Hierarchical Clustering in General Metric Spaces using Approximate Nearest Neighbors

Benjamin Moseley
Carnegie Mellon University

Sergei Vassilvitskii
Google

Yuyan Wang
Carnegie Mellon University

Abstract

Hierarchical clustering is a widely used data analysis method, but suffers from scalability issues, requiring quadratic time in general metric spaces.

In this work, we demonstrate how approximate nearest neighbor (ANN) queries can be used to improve the running time of the popular single-linkage and average-linkage methods. Our proposed algorithms are the first subquadratic time algorithms for non-Euclidean metrics. We complement our theoretical analysis with an empirical evaluation showcasing our methods' efficiency and accuracy.

1 Introduction

Hierarchical clustering is a popular unsupervised learning method used to cluster data when the precise number of clusters is not known apriori. Algorithms such as *single-linkage*, *average-linkage* and *complete-linkage* are common examples of methods for hierarchical clustering, and, while they started out as heuristics, recent work has shed light on the objectives these different methods optimize (6, 8, 9, 14).

All of these procedures fall into the family of agglomerative clustering algorithms. They start out with each data point in its own singleton cluster, and then merge two clusters at a time, using different objectives to select which pair to merge. While extremely popular, a notable downside of these methods is that they run in time quadratic in the number of points: given a dataset of n points, $n-1$ merges are required, and finding the optimal merge is itself a linear time operation.

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

For ℓ_1 metrics, Abboud *et al.* (1) showed how to use Approximate Nearest Neighbor (ANN) data structure to improve the running time to be subquadratic for average linkage. The ANN data structure, constructed based on locality sensitive hashing (LSH), can be used to quickly find points that are close to any given point. In this work we generalize their approach and show how to implement it in any space where a hashing function like LSH exists so that an ANN data structure can be constructed. Our key technical contribution is a new way to use ANN data structures to compute average similarity between a pair of clusters.

Many problems, however, are best captured by general metric spaces for which *ANN schemes do not exist*. Consider, for instance, the shortest path metric on a road graph. A general road network does not have a good ANN and therefore there is no fast average-linkage algorithm known. To overcome this we could use the latitude and longitude as the Euclidean location of points and use ℓ_2 distance to be a *proxy* for the road network distance. Then we have a space where LSH and an ANN data structure exists and we can leverage our algorithm. This has issues though because while the road distance and proxy are often close, natural obstacles like mountains and rivers may cause two points that are in close proximity in the Euclidean space are far apart according to the road metric.

This phenomenon is common. In several instances we have a well-behaved *proxy* metric that approximately preserves distances for majority of pairs. For instance, the distances between points after applying dimension reduction like Johnson–Lindenstrauss (10) or metric embedding (12). Using these techniques, most pairwise distances are approximately preserved, but a few of the pairs have high error.

This paper investigate how one can use such proxies to improve running times of hierarchical clustering in general metric spaces. One can, of course, directly use them in place of the true metric to compute a hierarchical clustering in linear time. This approach has two drawbacks. First it leads to poor solutions as small errors can propagate and lead to a very different

solution. Second, it is brittle and offers no trade-off between the quality of the solution and the running time. In contrast, we show how to use proxy metrics, ANN data structures, and the original metric in a way that allows us to keep the subquadratic running time and only pay a small cost in the accuracy of the final clustering. Our methods are applicable to any general metric for which such a proxy exists.

Related Work Hierarchical clustering encompasses a collection of methods that give laminar decompositions of the data. It received renewed attention with a breakthrough result by (9) who developed a cost function for the problem; this was later extended by (8) and (14), who gave theoretical justifications for the average-linkage algorithm. This work was further improved by (2) and (6) who gave tighter analyses and approximation ratios.

The focus of our work is on improving the running time of hierarchical clustering algorithms in general metric spaces. Our main tool will be that of locality sensitive hashing (LSH), first introduced by (13), that has had many developments, see for instance (3). However, as (7) showed formally, not all metric spaces support such Approximate Nearest Neighbor (ANN) data structures.

For hierarchical clustering specifically, (1) recently gave a way to use locality sensitive hashing to speed up average linkage algorithms. Critically, however, the methods in (8) only work for ℓ_1 spaces. We extend them to use arbitrary ANN data structures by identifying invariant properties that make speed improvements possible.

Our Contributions In this work we give new algorithms for improving the running time of hierarchical clustering algorithms.

- We show how to use Approximate Nearest Neighbor (ANN) data structures to speed up running times of single-linkage and average-linkage algorithms.
- In general metric spaces, when ANN data structures are not available, we show how to use proxy metrics to trade-off accuracy with running time.
- We demonstrate such a trade-off empirically, and show how to find hierarchical clusterings within a few percent of optimal in near linear running time.

2 Preliminaries

Agglomerative hierarchical clustering . Let S be a set of points equipped with a distance function $d : S \times S \mapsto \mathbb{R}^+$. Agglomerative hierarchical clustering algorithms start by initializing every point to be a singleton cluster, and iteratively pick two clusters with smallest dissimilarity to merge into a new cluster, until there is only one cluster left.

Different definitions of dissimilarity lead to different clustering algorithms. The **average linkage** algorithm defines the dissimilarity to be the average distance between all pairs of points across the two clusters. The **single linkage** algorithm defines it as smallest distance among all pairs of points.

Let $\text{avg}(A, B) = \frac{1}{|A||B|} \sum_{a \in A, b \in B} d(a, b)$ be the **average linkage distance** between two clusters A and B. The average linkage distance function is a metric, it satisfies the triangle inequality, $\text{avg}(A, B) \leq \text{avg}(A, C) + \text{avg}(B, C)$ for any three clusters $A, B, C \subseteq S$ (See Lemma 3.1 in (1)). We will also use the fact the minimum average linkage distance is monotonically non-decreasing with merging: $\text{avg}(A, B \cup C) \geq \min\{\text{avg}(A, B), \text{avg}(A, C)\}$ for all $A, B, C \subseteq S$. The same property holds for the single linkage.

The single linkage algorithm is equivalent to Kruskal algorithm for computing the Minimum Spanning Tree (MST) on the complete graph where edge weights correspond to the distance between points. We will also be interested in the approximate version of Kruskal algorithm, which selects any edge at each step whose distance is within a factor c of the smallest edge between different components. This version always returns a tree of weight at most c times optimal (5).

Approximate Nearest Neighbors. A key building block of our techniques will be the approximate nearest neighbor (ANN) data structure. We recall here key facts about Locality Sensitive Hashing (LSH) (13), see also the survey by Andoni & Indyk (3).

The data structure relies on an existence of a family of hash functions $\mathcal{H}$ with each $h \in \mathcal{H}$ a function from S to $\mathbb{Z}$, that map nearby points to the same hash value.

We recall the formal definition from (11). Let $B_S(q, r)$ denote the set of points in S that has distance at most r from point $q \in S$.

Definition 1 ($(r, \gamma r, p_1, p_2)$-sensitiveness)**.** *For any target distance r, point set S and any query point $q \in S$, given a constant factor γ ($\gamma > 1$), the hash function h is $(r, \gamma r, p_1, p_2)$-sensitive, if for $p_1 > p_2$:*

- $\forall v \in B_S(q, r)$, $Pr[h(q) = h(v)] \geq p_1$
- $\forall v \notin B_S(q, \gamma r)$, $Pr[h(q) = h(v)] \leq p_2$

Families $\mathcal{H}$ satisfying these constraints are known to exist for ℓ_p norm distances in Euclidean space for $p \in (0, 2]$ (11), and are known not to exist for some metric spaces (7).

Using a family of such functions $\mathcal{H}$ we can construct a data structure for approximate nearest neighbor

queries.

As we mentioned, such a family of functions $\mathcal{H}$ might not exist for a general distance function $d : S \times S \mapsto \mathbb{R}^+$ we are considering. However, assume we have a **proxy** metric $d' : S \times S \mapsto \mathbb{R}^+$ that approximates d by a factor of $\beta \geq 1$. Up to scaling, we assume that for any pair of points $(x, y) \in S \times S$, $\frac{1}{\beta} \leq \frac{d'(x,y)}{d(x,y)} \leq 1$. If there exists an ANN data structure for d', using d' as a proxy for d and applying a $(r, \gamma' r, p_1, p_2)$-sensitive function on d' gives a $(r, \beta\gamma' r, p_1, p_2)$-sensitive function for d. Then, we can combine the proxy metric d' and its LSH function to construct a data structure that works for d.

Definition 2. *For any set of points $P \subseteq S$, $Q_P(q, r)$ is a valid (r, γ)-Near-Neighbor (NN) query, if it satisfies these conditions:*

- *If $B_P(q, r) = \emptyset$, then $Q_P(q, r)$ returns $\emptyset$.*
- *If $B_P(q, r) \neq \emptyset$, then $Q_P(q, r)$ gives a point $p \neq q \in P$, such that $d(p, q) \leq \gamma \cdot r$.*

We call the point found by an approximate NN-query a *neighbor*.

([13]) and ([11]) described the way to build a data structure $\mathcal{D}(S, r, \gamma)$ that allows one to answer (r, γ)-NN queries on any subset P using $(r, \gamma r, p_1, p_2)$-sensitive hashes. Importantly, the queries run in time sublinear in $|S|$.

We let $n = |S|$ and $H(n)$ denote the time needed to construct a hash function. By the scheme described in ([11]), $\mathcal{D}(S, r, \gamma)$ is constructed by concatenating $O(\log n)$ hashes and repeating for $O(n^\rho \log n)$ times. Thus the construction time for $\mathcal{D}(S, r, \gamma)$ is bounded by $O(n^\rho \log^2 nH(n))$, where $\rho = \frac{\ln(1/p_1)}{\ln(1/p_2)} \in (0, 1)$.

Moreover, assuming it takes constant time to insert, delete or access an arbitrary point in any bucket, the query time of $Q_P(q, r)$ is $T(n, \gamma) = O(n^\rho \log n)$, while the insert/delete time is $D(n, \gamma) = O(n^\rho \log^2 n)$. We note that for a fixed p_1, p_2 decreases as γ grows, hence making ρ a function of γ.

3 Warm-up: Using ANNs to Approximate Single Linkage

In this section, we give a brief description of how an ANN data structure can be used to implement single linkage clustering.

Both ([13]) and ([4]) proposed algorithms that use (r, γ)-Near-Neighbor to compute an Approximate Minimum Spanning Tree. Here we provide a modified version that is more in line with our average-linkage algorithm described in the next section.

The goal is to implement the approximate variant of Kruskal's algorithm. Initially the tree T is empty. The algorithm iteratively selects an edge e and adds to to T if (1) $T \cup \{e\}$ does not form a cycle and (2) it is within a factor of $(1 + \epsilon)$ of the cheapest such edge.

We now re-interpret this approach in the language of single linkage. Initially all points are clusters. In each step, clusters C_i and C_j are merged if $\min_{a \in C_i, b \in C_j} d(a, b)$ is at most $(1 + \epsilon) \min_{i' \neq j'} \min_{a \in C_{i'}, b \in C_{j'}} d(a, b)$.

Let S be the set of input points. Without loss of generality, assume the smallest interpoint distance is 1 and the aspect ratio is $\Delta = \max_{u,v \in S} d(u, v)$. We want to find a pair of clusters to merge. We fix some $\delta > 0$, and proceed by merging all clusters with distance at most δ between them. When no more merges are possible, we increment δ by a $(1 + \epsilon)$ factor and repeat. Suppose we have a partition of S into different components (clusters) $\mathcal{C} = \{C_1, \ldots, C_m\}$. We will find edges with weights at most $\gamma\delta$ between components and merge components that share such an edge. It is easy to see that this results in a $(1 + \epsilon)\gamma$-approximation of the MST.

How then to use the ANN data structure to find clusters to merge? For a threshold δ, construct a data structure $\mathcal{D} = \mathcal{D}(S, \delta, \gamma)$. Pick any component $C_i \in \mathcal{C}$, remove the points in C_i from $\mathcal{D}$, and add all points in C_i to a query list. Query the first point in the list, p, using the current $\mathcal{D}$. If the query returns a neighbor q, let C_j be the component containing q. Add edge (p, q) to the tree T. Then add all points in C_j to the query list and remove C_j from $\mathcal{D}$. Repeat until the query returns an empty set for p, then use next point in the query list and repeat. If the query list becomes empty but there are components that haven't been queried before, pick an arbitrary component and do repeat the process again. The querying stops when $\mathcal{D}$ becomes empty.

The definition of the data structures immediately yields the following theorem; we defer the proof to the Supplementary Material.

Theorem 3. *With high probability, the algorithm returns a $(1 + \epsilon)\gamma$-approximate minimum spanning tree or singe-linkage tree, and has $O(\frac{1}{\epsilon} \log \Delta \log^2 n^\rho H(n))$ running time.*

4 Average Linkage

We now expand our method to show how to use ANNs to construct a provably approximate *average-linkage* algorithm. Using ANNs the algorithm iteratively merges clusters with approximately smallest average distance. Naively computing the distance between two clusters requires comparing the total distance of all pairs of points in the two clusters; we wish to avoid this quadratic

dependence.

Similar to the single linkage case, the average linkage algorithm will consider a geometric series of thresholds δ of the form $(1+\epsilon)^k$ for $k \in [\lceil \log \Delta \rceil]$. For any fixed δ, the algorithm merges clusters within $\gamma(1+O(\epsilon))\delta$ of each other.

Algorithmic Structure We now detail how to use ANNs to find clusters with average linkage distance $\gamma(1+O(\epsilon))\delta$ or smaller.

Let $\mathcal{C}$ be the current set of clusters. We represent every cluster in $\mathcal{C}$ by a point in the cluster that is close to the cluster center. (If points were in Euclidean space, we would use the centroid of the cluster.) Let $\phi(C_i)$ be the point in C_i we use to represent this cluster, we define this formally in the next section. We will maintain a data structure of the points $\phi(C_i)$ for $C_i \in \mathcal{C}$. This data structure allows for queries of the form $Q(\phi(C_i), r)$ for a r close to δ. The query finds all pairs C_i, C_j such that $d(\phi(C_i), \phi(C_j)) \leq \gamma r$. Intuitively, the mapping ϕ is chosen such that for any two clusters, the distance between the mapped points preserves their average distance up to some constant factor.

Unfortunately, it is hard to select a ϕ such that $d(\phi(C_i), \phi(C_j))$ approximate $d(C_i, C_j)$ for any pair of clusters C_i, C_j. For example, consider two clusters with overlapping centers, but points are close to the center in one of them, and far away from the center in another.

To make up for the information loss, for every cluster $C \in \mathcal{C}$, we maintain two values: one is $\phi(C)$ and the other is $\sigma(C) = \frac{1}{|C_i|}\sum_{v \in C_i} d(v, \phi(C_i))$. Intuitively $\sigma(C)$ represents the "deviation" of points in C from the center $\phi(C)$. Given any two clusters $C_i, C_j \in \mathcal{C}$, we use the $d(\phi(C_i), \phi(C_j))$ and $\sigma(C)$ to approximate the average distance $\text{avg}(C_i, C_j)$.

At a high level, the following properties should hold for $\phi(\cdot)$ and $\sigma(\cdot)$. These properties will allow us to identify clusters whose average linkage distance is small enough to merge them. (Formally, we will need slight variations on these, see the paragraph on Construction of Center and Deviation for exact details.)

1. $d(\phi(C_i), \phi(C_j)) - \sigma(C_i) - \sigma(C_j) \leq \text{avg}(C_i, C_j) \leq d(\phi(C_i), \phi(C_j)) + \sigma(C_i) + \sigma(C_j)$.
2. $\sigma(C_i), \sigma(C_j) \leq c \cdot \text{avg}(C_i, C_j)$ for a constant c.

We now describe how we find which pairs of clusters can be merged and which ones we should not consider merging. For any cluster C_i, we use the deviation term $\sigma(C_i)$ as a filter. Observe that if $\sigma(C_i) > c\delta$, then $\text{avg}(C_i, C_j) > \delta$ for any C_j by the above property. This implies that no C_i with $\sigma(C_i) > c\delta$ needs to be considered for merging with any other cluster.

We thus remove all such clusters from consideration. Since the deviation terms for all remaining clusters are small, property one will allow the algorithm to merge any pair of clusters where $d(\phi(C_i), \phi(C_j))$ is small compared to δ. To that end, as stated before we build a nearest neighbor data structure based on the points $\phi(C_i)$ for all remaining clusters.

There is one problem remaining. After a merge the new cluster needs to be considered for subsequent merges. The question is how to adapt the data structure efficiently such that the query is still valid? There are two steps involved: 1) finding a representative point for the new cluster and measuring deviation, and 2) deleting the old clusters and inserting the new cluster into the data structure.

For both, we will develop a fast algorithm that constructs a randomized approximation to ϕ and σ.

Construction of Center and Deviation For each cluster C in $\mathcal{C}$ we show how to construct $\phi(C)$ and $\sigma(C)$. The purpose of this section is to establish that these quantities exist for every cluster such that they have the requisite properties. Algorithmically, this section will allow us to discover any cluster C to discard based on $\sigma(C)$. For the kept clusters we will put their $\phi(C)$'s into the data structure, then use ANN queries to decide which pairs to merge.

If $|C| \leq \frac{1}{\epsilon}$, set $\phi(C) = \arg\min_{p \in C} \text{avg}(C, p)$ for some constant $0 < \epsilon < 1$ to be set later. This takes $O(\frac{1}{\epsilon}|C|)$ time to determine. Now, consider C such that $|C| \geq \frac{1}{\epsilon}$. To begin, we calculate $(\phi(C), \sigma(C))$ as follows. Sample m points uniformly from C to get a sample C'. Find the point $v \in C'$ such that $\sum_{p \in C} d(v, p)$ is minimized. Set $\phi(C)$ equal to v. Then set $\sigma(C) = \text{avg}(C, \phi(C)) = \text{avg}(C, v)$.

We now aim to show the properties regarding ϕ and σ. The proof can be found in Section A of the Supplementary material.

Lemma 4. *If $m = \Theta(\frac{1}{\epsilon}\log n)$, with high probability, $\sigma(C) = \text{avg}(\phi(C), C) \leq (1 + \frac{1}{1-\epsilon})\min_{p \in S} \text{avg}(p, C) \leq 2(1+\epsilon)\min_{p \in S} \text{avg}(p, C)$ for $\epsilon \leq 0.5$.*

The prior lemma immediately gives the following corollary.

Corollary 5. *For any cluster C_i with high probability if $\sigma(C_i) \geq 2(1+\epsilon)\delta$ then given any other cluster C_j, $\text{avg}(C_i, C_j) \geq \delta$ for $\epsilon \leq 0.5$.*

The corollary ensures that if $\sigma(C_i) > 2(1+\epsilon)\delta$ then w.h.p. C_i has average linkage distance greater than δ from every cluster in $\mathcal{C}$. Thus, C_i need not be considered for a merge. Moreover, since average linkage distance increases as clusters get merged, we never need to consider C_i for this δ.

The following lemma is a direct application of triangle inequality of average distances.

Lemma 6. *For any two clusters C_i and C_j, $d(\phi(C_i), \phi(C_j)) - \sigma(C_i) - \sigma(C_j) \leq \text{avg}(C_i, C_j) \leq d(\phi(C_i), \phi(C_j)) + \sigma(C_i) + \sigma(C_j)$.*

The following run time lemma bounds the running time of the construction of σ and ϕ.

Lemma 7. *For a cluster C, calculating $(\phi(C), \sigma(C))$ takes $O(\frac{1}{\epsilon}|C| \log n)$.*

Proof. It takes $O(\frac{1}{\epsilon}|C| \log n)$ time to sample $\frac{1}{\epsilon} \log n$ points, and $O(\frac{1}{\epsilon}|C| \log n)$ time to measure the average distance from C to every one of them, picking the center and calculating deviation. □

Formal Algorithm We present the formal algorithm which follows the intuition given in the previous section and the developed definitions of ϕ and σ.

See Algorithm 1 for the pseudocode. As stated, the algorithm considers δ of geometrically increasing values. Fix $\delta = (1+\epsilon)^{k-1}$ and $\mathcal{C}$ to be the current set of clusters. Initially $\delta = 1$ and $\mathcal{C}$ has a cluster for each individual point. Let P_k be the set of centers of clusters with small deviation: $P_k = \{\phi(C) \mid C \in \mathcal{C}, \sigma(C) \leq 2(1+\epsilon)\delta\}$.

The set P_k contains centers of clusters in $\mathcal{C}$ for which there may exist another cluster within average distance δ. We maintain a data structure that supports (r, γ)-NN queries on P_k for a chosen $r = O(\delta)$, denoted by $\mathcal{D}_k$. While P_k is not empty, we pick a point $\phi(C_i)$ from it and use it as a query in $\mathcal{D}_k$ to find a nearest neighbor. If there is no such point, $\phi(C_i)$ is discarded from P_k and $\mathcal{D}_k$. Otherwise, if the data structures returns another point $\phi(C_j)$, we merge C_i and C_j, adopting the merging procedure proposed by (1). This new point is added to P_k and $\mathcal{D}_k$.

For efficiency, the pair $(\phi(\cdot), \sigma(\cdot))$ is not recalculated at every merge. Rather, an update happens after a merge only if the cluster size grows significantly. We maintain a quantity $s(C)$ which denotes the size of cluster C the last time $(\phi(C), \sigma(C))$ was recalculated. When we merge C_i and C_j into a new cluster C, if $|C| \geq (1+\eta)\max\{s(C_i), s(C_j)\}$ for some fixed η, the center and deviation $(\phi(C), \sigma(C))$ are recalculated, and $s(C)$ is updated. Otherwise we use the center and deviation of the cluster with bigger $s(\cdot)$ value. See Algorithm 2 for details about merging.

In Section B in the Supplementary Material we show the following.

Theorem 8. *Algorithm 1 has the following guarantees:*

- *Let $n = |S|$ and Δ be the aspect ratio in S. The run time is $O(\frac{1}{\epsilon} n^{\rho} \log^2 n)(\log \Delta) H(n)$.*
- *With high probability, the algorithm has an approximation ratio of $(5\gamma + 4)(1 + O(\epsilon))$ for average linkage and completes the hierarchical clustering tree.*

Algorithm 1 Main Algorithm

procedure FastAverageLinkage(S, γ, ϵ)
  $\mathcal{C} \leftarrow \{\{p\} \mid p \in S\}$ ▷ Make leaf clusters.
  $\phi(\{p\}) \leftarrow p$, $\sigma(\{p\}) \leftarrow 0$, $s(\{p\}) \leftarrow 1$ for $p \in S$ ▷ Initialize center and deviation.
  for $k = 1, 2, \ldots, \log_{1+\epsilon} \Delta$ **do**
    $\delta \leftarrow (1+\epsilon)^{k-1}$
    $P_k \leftarrow \{\phi(C) \text{ for } C \in \mathcal{C} : \sigma(C) \leq 2(1+\epsilon)\delta\}$ ▷ Filter the clusters.
    $r \leftarrow 5(1+\epsilon)\delta$
    $\mathcal{D}_k \leftarrow \mathcal{D}(P_k, r, \gamma)$
    while $\mathcal{D}_k$ is not empty **do**
      Get an arbitrary $\phi(C_i)$ from P_k.
      Delete $\phi(C_i)$ from $\mathcal{D}_k$ and P_k. ▷ Maintain the data structure.
      while $Q_{P_k}(\phi(C_i), r) \neq \emptyset$ **do**
        $\phi(C_j) \leftarrow Q_{P_k}(\phi(C_i), r)$
        $\xi \leftarrow (1+\epsilon)^6(5\gamma + 4)$
        $\eta \leftarrow \frac{\epsilon^2}{1+\xi}$
        Merge($C_i, C_j, \mathcal{D}_k, P_k, \eta$)
  Return the resulting tree.

Algorithm 2 Merge Algorithm

procedure Merge($C_i, C_j, \mathcal{D}_k, P_k, \eta$)
  if $|C_i| + |C_j| \geq (1+\eta)\max\{s(C_i), s(C_j)\}$ **then**
    Update $(\phi(C_i \cup C_j), \sigma(C_i \cup C_j))$ ▷ Update only when cluster size increases by a factor.
    $s(C_i \cup C_j) \leftarrow |C_i| + |C_j|$
  else
    if $s(C_i) \geq s(C_j)$ **then**
      $(\phi(C_i \cup C_j), \sigma(C_i \cup C_j)) \leftarrow (\phi(C_i), \sigma(C_i))$
      $s(C_i \cup C_j) \leftarrow s(C_i)$
    else
      $(\phi(C_i \cup C_j), \sigma(C_i \cup C_j)) \leftarrow (\phi(C_j), \sigma(C_j))$
      $s(C_i \cup C_j) \leftarrow s(C_j)$
  Delete $\phi(C_j)$ from $\mathcal{D}_k$ and P_k, insert $\phi(C_i \cup C_j)$ into $\mathcal{D}_k$ and P_k only if $\sigma(C_i \cup C_j) \leq 2(1+\epsilon)\delta$.

5 Experiments

This section demonstrates the empirical effectiveness of our algorithms for both single linkage and average linkage clustering.

Recall that one of our motivations was that ANNs may not exist for general metrics, but often times there are well-behaved proxy metrics available; this is the setting we explore in this Section.

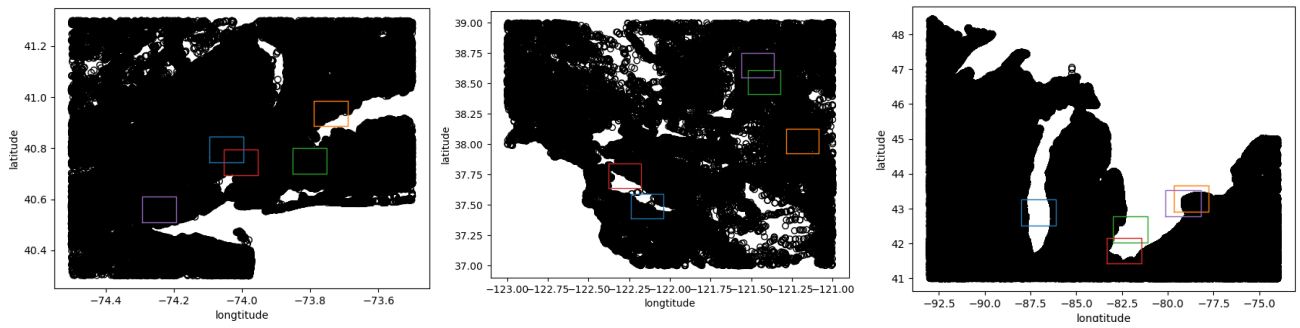

Figure 1: Subsamples for New York, Bay Area and Great Lakes

Table 1: Comparing the performance of different average linkage methods, New York

Sample Size	169	330	414	493	615	888	1141	1855
Aprx Ratio, mean, Proxy-AL	1.923	1.513	1.939	1.545	1.854	2.083	2.450	1.746
Aprx Ratio, mean, Proxy-Hash-AL	1.402	1.503	1.479	1.640	1.656	1.636	1.784	1.780
Aprx Ratio, 90%, Proxy-AL	3.705	2.096	3.678	1.813	2.865	3.913	3.240	2.372
Aprx Ratio, 90%, Proxy-Hash-AL	1.925	1.975	1.973	2.268	2.234	2.188	2.452	2.407
Aprx Ratio, max, Proxy-AL	21.391	13.930	28.233	20.042	54.844	44.783	228.615	42.991
Aprx Ratio, max, Proxy-Hash-AL	2.871	2.677	3.046	3.332	2.932	3.389	3.762	3.902
Global Obj, Proxy-AL	0.999	0.989	0.999	0.998	0.999	0.994	0.963	1.001
Global Obj, Proxy-Hash-AL	1.004	0.970	0.985	0.975	0.999	0.987	0.966	0.971

Our single linkage and average linkage implementations are named Proxy-Hash-SL and Proxy-Hash-AL, respectively. The goals of this section are to establish the following:

- Show that both Proxy-Hash-SL and Proxy-Hash-AL have strictly sub-quadratic running times.
- Show that using single/average linkage directly on the proxy metrics results in poor quality, yet Proxy-Hash-SL and Proxy-Hash-ALhave strong performance. This will show that they new algorithms are able leverage the proxy metrics to achieve scalability, while overcoming the shortcomings of using the proxy metrics directly.
- Demonstrate that using Proxy-Hash-SL and Proxy-Hash-AL, we can find solutions with only a small loss in quality compared with the solution found by using single/average linkage on the real metric.

Implementation Details. We implemented the algorithms in Section 3 and 4 with slight modifications. While building the ANN data structure for querying, we set the number of concatenations and repetitions to be constant. The values of the constants are tuned according to different data sets.

To improve show the trade-off between accuracy and running time, while querying a point using the data structure, we use LSH on the *proxy distance* to identify the set of candidates that are nearest neighbors to the query point, but then use the *real distance* to pick the closest candidate. We only take a constant number of neighbors from the point's LSH bucket and pick the one with smallest *real distance* from the query point. If the true distance is below the threshold δ_k, we merge the clusters.

Data Set	Nodes	Edges
New York	264,346	366,923
Bay Area	321,270	400,086
Great Lakes	2,758,119	3,442,829

Table 2: Dataset Details

Experiment 1: Road Maps. We use datasets from The 9th DIMACS Implementation Challenge[1]. The data files include the road networks of different cities. The data is given in graph format where nodes represent end points of roads, while edge weights represent the road lengths. Each node's latitude and longitude are provided. We choose three areas to study: New York, Bay Area, and the Great Lakes, see Table 2.

Based on the road network, the distance between any two points is the length of shortest path between them, termed *road distance*. This is correlated but not equivalent to the Euclidean distance calculated using lat/long values. We remark that the true metric is a *general metric*, and we will use the Euclidean metric as a proxy.

The original datasets have millions of points, so it is impractical to find all pairwise shortest-path road distances to compute the groundtruth average linkage clustering. We perform the following subsampling

[1] http://users.diag.uniroma1.it/challenge9/download.shtml

Table 3: Ratio between total road distance of the tree and real MST, New York

Sample Size	169	330	727	1166	1825	3765	6710	14428	28985
Proxy-SL	1.760	1.304	1.554	1.784	1.412	1.805	1.753	1.883	1.511
Proxy-Hash-SL	1.035	1.016	1.021	1.024	1.027	1.030	1.029	1.022	1.024

Table 4: Ratio between total real distance of the spanning tree and real MST, Seizure

Sample Size	100	200	400	800	1600
Proxy-SL	1.284	1.331	1.416	1.462	1.521
Proxy-Hash-SL	1.141	1.158	1.176	1.184	1.209

method: for each city, we draw a rectangle at a random position on the map. Then we take the subgraph induced by all points in this rectangle. If the subgraph is not connected, we take the biggest connected component. Figure 1 contains a map of all points for every city, and the boxes represent 5 rectangles drawn for some given lat/long lengths. This allows us to get subsamples with many different sizes and study the efficiency of our methods.

Experiment 2: Random Projections of High-Dimensional Datasets. We consider Euclidean datasets in a large number of dimensions and use a Johnson-Lindenstrauss (see, for instance, Dasgupta & Gupta (10) dimension reduction technique to reduce the number of dimensions. We consider the ℓ_2 distance between the original high-dimensional points as the real distance, and the ℓ_2 distance between the projected data points as the proxy distance.

We use the `seizure` data set from UCI data repository[2]. The dataset has 179 dimensions and 11500 points, with every point a recording of brain activity. We project the data to 4 dimensions and take subsamples of size $[100, 200, 400, 800, 1600]$ from the original data set and test Proxy-Hash-SL and Proxy-Hash-AL. For each of the data sizes we take 5 subsamples, and take the average of both performance and running time on these.

Performance Metrics. We use the following metrics to measure the performance of hierarchical clustering trees. For MST (single linkage), we use the objective in the MST problem. This is the total weight of the edges chosen in the spanning tree.

For average linkage, we use two metrics. Given a sequence of cluster merges, at every merge, using the real distance, we calculate the ratio between the average linkage of the merged clusters and the minimum average linkage. We call this metric the *approximation ratio*. Assuming there are n points, a hierarchical clustering tree gives $n-1$ such ratios. For both datasets we show the mean, 90%-percentile and the maximum of all approximation ratios. For vanilla average linkage on the *real distance*, this ratio is always 1. If all $n-1$ ratios are close to 1, the hierarchical clustering tree closely resembles the tree produced by average linkage. The other metric we use is the recently developed global objective for hierarchical clustering tree introduced in Cohen-Addad *et al.* (8).

Running Time. The bottleneck in all computations is the time spent on computing true distances between points. For the road map data set, every computation involves finding the shortest path between a pair of points; and for `seizure`, it is time consuming since the original data is high-dimensional. To give an implementation and problem independent view into the performance of our methods, we report the total number of *real distance computations* made by Proxy-Hash-SL and Proxy-Hash-AL.

Results. We first compare the performance of directly using proxy distance with using Proxy-Hash-SL and Proxy-Hash-AL. Namely, we first construct a hierarchical clustering tree by running MST/average linkage on the dataset using proxy distance as the distance metric. We use "Proxy-SL" and "Proxy-AL" to refer to the results produced in this way in all tables and figures. Then we construct another tree using our implementation of Proxy-Hash-SL/Proxy-Hash-AL. Then we compare the performance of these two hierarchical clustering trees using the proposed performance metrics.

For road map dataset, Table 3 compares the spanning tree found by Proxy-SLand Proxy-Hash-SL. The entries show the ratio of total weights of the spanning tree to the real MST (lower is better). We report the results for $\epsilon = 0.2$ in Proxy-Hash-SL. The results are quantitatively similar on all three datasets, we only show results for New York here, and postpone the other two cities to the Supplementary Material.

Table 1 compares the performance of hierarchical clustering tree by running average linkage directly on the proxy metric and Proxy-Hash-AL for road map in New York. The first column shows the performance metric we are using. Here "Aprx Ratio" refers to the approximation ratio, where "mean", "90%" and "max" refers to the mean, 90%-percentile and maximum of all approximation ratios for the tree, respectively. Following that, "Proxy-AL" or "Proxy-Hash-AL" specifies which algorithm we are using for that line in the table.

Analysis. The new algorithms perform significantly

[2] `https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition`

Table 5: Comparing the performance of different average linkage methods, Seizure

Sample Size	100	200	400	800	1600
Aprx Ratio, mean, Proxy-AL	1.723	1.955	2.141	2.651	3.159
Aprx Ratio, mean, Proxy-Hash-AL	1.210	1.240	1.284	1.348	1.623
Aprx Ratio, 90%, Proxy-AL	2.549	3.053	3.386	4.272	5.315
Aprx Ratio, 90%, Proxy-Hash-AL	1.438	1.456	1.504	1.567	2.120
Aprx Ratio, max, Proxy-AL	5.836	5.958	8.851	13.501	22.921
Aprx Ratio, max, Proxy-Hash-AL	1.763	1.757	1.891	1.969	2.973
Global Obj, Proxy-AL	0.993	0.991	0.993	0.986	0.990
Global Obj, Proxy-Hash-AL	1.012	1.013	1.018	1.013	1.012

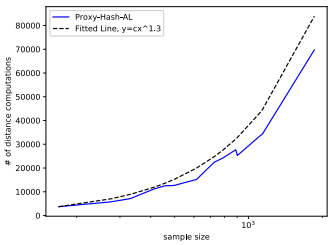

Figure 2: Growth of distance computation, Proxy-Hash-AL, New York

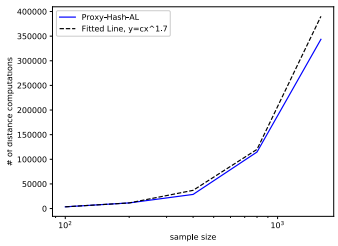

Figure 3: Growth of distance computation, Proxy-Hash-AL, Seizure

better than building a tree directly from the proxy metric. Naively building the MST on the proxy metric has poor performance, over 50% degradation in quality. At the same time Proxy-Hash-SL gives much better results, with only 1-3% loss at $\epsilon = 0.1$ to around 7-10% at $\epsilon = 0.5$. This result is independent of the graph size, but lower ϵ values in Proxy-Hash-SL lead to higher quality results. See the Supplementary Materials for more discussion about the impact of sample size and ϵ on the performance of Proxy-Hash-SL. We note that in general the performance is robust to small changes in ϵ.

The results extend to average linkage. In Table 1 and 5, note that both Proxy-AL and Proxy-Hash-AL perform well for the global objective in Cohen-Addad *et al.* (8). The degradation is negligible (about 1%) compared to the real average linkage tree. However, the statistics of approximation ratios show that on both datasets, Proxy-Hash-AL often beats Proxy-AL on both data sets in mean and 90% percentile of all approximation ratios. On `seizure`, the advantage is more apparent than in road maps. Especially, Proxy-Hash-AL has a significant advantage overProxy-ALin worst-case approximation ratios. This shows Proxy-Hash-AL makes decisions which are similar to true average linkage. The quality of its decision is stable and robust against large distortion between the proxy and real distances.

Running Time Analysis. Next, we look at the speed of our algorithms. The main bottleneck in all of the approaches is the number of distance computations. For the naive algorithm, we must compute distances for all n^2 pairs of nodes, resulting in a quadratic running time.

Figure 2 and 3 show the growth of number of distance computations as sample size grows. In both figures, sample sizes (the x-axis) are plotted on a log-scale for better visualization. For both data sets, we draw another "benchmark" polynomial curve $y = c \cdot x^{1+\rho}$ to show that the growth is strictly sub-quadratic. The plots show that for road map and `seizure`, the ρ value is bounded by 0.7 and 0.3, respectively. The gain in running time might depend on the data set. The running time plots for Proxy-Hash-SL and the road maps in Bay Area and the Great Lakes can be found in the supplementary materials.

6 Conclusion

In this work we presented algorithms that use approximate nearest neighbor data structures to speed up single and average linkage algorithms to run in sub-quadratic time. In addition we showed how to effectively use proxy metrics to achieve a tradeoff between accuracy and running time when ANN data structures are not available for general metrics. We complemented our theoretical exploration with empirical results demonstrating the efficacy of our methods.

Many interesting questions remain, among them extending this analysis to complete linkage, or more generally using proxy metrics in combination with approximate nearest neighbor data structure to speed up other algorithms.

7 Acknowledgements

B. Moseley and Y. Wang were supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.

References

[1] Abboud, Amir, Cohen-Addad, Vincent, & Houdrougé, Hussein. 2019. Subquadratic High-Dimensional Hierarchical Clustering. *Pages 11576–11586 of: Advances in Neural Information Processing Systems*.

[2] Ahmadian, Sara, Chatziafratis, Vaggos, Epasto, Alessandro, Lee, Euiwoong, Mahdian, Mohammad, Makarychev, Konstantin, & Yaroslavtsev, Grigory. 2020. Bisect and Conquer: Hierarchical Clustering via Max-Uncut Bisection. *In: AISTATS*.

[3] Andoni, Alexandr, & Indyk, Piotr. 2008. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, **51**(1), 117–122.

[4] Borodin, Allan, Ostrovsky, Rafail, & Rabani, Yuval. 1999. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Pages 435–444 of: Proceedings of the thirty-first annual ACM symposium on Theory of computing*.

[5] Calinescu, Gruia, Chekuri, Chandra, PÃ¡l, Martin, & VondrÃ¡k, Jan. 2011. Maximizing a Monotone Submodular Function Subject to a Matroid Constraint. *SIAM Journal on Computing*, **40**(6), 1740–1766.

[6] Charikar, Moses, & Chatziafratis, Vaggos. 2017. Approximate Hierarchical Clustering via Sparsest Cut and Spreading Metrics. *Pages 841–854 of: SODA*.

[7] Chierichetti, Flavio, Kumar, Ravi, & Mahdian, Mohammad. 2014. The complexity of LSH feasibility. *Theor. Comput. Sci.*, **530**, 89–101.

[8] Cohen-Addad, Vincent, Kanade, Varun, Mallmann-Trenn, Frederik, & Mathieu, Claire. 2018. Hierarchical Clustering: Objective Functions and Algorithms. *Pages 378–397 of: SODA*.

[9] Dasgupta, Sanjoy. 2016. A cost function for similarity-based hierarchical clustering. *Pages 118–127 of: STOC*.

[10] Dasgupta, Sanjoy, & Gupta, Anupam. 2003. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, **22**(1), 60–65.

[11] Datar, Mayur, Immorlica, Nicole, Indyk, Piotr, & Mirrokni, Vahab S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. *Pages 253–262 of: Proceedings of the twentieth annual symposium on Computational geometry*.

[12] Indyk, Piotr, & Matousek, Jiri. 2004. Low-Distortion Embeddings of Finite Metric Spaces. *Pages 177–196 of: in Handbook of Discrete and Computational Geometry*. CRC Press.

[13] Indyk, Piotr, & Motwani, Rajeev. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. *Pages 604–613 of: Proceedings of the thirtieth annual ACM symposium on Theory of computing*.

[14] Moseley, Benjamin, & Wang, Joshua. 2017. Approximation Bounds for Hierarchical Clustering: Average Linkage, Bisecting K-means, and Local Search. *Pages 3094–3103 of: NIPS*.