Affinity Clustering: Hierarchical Clustering at Scale

MohammadHossein Bateni
Google Research
bateni@google.com

Soheil Behnezhad*
University of Maryland
soheil@cs.umd.edu

Mahsa Derakhshan*
University of Maryland
mahsaa@cs.umd.edu

MohammadTaghi Hajiaghayi*
University of Maryland
hajiagha@cs.umd.edu

Raimondas Kiveris
Google Research
rkiveris@google.com

Silvio Lattanzi
Google Research
silviol@google.com

Vahab Mirrokni
Google Research
mirrokni@google.com

Abstract

Graph clustering is a fundamental task in many data-mining and machine-learning pipelines. In particular, identifying a good hierarchical structure is at the same time a fundamental and challenging problem for several applications. The amount of data to analyze is increasing at an astonishing rate each day. Hence there is a need for new solutions to efficiently compute effective hierarchical clusterings on such huge data.

The main focus of this paper is on minimum spanning tree (MST) based clusterings. In particular, we propose *affinity*, a novel hierarchical clustering based on Borůvka's MST algorithm. We prove certain theoretical guarantees for affinity (as well as some other classic algorithms) and show that in practice it is superior to several other state-of-the-art clustering algorithms.

Furthermore, we present two MapReduce implementations for affinity. The first one works for the case where the input graph is dense and takes constant rounds. It is based on a *Massively Parallel* MST algorithm for dense graphs that improves upon the state-of-the-art algorithm of Lattanzi *et al.* [34]. Our second algorithm has no assumption on the density of the input graph and finds the affinity clustering in $\mathrm{O}(\log n)$ rounds using Distributed Hash Tables (DHTs). We show experimentally that our algorithms are scalable for huge data sets, e.g., for graphs with trillions of edges.

1 Introduction

Clustering is a classic unsupervised learning problem with many applications in information retrieval, data mining, and machine learning. In hierarchical clustering the goal is to detect a nested hierarchy of clusters that unveils the full clustering structure of the input data set. In this work we study the hierarchical clustering problem on real-world graphs. This problem has received a lot of attention in recent years [13, 16, 41] and new elegant formulations and algorithms have been introduced. Nevertheless many of the newly proposed techniques are sequential, hence difficult to apply on large data sets.

*Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

With the constant increase in the size of data sets to analyze, it is crucial to design efficient large-scale solutions that can be easily implemented in distributed computing platforms (such as Spark [45] and Hadoop [43] as well as MapReduce and its extension Flume [17]), and cloud services (such as Amazon Cloud or Google Cloud). For this reason in the past decade several papers proposed new distributed algorithms for classic computer science and machine learning problems [3, 4, 7, 14, 15, 19]. Despite these efforts not much is known about distributed algorithms for hierarchical clustering. There are only two works analyzing these problems [27, 28], and neither gives any theoretical guarantees on the quality of their algorithms or on the round complexity of their solutions.

In this work we propose new parallel algorithms in the MapReduce model to compute hierarchical clustering and we analyze them from both theoretical and experimental perspectives. The main idea behind our algorithms is to adapt clustering techniques based on classic minimum spanning tree algorithms such as Borůvka's algorithm [11] and Kruskal's algorithm [33] to run efficiently in parallel. Furthermore we also provide a new theoretical framework to compare different clustering algorithms based on the concept of a "certificate" and show new interesting properties of our algorithms.

We can summarize our contribution in four main points.

First, we focus on the distributed implementations of two important clustering techniques based on classic minimum spanning tree algorithms. In particular we consider linkage-based clusterings inspired by Kruskal's algorithm and a novel clustering called *affinity clustering* based on Borůvka's algorithm. We provide new theoretical frameworks to compare different clustering algorithms based on the concept of a "certificate" as a proof of having a good clustering and show new interesting properties of both affinity and single-linkage clustering algorithms.

Then, using a connection between linkage-based clustering, affinity clustering and the minimum spanning tree problem, we present new efficient distributed algorithms for the hierarchical clustering problem in a MapReduce model. In our analysis we consider the most restrictive model for distributed computing, called *Massively Parallel Communication*, among previously studied MapReduce-like models [10, 23, 30]. Along the way, we obtain a constant round MapReduce algorithm for minimum spanning tree (MST) of dense graphs (in Section 5). Our algorithm for graphs with $\Theta(n^{1+c})$ edges and for any given ϵ with $0 < \epsilon < c < 1$, finds the MST in $\lceil \log(c/\epsilon) \rceil + 1$ rounds using $\tilde{\mathrm{O}}(n^{1+\epsilon})$ space per machine and $O(n^{c-\epsilon})$ machines (i.e., optimal total space). This improves the round complexity of the state-of-the-art MST algorithm of Lattanzi *et al.* [34] for dense graphs which requires up to $\lceil c/\epsilon \rceil$ rounds using the same number of machines and space. Prior to our work, no hierarchical clustering algorithm was known in this model.

Then we turn our attention to real world applications and we introduce efficient implementations of affinity clustering as well as classic single-linkage clustering that leverage Distributed Hash Tables (DHTs) [12, 31] to speed up computation for huge data sets.

Last but not least, we present an experimental study where we analyze the scalability and effectiveness of our newly introduced algorithms and we observe that, in most cases, affinity clustering outperforms all state-of-the-art algorithms from both quality and scalability standpoints.[2]

2 Related Work

Clustering and, in particular, hierarchical clustering techniques have been studied by hundreds of researchers [16, 20, 22, 32]. In social networks, detecting the hierarchical clustering structure is a basic primitive for studying the interaction between nodes [36, 39]. Other relevant applications of hierarchical clustering can be found in bioinformatics, image analysis and text classification.

Our paper is closely related to two main lines of research. The first one focuses on studying theoretical properties of clustering approaches based on minimum spanning trees (MSTs). Linkage-based clusterings (often based on Kruskal's algorithm) have been extensively studied as basic techniques for clustering datasets. The most common linkage-based clustering algorithms are single-linkage, average-linkage and complete-linkage algorithms. In [44], Zadeh and Ben-David gave a characterization of the single-linkage algorithm. Their result has been then generalized to linkage-based algorithms in [1]. Furthermore single-linkage algorithms are known to provably recover a ground truth clustering if the similarity function has some stability properties [6]. In this paper we

[2]Implementations are available at `https://github.com/MahsaDerakhshan/AffinityClustering`.

introduce a new technique to compare clustering algorithms based on "certificates." Furthermore we introduce and analyze a new algorithm—affinity—based on Borůvka's well-known algorithm. We show that affinity is not only scalable for huge data sets but also its performance is superior to several state-of-the-art clustering algorithms. To the best of our knowledge though Borůvka's algorithm is a well-known and classic algorithm, not many clustering algorithms have been considered based on Borůvka's.

The second line of work is closely related to distributed algorithms for clustering problems. Several models of MapReduce computation have been introduced in the past few years [10, 23, 30]. The first paper that studied clustering problems in these models is by Ene *et al.* [18], where the authors prove that any α approximation algorithm for the k-center or k-median problems can produce $4\alpha + 2$ and $10\alpha + 3$ approximation factors, respectively, for the k-center or k-median problems in the MapReduce model. Subsequently several papers [5, 7, 8] studied similar problems in the MapReduce model. A lot of efforts also went into studying efficient algorithms on graphs [3, 4, 7, 15, 14, 19]. However the problem of hierarchical clustering did not receive a lot of attention. To the best of our knowledge there are only two papers [27, 28] on this topic, and neither analyzes the problem formally or proves any guarantee in any MapReduce model.

3 Minimum Spanning Tree-Based Clusterings

We begin by going over two famous algorithms for minimum spanning tree and define the corresponding algorithms for clustering.

Borůvka's algorithm and affinity clustering: *Borůvka's algorithm* [11], first published in 1926, is an algorithm for finding a minimum spanning tree (MST)[3]. The algorithm was rediscovered a few times, in particular by Sollin [42] in 1965 in the parallel computing literature. Initially each vertex forms a group (cluster) by itself. The algorithm begins by picking the cheapest edge going out of each cluster, in each *round* (in parallel) joins these clusters to form larger clusters and continues joining in a similar manner until a tree spanning all vertices is formed. Since the size of the smallest cluster at least doubles each time, the number of rounds is at most $O(\log n)$. In *affinity clustering*, we stop Borůvka's algorithm after $r > 0$ rounds when for the first time we have at most k clusters for a desired number $k > 0$. In case the number of clusters is strictly less than k, we delete the edges that we added in the last round in a non-increasing order (i.e., we delete the edge with the highest weight first) to obtain exactly k clusters. To the best of our knowledge, although Borůvka's algorithm is a well-known and classic algorithm, clustering algorithms based on it have not been considered much. A natural hierarchy of nodes can be obtained by continuing Borůvka's algorithm: each cluster here will be a subset of future clusters. We call this *hierarchical affinity clustering*.

We present distributed implementations of Borůvka/affinity in Section 5 and show its scalability even for huge graphs. We also show affinity clustering, in most cases, works much better than several well-known clustering algorithms in Section 6.

Kruskal's algorithm and single-linkage clustering: Kruskal's algorithm [33] first introduced in 1956 is another famous algorithm for finding MST. The algorithm is highly sequential and iteratively picks an edge of the least possible weight that connects any two trees (clusters) in the forest.[4] Though the number of iterations in Kruskal's algorithm is $n - 1$ (the number of edges of any tree on n nodes), the algorithm can be implemented in $O(m \log n)$ time with simple data structures (m is the number of edges) and in $O(ma(n))$ time using a more sophisticated disjoint-set data structure, where $a(.)$ is the extremely slowly growing inverse of the single-valued Ackermann function.

In *single-linkage clustering*, we stop Kruskal's algorithm when we have at least k clusters (trees) for a desired number $k > 0$. Again if we desire to obtain a corresponding *hierarchical single-linkage clustering*, by adding further edges which will be added in Kruskal's algorithm later, we can obtain a natural hierarchical clustering (each cluster here will be a subset of future clusters).

As mentioned above, Kruskal's Algorithm and single-linkage clustering are highly sequential, however as we show in Section 5 thinking backward once we have an efficient implementation of Borůvka's

[3] More precisely the algorithm works when there is a unique MST, in particular, when all edge weights are distinct; however this can be easily achieved by either perturbing the edge weights by an $\epsilon > 0$ amount or have a tie-breaking ordering for edges with the same weights

[4] Unlike Borůvka's method, this greedy algorithm has no limitations on the distinctness of edge weights.

(or any MST algorithm) in Map-Reduce and using Distributed Hash Tables (DHTs), we can achieve an efficient parallel implementation of single-linkage clustering as well. We show scalability of this implementation even for huge graphs in Section 5 and its performance in experiments in Section 6.

4 Guaranteed Properties of Clustering Algorithms

An important property of affinity clustering is that it produces clusters that are roughly of the same size. This is intuitively correct since at each round of the algorithm, each cluster is merged to at least one other cluster and as a result, the size of even the smallest cluster is at least doubled. In fact linkage based algorithms (and specially single linkage) are often criticized for producing uneven clusters; therefore it is tempting to give a theoretical guarantee for the size ratio of the clusters that affinity produces. Unfortunately, as it is illustrated in Figure 1, we cannot give any worst case bounds since even in one round we may end up having a cluster of size $\Omega(n)$ and another cluster of size $\mathrm{O}(1)$. As the first property, we show that at least in the first round, this does not happen when the observations are randomly distributed. Our empirical results on real world data sets in Section 6.1, further confirm this property for all rounds, and on real data sets.

Figure 1: An example of how affinity may produce a large component in one round.

We start by defining the nearest neighbor graph.

Definition 1 (Nearest Neighbor Graph)**.** *Let S be a set of points in a metric space. The nearest neighbor graph of S, denoted by $\mathcal{G}_S$, has $|S|$ vertices, each corresponding to an element in S and if $a \in S$ is the nearest element to $b \in S$ in S, graph $\mathcal{G}_S$ contains an edge between the corresponding vertices of a and b.*

At each round of affinity clustering, all the vertices that are in the same connected component of the nearest neighbor graph will be merged together[5]. Thus, it suffices to bound the connected components' size.

For a random model of points, consider a *Poisson point process* X in $\mathbb{R}^d$ ($d \geq 1$) with density 1. It has two main properties. First, the number of points in any finite region of volume V is Poisson distributed with mean V. Second, the number of points in any two disjoint regions are independent of each other.

Theorem 1 (Häggström *et al.* [38])**.** *For any $d \geq 2$, consider the (Euclidean distance) nearest neighbor graph $\mathcal{G}$ of a realization of a Poisson point process in $\mathbb{R}^d$ with density 1. All connected components of $\mathcal{G}$ are finite almost surely.*

Theorem 1 implies that the size of the maximum connected component of the points within any finite region in $\mathbb{R}^d$ is bounded by almost a constant number. This is a very surprising result compared to the worst case scenario of having a connected component that contains all the points.

Note that although the aforementioned bound holds for the first round of affinity, after the connected components are contracted, we cannot necessarily assume that the new points are Poisson distributed and the same argument cannot be used for the rest of the rounds.

Next we present further properties of affinity clustering. Let us begin by introducing the concept of "cost" for a clustering solution to be able to compare clustering algorithms.

Definition 2. *The* cost of a cluster *is the sum of edge lengths (weights) of a minimum Steiner tree connecting all vertices inside the cluster. The* cost of a clustering *is the sum of the costs of its clusters. Finally a* non-singleton clustering *of a graph is a partition of its vertices into clusters of size at least two.*

Even one round of affinity clustering often produces good solutions for several applications. Now we are ready to present the following extra property of the result of the first round of affinity clustering.

[5]Depending on the variant of affinity that we use, the distance function will be updated.

Theorem 2. *The cost of any non-singleton clustering is at least half of that of the clustering obtained after the first round of affinity clustering.*

Before presenting the proof of Theorem 2, we need to demonstrate the concept of *disc painting* introduced previously in [29, 2, 21, 9, 25]. In this setting, we consider a topological structure of a graph metric in which each edge is a curve connecting its endpoints whose length is equal to its weight. We assume each vertex has its own color. A *disc painting* is simply a set of disjoint disks centered at terminals (with the same colors of the center vertices). A disk of *radius* r centered at vertex v paints all edges (or portions) of them which are at distance r from vertex v with the color of v. Thus we paint (portions of) edges by different disks each corresponding to a vertex and each edge can be painted by at most two disks. With this definition of disk painting, we now demonstrate the proof of Theorem 2.

Next we turn our focus to obtain structural properties for single-linkage clustering. We denote by F_k the set of edges added after k iterations of Kruskal, i.e., when we have $n-k$ clusters in single-linkage clustering. Note that F_k is a forest, i.e., a set of edges with no cycle. First we start with an important observation whose proof comes directly from the description of the single-linkage algorithm.

Proposition 3. *Suppose we run single-linkage clustering until we have $n-k$ clusters. Let d_{outside} be the minimum distance between any two clusters and d_{inside} be the maximum distance of any edge added to forest F_k. Then $d_{\text{outside}} \geq d_{\text{inside}}$.*

We note that Proposition 3 demonstrates the following important property of single-linkage clustering: *Each vertex of a cluster at any time has a neighbor inside to which is closer than any other vertex outside of its clusters.*

Next we define another criterion for desirability of a clustering algorithm. This generalizes Proposition 3.

Definition 3. *An α-certificate for a clustering algorithm, where $\alpha \geq 1$, is an assignment of* shares *to each vertex of the graph with the following two properties: (1) The cost of each cluster is at most α times the sum of shares of vertices inside the cluster; (2) For any set S of vertices containing at most one from each cluster in our solution, the imaginary cluster S costs at least the sum of shares of vertices in S.*

Note that intuitively the first property guarantees that vertices inside each cluster can pay the cost of their corresponding cluster and that there is no free-rider. The second property intuitively implies we cannot find any better clustering by combining vertices from different clusters in our solution.

Next we show that there always exists a 2-certificate for single-linkage clustering guaranteeing its worst-case performance.

Theorem 4. *Single-linkage always produces a clustering solution that has a 2-certificate.*

5 Distributed Algorithms

5.1 Constant Round Algorithm For Dense Graphs

Unsurprisingly, finding the affinity clustering of a given graph G is closely related to the problem of finding its Minimum Spanning Tree (MST). In fact, we show the data that is encoded in the MST of G is sufficient for finding its affinity clustering (Theorem 9). This property is also known to be true for single linkage [24]. For MapReduce algorithms this is particularly useful because the MST requires a substantially smaller space than the original graph and can be stored in one machine. Therefore, once we have the MST, we can obtain affinity or single linkage in one round.

The main contribution of this section is an algorithm for finding the MST (and therefore the affinity clustering) of dense graphs in constant rounds of MapReduce which improves upon prior known dense graph MST algorithms of Karloff *et al.* [30] and Lattanzi *et al.* [34].

Theoretical Model. Let N denote the input size. There are a total number of M machines and each of them has a space of size S. Both S and M must be substantially sublinear in N. In each round, the machines can run an arbitrary polynomial time algorithm on their local data. No communication is allowed during the rounds but any two machines can communicate with each other between the rounds as long as the total communication size of each machine does not exceed its memory size.

Algorithm 1 MST of Dense Graphs

Input: A weighted graph G
Output: The minimum spanning tree of G

function MST($G = (V, E)$, ϵ)
  $c \leftarrow \log_n (m/n)$ ▷ Since G is assumed to be dense we know $c > 0$.
  while $|E| > \mathrm{O}(n^{1+\epsilon})$ **do**
    REDUCEEDGES(G, c)
    $c \leftarrow (c - \epsilon)/2$
  Move all the edges to one machine and find MST of G in there.
function REDUCEEDGES($G = (V, E)$, c)
  $k \leftarrow n^{(c-\epsilon)/2}$
  Independently and u.a.r. partition V into k subsets $\{V_1, \ldots, V_k\}$.
  Independently and u.a.r. partition V into k subsets $\{U_1, \ldots, U_k\}$.
  Let $G_{i,j}$ be a subgraph of G with vertex set $V_i \cup U_j$ containing any edge $(v, u) \in E(G)$ where $v \in V_i$ and $u \in U_j$.
  for any $i, j \in \{1, \ldots, k\}$ **do**
    Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.
    Remove an edge e from $E(G)$, if $e \in G_{i,j}$ and it is not in MST of $G_{i,j}$.

This model is called Massively Parallel Communication ($\mathcal{MPC}$) in the literature and is "arguably the most popular one" [26] among MapReduce like models.

Theorem 5. *Let $G = (V, E)$ be a graph with n vertices and n^{1+c} edges for any constant $c > 0$ and let $w : E \mapsto \mathbb{R}^+$ be its edge weights. For any given ϵ such that $0 < \epsilon < c$, there exists a randomized algorithm for finding the MST of G that runs in at most $\lceil \log (c/\epsilon) \rceil + 1$ rounds of $\mathcal{MPC}$ where every machine uses a space of size $\tilde{\mathrm{O}}(n^{1+\epsilon})$ with high probability and the total number of required machines is $\mathrm{O}(n^{c-\epsilon})$.*

Our algorithm, therefore, uses only enough total space ($\tilde{\mathrm{O}}(n^{1+c})$) on all machines to store the input.

The following observation is mainly used by Algorithm 1 to iteratively remove the edges that are not part of the final MST.

Lemma 6. *Let $G' = (V', E')$ be a (not necessarily connected) subgraph of the input graph G. If an edge $e \in E'$ is not in the MST of G', then it is not in the MST of G either.*

To be more specific, we iteratively divide G into its subgraphs, such that each edge of G is at least in one subgraph. Then, we handle each subgraph in one machine and throw away the edges that are not in their MST. We repeat this until there are only $\mathrm{O}(n^{1+\epsilon})$ edges left in G. Then we can handle all these edges in one machine and find the MST of G. Algorithm 1 formalizes this process.

Lemma 7. *Algorithm 1 correctly finds the MST of the input graph in $\lceil \log (c/\epsilon) \rceil + 1$ rounds.*

By Lemma 6 we know any edge that is removed from is not part of the MST therefore it suffices to prove the while loop in Algorithm 1 takes $\lceil \log (c/\epsilon) \rceil + 1$ iterations.

Lemma 8. *In Algorithm 1, every machine uses a space of size $\tilde{\mathrm{O}}(n^{1+\epsilon})$ with high probability.*

The combination of Lemma 7 and Lemma 8 implies that Algorithm 1 is indeed in $\mathcal{MPC}$ and Theorem 5 holds. See supplementary material for omitted proofs.

The next step is to prove all the information that is required for affinity clustering is indeed contained in the MST.

Theorem 9. *Let $G = (V, E)$ denote an arbitrary graph, and let $G' = (V, E')$ denote the minimum spanning tree of G. Running affinity clustering algorithm on G gives the same clustering of V as running this algorithm on G'.*

By combining the MST algorithm given for Theorem 5 and the sufficiency of MST for computing affinity clustering (Theorem 9) and single linkage ([24]) we get the following corollary.

Corollary 10. *Let $G = (V, E)$ be a graph with n vertices and n^{1+c} edges for any constant $c > 0$ and let $w : E \mapsto \mathbb{R}^+$ be its edge weights. For any given ϵ such that $0 < \epsilon < c$, there exists a*

randomized algorithm for affinity clustering and single linkage that runs in $\lceil \log(c/\epsilon) \rceil + 1$ *rounds of* $\mathcal{MPC}$ *where every machine uses a space of size* $\tilde{\mathrm{O}}(n^{1+\epsilon})$ *with high probability and the total number of required machines is* $\mathrm{O}(n^{c-\epsilon})$.

5.2 Logarithmic Round Algorithm For Sparse Graphs

Consider a graph $G(V, E)$ on $n = |V|$ vertices, with edge weights $w : E \mapsto \mathbb{R}$. We assume that the edge weights denote distances. (The discussion applies *mutatis mutandis* to the case where edge weights signify similarity.)

The algorithm works for a fixed number of synchronous rounds, or until no further progress is made, say, by reaching a single cluster of all vertices. Each round consists of two steps: First, every vertex picks its *best* edge (i.e., that of the minimum weight) at each round; and then the graph is contracted along the selected edges. (See Algorithm 2 in the appendix.)

For a connected graph, the algorithm continues until a single cluster of all vertices is obtained. The supernodes at different rounds can be thought of as a hierarchical clustering of the vertices.

While the first step of each round has a trivial implementation in MapReduce, the latter might take $\Omega(\log n)$ MapReduce rounds to implement, as it is an instance of the connected components problem. Using a DHT was shown to significantly improve the running time here, by implementing the operation in one round of MapReduce [31]. Basically we have a read-only random-access table mapping each vertex to its best neighbor. Repeated lookups in the table allows each vertex to follow the chain of best neighbors until a loop (of length two) is encountered. This assigns a unique name for each connected component; then all the vertices in the same component are reduced into a supernode.

Theorem 11. *The affinity clustering algorithm runs in* $O(\log n)$ *rounds of MapReduce when we have access to a distributed hash table (DHT). Without the DHT, the algorithm takes* $O(\log^2 n)$ *rounds.*

6 Experiments

6.1 Quality Analysis

In this section, we compare well known hierarchical and flat clustering algorithms, such as k-means, single linkage, complete linkage and average linkage with different variants of affinity clustering, such as single affinity, complete affinity and average affinity. We run our experiments on several data sets from the UCI database [37] and use Euclidean distance[6].

To evaluate the outputs of these algorithms we use Rand index which is defined as follows.

Definition 4 (Rand index [40])**.** *Given a set* $V = \{v_1, \ldots, v_n\}$ *of* n *points and two clusterings* $X = \{X_1, \ldots, X_r\}$ *and* $Y = \{Y_1, \ldots, Y_s\}$ *of* V*. Define the following.*

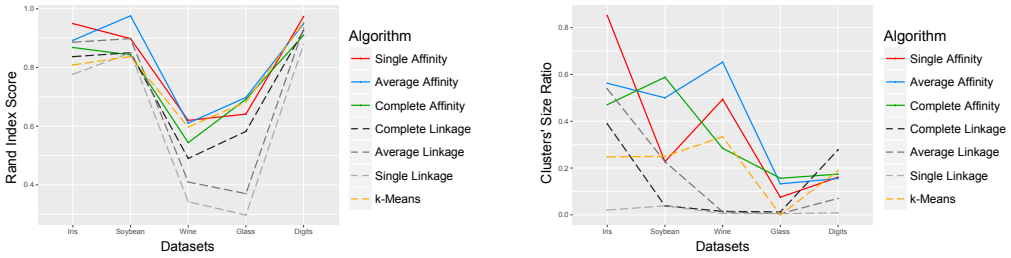
- a*: the number of pairs in* V *that are in the same cluster in* X *and in the same cluster in* Y*.*
- b*: the number of pairs in* V *that are in different clusters in* X *and in different clusters in* Y*.*

the Rand index $r(X, Y)$ *is defined to be* $(a + b)/\binom{n}{2}$*. By having the ground truth clustering* T *of a data set, we define the Rand index score of a clustering* X*, to be* $r(X, T)$*.*

The Rand index based scores are in range $[0, 1]$ and a higher number implies a better clustering. For a hierarchical clustering, the level of its corresponding tree with the highest score is used for evaluations.

Figure 2 (a) compares the Rand index score of different clustering algorithms for different data sets. We observe that single affinity generally performs really well and is among the top two algorithms for most of the datasets (all except Glass). Average affinity also seems to perform well and in some cases (e.g., for Soybean data set) it produces a very high quality clustering compared to others. To summarize, linkage based algorithms do not seem to be as good as affinity based algorithms but in some cases k-means could be close.

[6] We consider Iris, Wine, Soybean, Digits and Glass data sets.



(a) (b)

Figure 2: Comparison of clustering algorithms based on their Rand index score (a) and clusters size ratio (b).

Table 1: Statistics about datasets used. (Numbers for ImageGraph are approximate.) The fifth column shows the relative running time of affinity clustering, and the last column is the speedup obtained by a ten-fold increase in parallelism.

Dataset	# nodes	# edges	max degree	running time	speedup
LiveJournal	4,846,609	7,861,383,690	444,522	1.0	4.3
Orkut	3,072,441	42,687,055,644	893,056	2.4	9.2
Friendster	65,608,366	1,092,793,541,014	2,151,462	54	5.9
ImageGraph	2×10^{10}	10^{12}	14000	142	4.1

Another property of the algorithms that we measure is the clusters' *size ratio*. Let $X = \{X_1, \ldots, X_r\}$ be a clustering. We define the size ratio of X to be $\min_{i,j \in [r]} |X_i|/|X_j|$. As it is visualized in Figure 2 (b), affinity based algorithms have a much higher size ratio (i.e., the clusters are more balanced) compared to linkage based algorithms. This confirms the property that we proved for Poisson distributions in Section 4 for real world data sets. Hence we believe affinity clustering is superior to (or at least as good as) the other methods when the dataset under consideration is not extremely unbalanced.

6.2 Scalability

Here we demonstrate the scalability of our implementation of affinity clustering. A collection of public and private graphs of varying sizes are studied. These graphs have between 4 million and 20 billion vertices and from 4 billion to one trillion edges. The first three graphs in Table 1 are based on public graphs [35]. As most public graphs are unweighted, we use the number of common neighbors between a pair of nodes as the weight of the edge between them. (This is computed for all pairs, whether they form a pair in the original graph or not, and then new edges of weight zero are removed.) The last graph is based on (a subset of) an internal corpus of public images found on the web and their similarities.

We note that we use the "maximum" spanning tree variant of affinity clustering; here edge weights denote similarity rather than distance.

While we cannot reveal the exact running times and number of machines used in the experiments, we report these quantities in "normalized form." We only run one round of affinity clustering (consisting of a "Find Best Neighbors" and a "Contract Graph" step). Two settings are used in the experiments. We once use W MapReduce workers and D machines for the DHT, and compare this to the case with $10W$ MapReduce workers and D machines for the DHT. This ten-fold increase in the number of MapReduce workers leads to four- to ten-fold decrease in the total running time for different datasets. Each running time is itself the average over three runs to reduce the effect of external network events.

Table 1 also shows how the running time changes with the size of the graph. With a modest number of MapReduce workers, affinity clustering runs in less than an hour for all the graphs.

References

[1] Margareta Ackerman, Shai Ben-David, and David Loker. Characterization of linkage-based clustering. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, pages 270–281, 2010.

[2] Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012.

[4] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 574–583. ACM, 2014.

[5] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.

[6] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 671–680, 2008.

[7] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1995–2003, 2013.

[8] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab S. Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2591–2599, 2014.

[9] MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21:1–21:37, 2011.

[10] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 273–284. ACM, 2013.

[11] Oktar Boruvka. *O jistém problému minimálním*. Práce Moravské přírodovědecké společnosti. Mor. přírodovědecká společnost, 1926.

[12] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008.

[13] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 841–854, 2017.

[14] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1326–1344, 2016.

[15] Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1234–1251, 2015.

[16] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 118–127, 2016.

[17] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689, 2011.

[19] Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1217–1233, 2015.

[20] Assaf Glazer, Omer Weissbrod, Michael Lindenbaum, and Shaul Markovitch. Approximating hierarchical mv-sets for hierarchical clustering. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 999–1007, 2014.

[21] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[22] Jacob Goldberger and Sam T. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 505–512, 2004.

[23] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

[24] John C Gower and GJS Ross. Minimum spanning trees and single linkage cluster analysis. *Applied statistics*, pages 54–64, 1969.

[25] Mohammad Taghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 558–567, 2013.

[26] Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, 2017.

[27] Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. A scalable hierarchical clustering algorithm using spark. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 418–426, 2015.

[28] Chen Jin, Md Mostofa Ali Patwary, Ankit Agrawal, William Hendrix, Wei-keng Liao, and Alok Choudhary. Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce. In *Proceedings of the 4th International SC Workshop on Data Intensive Computing in the Clouds*, 2013.

[29] Michael Jünger and William R. Pulleyblank. New primal and dual matching heuristics. *Algorithmica*, 13(4):357–386, 1995.

[30] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

[31] Raimondas Kiveris, Silvio Lattanzi, Vahab S. Mirrokni, Vibhor Rastogi, and Sergei Vassilvitskii. Connected components in MapReduce and beyond. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 03 - 05, 2014*, pages 18:1–18:13, 2014.

[32] Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[33] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[34] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 85–94. ACM, 2011.

[35] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[36] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press, 2014.

[37] Moshe Lichman. UCI machine learning repository, 2013.

[38] Ronald Meester et al. Nearest neighbor and hard sphere models in continuum percolation. *Random structures and algorithms*, 9(3):295–315, 1996.

[39] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[40] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[41] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2316–2324, 2016.

[42] M. Sollin. Le tracé de canalisation. *Programming, Games, and Transportation Networks (in French)*, 1965.

[43] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[44] Reza Zadeh and Shai Ben-David. A uniqueness theorem for clustering. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 639–646, 2009.

[45] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, 2010.

A Appendix

A.1 Section 4

Proof of Theorem 2. For a vertex v of the graph, let v_{nearest} be its nearest neighbor in the first iteration of affinity clustering. Now for each vertex v of the graph, we paint a disk of radius $\text{distance}(v, v_{\text{nearest}})/2$ with its color v. By definition, all disks will be disjoint (i.e., paint different parts of the graph). As a result, the union of Steiner trees corresponding to any non-singleton clustering must intersect each disk at least to the extend of the disk's radius. Therefore, the total length of the union of Steiner trees is at least $\sum_{v \in V(G)} \text{distance}(v, v_{\text{nearest}})/2$ which is half the cost of the affinity clustering just after the first round. □

Proof of Theorem 4. First we start with the notion of *moat painting* which is a generalization of disk painting described above. A *moat* is a generalization of a disk which surrounds a set of vertices and its color is that of one of the vertices in this set. The *depth of a moat* generalizes the concept of the radius of a disk and is the minimum length that any path should traverse from outside the moat to inside the moat. Bateni, Hajiaghayi, and Marx [9] introduce the concept of *prize-collecting clustering* which itself builds on primal-dual algorithms of Goemans and Williamson [21] and Agrawal, Klein and Ravi [2]. It is also worth mentioning that the idea of disc painting and its intuitive analysis predates the modern primal-dual analysis and it is originally introduced by Jünger and Pulleyblank [29]. Via prize-collecting clustering Bateni *et al.* build a moat painting (with disjoint moats) for clusters generated by single-linkage clustering since single-linkage clustering is a special case of their prize-collecting clustering (with infinite penalties for all vertices). The share of each vertex in our certificate is the sum of depths of the moats surrounding the vertex with the same color as the vertex itself.

Now the factor two in Property 1 of the certificate indeed comes directly from factor two in the approximation guarantee of Bateni et al. [9].

Proof of Property 2 of our certificate is Similar to that of Theorem 2. Disjointness of moats implies that any Steiner tree of a set S of vertices must intersect each moat at least to the extent of its depth, and these moats paint disjoint portions of edges in any Steiner tree of set S. Thus the cost of (an imaginary cluster) S is at least the sum of the depths of moats with colors of any vertex in S which in turn is equal to the sum of the shares of the vertices in S. This proves Property 2 of our certificate. □

A.2 Section 5.1

Proof of Lemma 6. It is a known fact that any graph with distinct edge weights has a unique MST. Let M denote the MST of G and let M' denote the MST of G'. For the sake of contradiction, assume that there exists an edge $e = (u, v) \in M$ that is not in M'. Since $e \notin M'$, there is a path p from v to u in M' where $w(e') < w(e)$ for any $e' \in p$. Removing e from M partitions it into two connected components, one of which contains vertex u and the other one contains vertex v. We already know that path p connects u to v, so it has at least an edge e' that connects these two components. Since $w(e') < w(e)$, we can replace e with e' in M to obtain a better MST for G, which is a contradiction. □

Proof of Lemma 7. Let c_r denote the value of variable c at the r-th iteration (round) of the while loop in Algorithm 1. At round r, the algorithm independently and uniformly at random partitions the vertices of G into two partition sets $U = \{U_1, \ldots, U_{k_r}\}$, and $V = \{V_1, \ldots, V_{k_r}\}$, where $k_r = n^{(c_r - \epsilon)/2}$. Then, for any $i, j \in \{1, .., k_r\}$, it finds the MST of $G_{i,j}$, which we denote by $T_{i,j}$, and removes any edge in $G_{i,j}$ that is not in $T_{i,j}$. By Lemma 6, none of the removed edges are part of the solution. Therefore if the algorithm terminates, the output is indeed correct.

Now it suffices to prove the algorithm terminates in $\lceil \log(c/\epsilon) \rceil + 1$ rounds.

We first prove the total number of edges that will be kept by the end of the r-th round is at most $k_r n$. The key observation is that the MST of a subgraph with n' vertices has at most $n' - 1$ edges. This allows us to charge each edge of $T_{i,j}$ to one of the vertices of $G_{i,j}$ such that no two edges of $T_{i,j}$ are charged to the same vertex. Fix any vertex v and assume it is in set V_i. Vertex v is charged by at most k_r edges since v is only in $G_{i,1}, G_{i,2}, \ldots, G_{i,k_r}$. This means all the vertices combined are charged by at most $k_r n$ edges and hence $k_r n$ is an upper bound for the number of edges that are left by the end of the r-th round. Recall that $k_r = n^{(c_r - \epsilon)/2}$, hence $k_r n = n^{1 + (c_r - \epsilon)/2}$. On the other hand, note that $c_r < c/2^r$ since $c_0 = c$ and $c_r = (c_{r-1} - \epsilon)/2$. This implies that in round $\lceil \log(c/\epsilon) \rceil$,

$$c_{\lceil \log(c/\epsilon) \rceil} < \frac{c}{2^{\lceil \log(c/\epsilon) \rceil}} <= \epsilon.$$

Therefore after $\lceil \log(c/\epsilon) \rceil$ rounds at most $O(n^{1+\epsilon})$ edges are left in the graph. This is exactly the termination condition of the while loop. In the final round of the algorithm, we send all the remaining edges to one machine and find the MST of G, therefore it takes at most $\lceil \log(c/\epsilon) \rceil + 1$ rounds for the algorithm to terminate and correctly report the MST. □

We first state a lemma that will be useful in proving the forthcoming lemmas.

Lemma 12. *Let β be a constant number in $(1,2)$, and let $S = \{x_1, \ldots, x_n\}$ be a set of numbers where $\sum_{i=1}^{n} x_i = n^{\beta}$, and $\max_{i=1}^{n} x_i \leq n$. For any two constant numbers c and α where $0 < \alpha \leq b$, if we distribute members of S into n^{α} groups uniformly and independently at random, with probability at least $1 - \frac{\log n}{n^c}$, total sum of the numbers in any group is at most $\tilde{\mathrm{O}}(n^{\beta-\alpha})$.*

Proof. Let G_i denote the i-th group, and let $D_j \subset S$ denote the set of numbers such that for any $x \in D_j$, $2^{j-1} \leq x \leq 2^j$. For any i and j with $j \in \{1, \ldots, \log n\}$ and $i \in \{1, \ldots, n^{\alpha}\}$ we prove that with probability at least $1 - \frac{1}{n^c}$,

$$\sum_{x \in D_j \cap G_i} x \leq \tilde{\mathrm{O}}(n^{\beta-\alpha}).$$

It obviously holds for any j where $|D_j| \leq n^{\beta-\alpha-1}$ since $n \cdot n^{\beta-\alpha-1} = n^{\beta-\alpha}$. In addition for any j that $|D_j| > n^{\beta-\alpha-1}$, by a simple application of Chernoff bound, weith probability at least $1 - \frac{1}{n^c}$ for any G_i, $|D_j \cap G_i| \leq \frac{c|D_j|\log n}{n^{\alpha}}$. Since $|D_j| \leq \mathrm{O}(\frac{n^{\beta}}{2^j})$, and $\sum_{x \in D_j \cap G_i} x \leq 2^j \cdot |D_j \cap G_i|$, we have

$$\sum_{x \in D_j \cap G_i} \deg(v) \leq 2^j \cdot \mathrm{O}(\frac{n^{\beta} \log n}{2^j n^{\alpha}}) = \tilde{\mathrm{O}}(n^{\beta-\alpha}).$$

Therefore by Union bound with probability at least $1 - \frac{\log n}{n^c}$, for any G_i,

$$\sum_{x \in G_i} x = \sum_{j=1}^{\log n} \sum_{x \in D_j \cap G_i} x \leq \sum_{j=1}^{\log n} \tilde{\mathrm{O}}(n^{\beta-\alpha}) = \tilde{\mathrm{O}}(n^{\beta-\alpha}),$$

concluding the proof. □

Proof of Lemma 8. Let G denote the graph in an arbitrary round of the algorithm which has n^{1+c} edges. Note that in any round of the algorithm we have two sets of graph partitions, denoted by $U = \{U_1, \ldots, U_k\}$ and $V = \{V_1, \ldots, V_k\}$, where $k = n^{(c-\epsilon)/2}$. Moreover, $G_{i,j}$ is the subgraph with vertex set $V_i \cup U_i$ that contains all the edges of G with one end-point in V_i and one end-point in U_j. To prove this algorithm does not violate the space limits, we need to prove for any i and j that $G_{i,j}$ has at most $\tilde{\mathrm{O}}(n^{1+\epsilon})$ edges. We first give a bound for the total degree of vertices in one partition. Then we prove that with high probability the number of edges in any machine is $\tilde{\mathrm{O}}(n^{1+\epsilon})$. Let $\deg(v)$ denote the degree of vertex v in graph G, and let $\deg_{i,j}(v)$ denote the degree of vertex v in $G_{i,j}$.

Note that $|V| = n^{(c-\epsilon)/2}$, and the total degree of vertices in G is $\mathrm{O}(n^{1+c})$. By Lemma 12, for any partition $V_i \in V$, with probability at least $1 - \log n/n^3$,

$$\sum_{v \in V_i} \deg_i(v) = \tilde{\mathrm{O}}(n^{1-c}/n^{(c-\epsilon)/2}) = \tilde{\mathrm{O}}(n^{1-(c+\epsilon)/2}).$$

Let $\deg(V_i) := \sum_{v \in V_i} \deg(v)$ denote the sum of degrees of all the vertices in V_i. Note that for any edge $e \in G$ whose one end point is in V_i, there exists exactly one $j \in [k]$ such that e is in $G_{i,j}$. Therefore, $\sum_{j=1}^{k} |E(G_{i,j})| = \deg(V_i)$.

Since $|U| = n^{(c-\epsilon)/2}$, by Lemma 12, with probability at least $1 - \log n/n^3$, for any j,

$$|E(G_{i,j})| = \tilde{\mathrm{O}}(\frac{\deg(V_i)}{n^{(c-\epsilon)/2}}).$$

If $\deg(V_i) = \tilde{\mathrm{O}}(n^{1-(c+\epsilon)/2})$, then with probability at least $1 - \log n/n^3$, for any j, $|E(G_{i,j})| = \tilde{\mathrm{O}}(n^{1+\epsilon})$ which is the space of each machine. By an application of Union bound, probability of holding $|E(G_{i,j})| = \tilde{\mathrm{O}}(n^{1+\epsilon})$ for any i, j is

$$1 - (k+1) \cdot (\log n/n^3) \geq 1 - \log n/n^2.$$

By Lemma 7, the number of rounds of this algorithm is $\lceil \log(c/\epsilon) \rceil + 1$, therefore with probability at least $1 - (\lceil \log(c/\epsilon) \rceil + 1) \cdot \log n/n^2$ (high probability) throughout this algorithm each machine needs a space of at most $\tilde{\mathrm{O}}(n^{1+\epsilon})$. □

Proof of Theorem 9. For the sake of contradiction assume that running affinity clustering on G gives a different clustering of V than running it on G'. For any round i, and graph H let $C(i, H)$ denote the set of clusters that we have in round i of running affinity clustering algorithm on graph H. It is easy to see that $C(0, G') =$

Algorithm 2 Affinity Clustering

Input: A graph $G = (V, E)$
Output: One clustering (denoted by mapping λ_i) per level

$i \leftarrow 0$
repeat
  Find Best Neighbors for G to get λ ▷ See Algorithm 3 in the appendix.
  Contract Graph G based on λ to get G' ▷ See Algorithm 4 in the appendix.
  $G \leftarrow G'$
  $\lambda_i \leftarrow \lambda$
  $i \leftarrow i + 1$
until λ is the identity mapping

Algorithm 3 Find Best Neighbors

Input: A graph $G = (V, E)$
Output: A mapping $\lambda : V \mapsto V$
Run the following MapReduce where $N(v)$ is the set of neighbors of $v \in V$.
Map $\langle u; N(u)\rangle$

if $N(u) = \emptyset$ **then**
  Emit $\langle u; u\rangle$
else
  $v \leftarrow \arg\min_{v \in N(u)} w(u, v)$
  Emit $\langle u; v\rangle$

Reduce $\langle u; v\rangle$

Emit $\langle u; v\rangle$

$C(0, G) = V$. Let j denote the first round that $C(j, G') \neq C(j, G)$. This means that there exists at least one cluster c_1 in both $C(j-1, G')$ and $C(j-1, G)$ that its cheapest edge going out of the cluster in G, denoted by e_1, is different from its cheapest edge going out of the cluster in G', denoted by e_2. We know $E' \subset E$, therefore $w(e_2) > w(e_1)$. Let $e_1 = (u, v)$. Since G' is an MST of G there is a path p in G' from u to v such that any edge in this path is cheaper than e_1 (otherwise we can replace e_1 with an edges of p in G' and decrease the cost of the MST which is a contradiction). Existing a path between u and v with all the edges cheaper than e_1, implies that there is an edge going out of cluster c_1 in E' that is cheaper than e_1 which is a contradiction since $E' \subset E$ and e_1 is the cheapest edge going out of the cluster c_1 in G. □

A.3 Section 5.2

Algorithm 2 shows the MapReduce implementation using DHTs. To implement the algorithm purely in MapReduce without a DHT, we only need to replace the Map phase of Algorithm 4 (see the appendix) with the connected-components algorithm developed in prior work [31].

Proof of Theorem 11. Affinity clustering (Algorithm 2) makes $O(\log n)$ calls to "Find Best Neighbors" (Algorithm 3) and "Contract Graph" (Algorithm 4), because each round reduces the number of vertices by a factor of at least two. "Find Best Neighbors" has a simple logic and takes only one round of MapReduce. "Contract Graph," on the other hand, is a special case of the connected components problem, that may be implemented in $O(\log n)$ rounds of MapReduce [31] without a DHT, or in one round using a DHT as shown in Algorithm 4. As a result, affinity clustering takes $O(\log n)$ and $O(\log^2 n)$ rounds of MapReduce with and without a DHT, respectively. □

Algorithm 4 Contract Graph

Input: A graph $G = (V, E)$ and mapping $\lambda : V \mapsto V$
Output: A contracted graph $G' = (V', E')$
Run the following MapReduce with random access via a DHT to λ.
Map $\langle u; N(u) \rangle$

1: $c \leftarrow v \leftarrow u$
2: $S \leftarrow \emptyset$
3: **while** $v \notin S$ **do**
4: $\quad S \leftarrow S \cup \{v\}$
5: $\quad c \leftarrow \min(c, v)$
6: $\quad v \leftarrow \lambda(v)$
7: Emit $\langle c; N(u) \rangle$

Reduce $\langle u; F = \{A_1, A_2, \ldots, A_l\} \rangle$

1: $N \leftarrow \bigcup_{A \in F} A$
2: Emit $\langle u; N \rangle$