\subseteq\mathcal{C}} \text{Avg}(S_1, S_2) \leq \delta$ by Lemma 16, since all LSH rounds for threshold

value $\delta/(1+\epsilon)$ are over. Therefore, the approximation ratio is at most $4c\alpha(1+\epsilon)$ for all merges.

For Theorem 15, since with high probability it eliminates all $(1+\epsilon)\delta$-good points for threshold δ, and within $O(\log n)$ number of thresholds we have investigated all the possible average-distance values since it must be at most $\max_{\{i,j\}\subseteq S} D(i,j)$ and therefore merged all the pairs. □

Now we prove Lemmas 16 and 17. The proof of Lemma 17 is a simple application of Lemma 11 combined with the fact that the algorithm does average-linkage on the embedded clusters until the minimum average distance reaches the threshold $\alpha(1+\epsilon)\delta$. So in both robust merging and average-linkage merging, the pairs we merge are always $4c\alpha(1+\epsilon)\delta$-good. The rest of this section will be devoted to proving Lemma 16.

In each big WHILE loop of LSH, Algorithm 1 can go to vanilla average-linkage if all buckets are small, and to RobustMerging if we have at least one big bucket. For any round of LSH, we have the following lemma.

Lemma 19 *Every round of LSH that ends up with doing average-linkage within each bucket has at least constant probability p_0 of reducing the number of $(1+\epsilon)\delta$-good pairs by a constant of $\frac{p_1}{10}$, and $p_0 \geq \frac{9}{10}p_1$.*

Proof

Let's focus on one round of LSH, suppose that before LSH we have N number of $(1+\epsilon)\delta$-good pairs. Regardless of whether average-linkage or RobustMerging takes place after LSH, we use E_0 to denote the event, where at least $\frac{p_1}{10}N$ $(1+\epsilon)\delta$-good pairs collide. Let $p_{collide}$ be the probability of this occurring. Let $N_{collide}$ denote the number of $(1+\epsilon)\delta$-good pairs that collide during the LSH. Since every good pair has probability at least p_1 of colliding, we have $\mathbb{E}[N_{collide}] \geq p_1 \cdot N$ by linearity of expectation. So

$$\begin{aligned} p_1 \cdot N \leq \mathbb{E}[N_{collide}] &\leq (1-p_{collide}) \cdot \frac{p_1}{10}N + p_{collide} \cdot N \\ &\leq \frac{p_1}{10}N + p_{collide}N \end{aligned}$$

This yields $p_{collide} \geq \frac{9}{10}p_1$. Note that the event E_0 happens with probability at least $p_{collide}$, independent of the location of the points and the merges we have done before. ■

We have established the fact that every single round of LSH has some constant probability p_0 of getting a constant portion of good pairs to collide. Here we quote a version of Chernoff's bound:

Theorem 20 (Lower Chernoff Bound) *Let $X_1, X_2, \ldots, X_t$ be a collection of independent Bernoulli RV's with $\Pr[X_i = 1] = p_i$. Let $X = \sum_i X_i$ and $\mu = \sum_i \mathbb{E}[X_i]$. Then for all $\delta \in (0,1)$, $\Pr[X < (1-\delta)\mu] \leq \exp\left(\frac{-\delta^2\mu}{2}\right)$.*

By performing $\Theta(\frac{1}{p_0}\log(n))$ rounds of LSH and using Theorem 20, with high probability E_0 happens $\Theta(\log(n))$ times during these rounds. We focus only on the LSH rounds where E_0 happens. During these rounds, Algorithm 1 either goes to average-linkage or robust merging. At least one of them will be repeated $\Theta(\log(n))$ times.

For average-linkage, we prove the following lemma. Intuitively, it guarantees that the total number of good pairs will go down by a constant factor.

Lemma 21 *Suppose in the t^{th} round of LSH event E_0 happens. Let N_t denote the number of $(1+\epsilon)\delta$-good pairs before and N_{t+1} denote the number of $(1+\epsilon)\delta$-good pairs after. If Algorithm 1 goes to average-linkage, then $N_{t+1} \leq \frac{9}{10}N_t$. If it goes to robust merging, then $N_{t+1} \leq N_t$.*

Proof After average-linkage, since we have merged the clusters until the minimum average distance for $\phi(\cdot)$ hits $\alpha(1+\epsilon)\delta$, the true minimum average distance is more than $(1+\epsilon)\delta$. So there remains no $(1+\epsilon)\delta$-good pairs in all the buckets.

If we can somehow prove when we merge two clusters in the same bucket, the number of $(1+\epsilon)\delta$-good pairs across the buckets does not go up, we can safely conclude that $N_{t+1} \leq N_t - N_{collide} \leq \frac{9}{10}N_t$. Suppose average-linkage chooses to merge clusters A and B. For any cluster C in other buckets, if $\text{Avg}(A \cup B, C) \leq (1+\epsilon)\delta$, we must have $\min\{\text{Avg}(A,C), \text{Avg}(B,C)\} \leq (1+\epsilon)\delta$, so at least one of the two pairs (A,C) and (B,C) are good. Now we merge A and B and only one good pair is produced, therefore the number of $(1+\epsilon)\delta$-good pairs across the buckets will not increase if we do average-linkage within each bucket. For the same reason, if Algorithm 1 does robust merging, we also have $N_{t+1} \leq N_t$. Thus Lemma 21 is proved. ■

Recall the following lemma from previous subsection:

Lemma 22 *Fix any threshold δ and let $\mathcal{C}^t$ be the set $\mathcal{C}$ in Algorithm 2 after t iterations. We have $|\mathcal{C}^{t+1}| \leq \frac{1}{2}|\mathcal{C}^t|$.*

Intuitively, in every LSH such that at least $\frac{p_1}{10}$ portion of $(1+\epsilon)\delta$-good pairs collide, either buckets are small, in which case the number of $(1+\epsilon)\delta$-good pairs will shrink by a constant factor with constant probability $p_{collide}$, or there is at least one bucket that's big, in which case the number of candidates for $(1+\epsilon)\delta$-good clusters will shrink by a constant factor with high probability. Putting Lemma 21 and 22 gives us Lemma 16.

Appendix B. Average-Linkage Cluster Embedding Proofs

Lemma 23 *For any point y and finite set of points X in Euclidean space, $\sum_{x\in X}\|x-y\|^2 = |X|\cdot\|y-\mu(X)\|^2 + |X|\cdot\text{Var}(X)$.*

Proof of [Lemma 23**]** Rewriting and expanding the square via $\|a+b\|^2 = \|a\|^2+\|b\|^2+2\langle a,b\rangle$ we have:

$$\begin{aligned}
\sum_{x\in X}\|x-y\|^2 &= \sum_{x\in X}\|x-\mu(X)+\mu(X)-y\|^2 \\
&= \sum_{x\in X}\|x-\mu(X)\|^2 + \sum_{x\in X}\|\mu(X)-y\|^2 \\
&\quad + \sum_{x\in X} 2\langle x-\mu(X), \mu(X)-y\rangle \\
&= |X|\cdot\text{Var}(X) + |X|\cdot\sum_{x\in X}\|y-\mu(X)\|^2
\end{aligned}$$

where the last term on the second line evaluates to 0. □

Proof of [Proposition 2**]** Applying Lemma 23 to the sum $\sum_{y\in B}\sum_{x\in A}\|x-y\|^2$ twice we have:

$$\sum_{y\in B}\sum_{x\in A}\|x-y\|^2 = \sum_{y\in B}\left(|A|\cdot\|y-\mu(A)\|^2 + |A|\cdot\mathrm{Var}(A)\right)$$
$$= |A||B|(\|\mu(A)-\mu(B)\|^2 + \mathrm{Var}(A) + \mathrm{Var}(B))$$

and dividing by $|A|\cdot|B|$ yields the proposition. □

Lemma 24 *For any two clusters A and B, $R_A(\mathrm{gm}(A)) \le \mathrm{Avg}(A,B)$.*

Proof of [Lemma 24**]** Since $\mathrm{gm}(A)$ minimizes $R_A(y)$ for any $y\in\mathbb{R}^k$, for all $y\in B$ we have $R_A(\mathrm{gm}(A)) \le R_A(y)$. Summing this inequality over all such $y\in B$ we have $|B|R_A(\mathrm{gm}(A)) \le \sum_{y\in B}R_A(y) = \sum_{y\in B}\sum_{x\in A}\frac{\|y-x\|}{|A|}$. Dividing both sides by $|B|$ yields the lemma. □

Lemma 25 *For any cluster X, $\mathrm{Dev}(X) \le 2R_X(\mathrm{gm}(X))$.*

Proof of [Lemma 25**]** By the definition of $\mathrm{Dev}(X)$ and the triangle inequality: $\mathrm{Dev}(X) = \frac{1}{|X|}\sum_{x\in X}\|x-\mu(X)\| \le \frac{1}{|X|}\sum_{x\in X}(\|x-\mathrm{gm}(X)\| + \|\mathrm{gm}(X)-\mu(X)\|)$
$= R_X(\mathrm{gm}(X)) + \|\mathrm{gm}(X)-\mu(X)\|$.

Now to complete the proof we claim that $\|\mathrm{gm}(X)-\mu(X)\| \le R_X(\mathrm{gm}(X))$. Using the definition of $\mu(X)$ and subadditivity we have: $\|\mu(X)-\mathrm{gm}(X)\| = \left\|\sum_{x\in X}\frac{x-\mathrm{gm}(X)}{|X|}\right\| \le \frac{1}{|X|}\sum_{x\in X}\|x-\mathrm{gm}(X)\| = R_X(\mathrm{gm}(X))$. □

Lemma 26 *For any two clusters A,B, $\|\mu(A)-\mu(B)\| \le \mathrm{Avg}(A,B)$.*

Proof of [Lemma 26**]** By definition of centroid we have:

$$\mu(A)-\mu(B) = \frac{1}{|A|}\sum_{x\in A}x - \frac{1}{|B|}\sum_{y\in B}y$$
$$= \frac{1}{|A||B|}\sum_{x\in A}\sum_{y\in B}x - \frac{1}{|A||B|}\sum_{x\in A}\sum_{y\in B}y$$
$$= \frac{1}{|A||B|}\sum_{x\in A}\sum_{y\in B}(x-y)$$

Now applying $\|\cdot\|$ to both sides and then subadditivity of $\|\cdot\|$ completes the proof. □

Proof of [Lemma 3**]** We start by proving the lower bound. By the triangle inequality, for any $x\in A$ and $y\in B$ we have $\|x-y\| \le \|x-\mu(A)\| + \|\mu(A)-\mu(B)\| + \|y-\mu(B)\|$. Summing this inequality over all $x\in A$ and all $y\in B$ then dividing by $|A|\cdot|B|$ yields $\mathrm{Avg}(A,B) \le f(A,B)$. For the upper bound we have the following sequence of inequalities: $\|\mu(A)-\mu(B)\| + \mathrm{Dev}(A) + \mathrm{Dev}(B) \le \mathrm{Avg}(A,B) + 2R_A(\mathrm{gm}(A)) + 2R_B(\mathrm{gm}(B)) \le 5\,\mathrm{Avg}(A,B)$.

The first inequality follows from Lemmas 26 and 25, and the second inequality follows from applying Lemma 24 to the last two terms. □

The next theorem is an extension of Theorem 1 into general ℓ_p norm where $p\in[1,+\infty]$.

Theorem 27 *Given a clustering $\mathcal{C}$ of $S \subset \mathbb{R}^d$, there exists a constant $\alpha \geq 1$ and a mapping $\phi : \mathcal{C} \to \mathbb{R}^{d'}$ such that for all $A, B \in \mathcal{C}$, $\text{Avg}(A, B) \leq \|\phi(A) - \phi(B)\| \leq \alpha \, \text{Avg}(A, B)$. In particular, we have $d' = d + |\mathcal{C}|$ and $\alpha = 5 \cdot 3^{1-\frac{1}{p}}$.*

In order to prove the theorem, given any ℓ_p-norm, we define the geometric median according: $\text{gm}(A) = \arg\min_{y \in \mathbb{R}^d} \sum_{x \in X} \frac{||y-x||_p}{|X|}$. Also, the following proposition is Proposition 4 generalized into ℓ_p-norm:

Proposition 28 *For all $a, b, c \in \mathbb{R}_+$, and $p \geq 1$, $\frac{1}{3^{1-\frac{1}{p}}}(a+b+c) \leq (a^p+b^p+c^p)^{\frac{1}{p}} \leq a+b+c$.*

Proof In general, for vectors in $\mathbb{R}^n$, it is known that for $0 < r < p$, $||x||_p \leq ||x||_r \leq n^{\frac{1}{r}-\frac{1}{p}}||x||_p$. We let $n = 3$, $x = (a, b, c)$ and $r = 1$, then we get $(a^p+b^p+c^p)^{\frac{1}{p}} \leq a+b+c \leq 3^{1-\frac{1}{p}}(a^p+b^p+c^p)^{\frac{1}{p}}$, which is the proposition. ■

Proof of [Theorem 27] Lemma 24 still holds. Lemma 25 and Lemma 26 also holds, since what we used in the proofs are the triangle inequality, and convexity of norms, which hold for any ℓ_p-norm: $||x - (ty_1 + (1-t)y_2)||_p \leq t||x - y_1||_p + (1-t)||x - y_2||_p$ for $t \in [0, 1]$. As a result, Lemma 3 holds for any ℓ_p-norm. Now, combining with Proposition 28, analogous to (2), we let the embedding for cluster C_i be:

$$\phi(C_i) := 3^{1-\frac{1}{p}}(\mu(C_i), 0, \ldots, \underbrace{\text{Dev}(C_i)}_{d+i\text{'th coordinate}}, \ldots, 0) \tag{4}$$

For any cluster A and B, $||\phi(A)-\phi(B)||_p = 3^{1-\frac{1}{p}}(||\mu(A)-\mu(B)||_p^p+\text{Dev}(A)^p+\text{Dev}(B)^p)^{\frac{1}{p}}$. In Proposition 28, let $a = \text{Dev}(A)$, $b = \text{Dev}(B)$ and $c = ||\mu(A) - \mu(B)||_p$, then it shows that $||\phi(A) - \phi(B)||_p$ lies in the range $[a + b + c, 3^{1-\frac{1}{p}}(a + b + c)]$. We have $\text{Avg}(A, B) \leq ||\phi(A) - \phi(B)||_p \leq 5 \cdot 3^{1-\frac{1}{p}} \text{Avg}(A, B)$ then. For the special case where $p = 2$, this is what we showed in (2). □

Appendix C. Additional Experiments

Here we present additional data from our experimental evaluation.

First we give our results for the sub-quadratic average-linkage algorithm due to Abboud et al. (2019) in Table 10. We present the average running time across five sampled instances at each input size. Note that these times are significantly worse than our algorithm as shown in Section 6. Moreover, this algorithm was unable to scale to large input sizes over $\approx$ 100k points as our algorithm did. Also observe that the performance of this method significantly degrades on higher dimensional data sets such as Blog and Covertype, which is due to the significant increase in the dimension the algorithm carries out.

Input Size	64	128	256	512	1024	2048	4096	8192
Shuttle	41.45	102.75	229.83	634.35	2897.20	7404.18	18432.55	46840.01
Blog	2140.38	6030.19	16089.31	52891.13	-	-	-	-
Covertype	407.44	1204.50	3369.60	13730.49	47937.79	161861.26	-	-
Higgs (S)	37.77	80.92	189.81	511.43	2936.55	8001.01	21468.35	-
Susy (S)	6.48	13.84	31.85	95.76	374.26	977.79	2896.95	8565.05

Table 10: Average running time in seconds for algorithm due to Abboud et al. (2019). A blank entry indicates that the algorithm crashed due to running out of memory at this input size.