Subquadratic Approximation Algorithms For Clustering Problems in High Dimensional Spaces*

Allan Borodin† Rafail Ostrovsky‡ Yuval Rabani§

December 12, 2002

Abstract

One of the central problems in information retrieval, data mining, computational biology, statistical analysis, computer vision, geographic analysis, pattern recognition, distributed protocols is the question of classification of data according to some clustering rule. Often the data is noisy and even approximate classification is of extreme importance. The difficulty of such classification stems from the fact that usually the data has many incomparable attributes, and often results in the question of clustering problems in high dimensional spaces. Since they require measuring distance between every pair of data points, standard algorithms for computing the exact clustering solutions use quadratic or "nearly quadratic" running time; i.e., $O(dn^{2-\alpha(d)})$ time where n is the number of data points, d is the dimension of the space and

*An extended abstract of this paper appeared in STOC 1999

†Computer Science Department, University of Toronto. Part of this work was done while visiting Telcordia Technologies (previously Bellcore). Email: borodin@cs.toronto.edu

‡Telcordia Technologies (previously Bellcore), MCC-1C357B, 445 South Street, Morristown, NJ 07960-6438, USA. Email: rafail@bellcore.com

§Computer Science Department, Technion — IIT, Haifa 32000, Israel. Part of this work was done while visiting Telcordia Technologies (previously Bellcore). Work at the Technion supported by BSF grant 96-00402, by MoS contract number 9480198, and by a grant from the Fund for the Promotion of Research at the Technion. Email: rabani@cs.technion.ac.il

$\alpha(d)$ approaches 0 as d grows. In this paper, we show (for three fairly natural clustering rules) that computing an approximate solution can be done much more efficiently. More specifically, for agglomerative clustering (used, for example, in the Alta Vista$^{\text{TM}}$ search engine), for the clustering defined by sparse partitions, and for a clustering based on minimum spanning trees we derive randomized $(1+\epsilon)$ approximation algorithms with running times $\tilde{O}(d^2 n^{2-\gamma})$ where $\gamma > 0$ depends only on the approximation parameter ϵ and is independent of the dimension d.

1 Introduction

Clustering of data is an essential ingredient in many information retrieval systems (e.g., for building and maintaining taxonomies), and plays a central role in statistics, pattern recognition, biology, web search engines, distributed networks, and other fields. Recently, the concept of clustering has taken on some added significance as researchers have begun to view "data mining" as a question of finding "hidden clusters" in large collections of data. (For a survey of clustering methods see [18, 7] and references therein.) Informally, clustering algorithms attempt to form groups of similar objects into clusters based on the attributes of these objects. The question as to how best to define "the clustering problem" seems to be particularly difficult in large unstructured databases whose members are viewed as points in some high dimensional vector space.

The most successful formulations of clustering seem to be graph-theoretic formulations, providing results which have the best agreement with human performance [13]. In this formulation, the main ingredient of graph-theoretic clustering can be stated as follows: given n data points in some metric space, do the following: (1) compute some spanning graph (such as the complete graph, or a minimum spanning tree) of the original data set; (2) Delete (in parallel) some edges of this graph (according to some criterion, such as distance); (3) output clustering (such as connected components or some partitioning of the nodes which depends on the topology) of the resulting graph.

For a more concrete example, consider the framework of *hierarchical clustering* where data points are joined into sets of objects, called clusters, with the property that any two sets are either disjoint or nested. In the *agglomerative* approach to hierarchical clustering, clusters are joined to form larger clusters based on the distance between the clusters. That is, all clusters with inter-cluster distance below a certain threshold are joined to form bigger clusters and so on. The distance between clusters can be defined in a number of ways. A simple and common choice is to compute the distance between the centroids of the clusters. Examples for applications that use such clustering rules include methods for determining consensus in biological sequencing data, and the mutual fragments heuristic for computing traveling salesman tours, see Eppstein [7]. (A more general choice of distance function is the so-called *single-linkage* distance, where the distance between clusters is the minimum distance between a point in one cluster and a point in the other cluster. Of course, measuring the distance between centroids, as well as other choices for a distance function, can be formulated as a single-linkage distance between cluster representatives; i.e., sets of points that replace the clusters.)

In this formulation, the core task for computing an agglomerative clustering is the following:

Start with a complete weighted graph on the data set (points, or cluster representatives). Delete edges whose weight exceeds some given absolute bound, then output the connected components of the resulting graph. In addition to the afore-mentioned application to hierarchical clustering, this graph-theoretic clustering is used by the Alta Vista^{TM} search engine in order to prune identical documents from its world-wide web database [4]. We shall refer to it, somewhat imprecisely, as agglomerative clustering.

A second type of clustering with many applications in distributed computing (e.g., for routing tables, load balancing, file allocation) is defined by the *sparse partitions* of Awerbuch and Peleg [1]. Roughly speaking, in our setting it says that given n points and a distance r, as above define a graph where two points are adjacent if they are at (weighted) distance r or less. Nodes must be partitioned into a collection of possibly overlapping clusters. The constraints on the clusters are that (i) the (unweighted graph theoretic) diameter of each each cluster is small (a typical value is $O(\log n)$); (ii) the clusters belong to a small number (again a typical number is $O(\log n)$) of color classes, so that the clusters within each class are disjoint and, (iii) for every point x, x and all its neighbors are contained entirely in at least one cluster. Notice that this type of clustering is not hierarchical, as clusters (from different classes) may overlap.

A third type of clustering requires computing a minimum spanning tree (MST), so we call it MST-clustering. The clusters are the connected components of the forest that results from eliminating edges in the MST whose length exceeds a given parameter. Exact MST-clustering is equivalent to exact agglomerative clustering. It is not hard to see, however, that the approximate versions may differ substantially (because the approximation of MST is with respect to the total length of edges, and not with respect to the length of each edge separately). We remark that MST can be used as a subroutine for a variety of clustering methods.

We study all these clustering problems for points in d dimensional Euclidean space. (We remark that our results easily extend to other spaces and norms such as the Hamming cube and the L_1 norm on the d-dimensional real space.) This is a natural setting for applications to information retrieval and data mining. The main issue we address is the following: A naive approach for any one of these problems (in high dimension) is to compute the distances among all pairs of points, and then execute any of the above clustering rules on the resulting graph. Of course, this naive approach has quadratic (in n) or worse time complexity. Can this quadratic behavior be avoided? This is the context in which similar problems, such as minimum spanning tree or closest pair, have been studied in computational geometry (see [9]), but often the solutions are better than quadratic for low dimensions only. For example, Yao [19] shows that MST can be computed in subquadratic time but

the exponent is rapidly converging to 2 as the dimension grows. However, recent work on another geometric problem — nearest neighbor search (NNS) — shows that performance degradation as the dimension increases can be avoided if an approximation to the distances is allowed [14, 10, 15].

We therefore focus on approximate versions of the agglomerative and MST clustering problems. In most applications, the choice of both distances and the clustering rule is done on a heuristic basis, so approximate clustering is quite sufficient. The problem of sparse partition clustering already contains a certain degree of approximation implicit in the use of the "big Oh" notation and our methods can be used to derive improved bounds for the "exact problem" as defined with the understanding that the "big Oh" hides a factor of $1+\epsilon$. (For uniformity, we refer to our sparse partition algorithm as an approximation algorithm.) For all of the above problems and for any (approximation factor) ϵ, we derive clustering algorithms with time complexity $\tilde{O}(d^2 n^{2-\gamma})$ for $\gamma > 0$ which depends only on ϵ. (The $\tilde{O}$ notation also hides factors of $\log n$.) Our subquadratic [1] algorithms use some of the recent results on approximate NNS. In particular, we use modified versions of the algorithms and data structures of Kushilevitz et al. [15]. One of the difficulties in obtaining our results is that these NNS algorithms are randomized, and their probabilistic guarantees are not strong enough to allow a simple high-probability successful termination of the clustering algorithms (without requiring quadratic or worse running time). Subsequent to our work, Har-Peled, Indyk and Motwani [11] have recently reported a $1+\epsilon$ approximation to MST with running time bounds very similar to ours (i.e., the same $\tilde{O}(d^2 n^{2-\gamma})$ for $\gamma > 0$ but with slightly better functional dependence of γ on ϵ.)

Once we are willing to settle for approximate solutions we can appeal to the dimension reduction techniques of Johnson and Lindenstrauss [12] (see also [8, 16, 10]). Simply stated, any n points in Euclidean space of any dimension d can be probabilistically mapped to an $O(\log n/\epsilon^2)$ dimensional space so that no distance is increased and (with high probability) no distance shrinks by more than a $1+\epsilon$ factor and this mapping has time complexity $O(dn \log n/\epsilon^2)$. It follows that for all the approximate clustering algorithms we consider, we can assume that $d = O(\log n)$. However, for the sake of completeness (and because the dependence on d is interesting even for "small" d), we state all results in terms of d and n as well as the approximation factor ϵ.

The naive approach to clustering in Euclidean space is to first compute all $\binom{n}{2}$ distances, and then apply a clustering algorithm to the resulting weighted complete graph. For agglomerative clustering, (with standard implementations of a connected components algorithm) this naive approach

[1] Some papers will refer to any bound which is $o(n^2)$ as being "sublinear" since the naive presentation of the input would require n^2 distances.

would require $O(n^2)$ time. Broder, Glassman, Manasse, and Zweig [4] give a heuristic used in Alta Vista$^{\text{TM}}$ that seems to give good running time in practice, but requires quadratic time in the worst case.

For sparse partitions Awerbuch and Peleg [1] give a polynomial time algorithm (in the number of edges). In the naive approach, we have a weighted complete graph, where all $\binom{n}{2}$ edges are present. The following bounds are stated for this setting. Linial and Saks [17] yield a randomized $\tilde{O}(n^2)$ time algorithm (where the $\tilde{O}$ notation hides polylog(n) factors). The best result to date for the above setting (on a complete graph) is the deterministic $\tilde{O}(n^2)$ algorithm of Awerbuch, Berger, Cowen and Peleg [2].

The bottleneck for MST-clustering is computing the MST. We show how to apply the agglomerative clustering algorithm to derive a subquadratic approximate Euclidean minimum spanning tree (MST) algorithm. Sollin (see in [3]) shows how to reduce the MST problem to a problem similar to exact NNS. Yao [19] gives another reduction, and uses it to give a subquadratic algorithm for (exact) MST in low dimension. (Yao mentions obtaining a fast approximate MST algorithm as an open problem.) Yao's reduction seems specific to exact NNS, whereas Sollin's may be adapted to approximate NNS, though it is not clear how to overcome the probabilistic guarantees problem without resorting to our methods. The reduction of Yao yields an MST algorithm with time complexity $O(n^{2-2^{-d+1}}(\log n)^{1-2^{-d+1}})$. (This is somewhat better than what would result from using Yao's algorithm with Sollin's reduction.) Chazelle [6] improves this bound further, still the bound approaches quadratic behavior as the dimension grows. Notice that dimension reduction techniques cannot guarantee low distortion unless the dimension reduces to $\Omega(\log n)$, for which Chazelle's algorithm is still quadratic.

We also briefly indicate some other applications of our methods and algorithms to traditional computational geometry problems, for example a subquadratic algorithm for the Euclidean closest and furthest pair problem. The closest pair problem has many applications; for a full history see Cohen and Lewis [5] and Eppstein[7] and references therein. Kleinberg [14] uses his approximate NNS results to get an approximate closest pair algorithm with $O((n^2 + dn \log n)/\epsilon^2)$ running time (thus eliminating the dependency on the dimension for $d \leq n/\log n$). We show how to solve approximate closest pair (and approximate furthest pair) problems in subquadratic time for all d.

2 Preliminaries

We denote by E^d the d dimensional Euclidean space; i.e., $\mathbb{R}^d$ with the metric induced by the L_2 norm.

An ϵ-approximate Δ-proximity table for a finite set $P \subseteq E^d$ is a data structure supporting the following operations

- CONSTRUCT(P), which creates a new instance of the data structure for $P \subseteq E^d$; this data structure consists of $|P|^{O(1)}$ entries, where $O(1)$ may depend on ϵ. Each entry consists of a set of points.
- NEIGHBORS(x), for any $x \in E^d$, which returns a subset $P' \subseteq P$ (obtained from one of the above entries) containing all the points in P within distance Δ of x, and perhaps some additional points of P within distance $(1+\epsilon)\Delta$ of x. (Notice that there may be several possible correct answers to NEIGHBORS.)

An efficient construction of an ϵ-approximate Δ-proximity table lies at the heart of our results. We can get such a construction by adapting the approximate nearest neighbor search algorithm of [15]. These results provide the following guarantee:

Lemma 1. For every $\beta > 0$, there exists $c = c(\beta) > 0$ scuh that for every $\epsilon > 0$ there is a randomized implementation of an ϵ-approximate Δ-proximity table with the following properties:

1. CONSTRUCT takes $T(|P|) = |P|^{c/\epsilon^2}$ arithmetic operations;
2. NEIGHBORS takes $q(|P|) = c\epsilon^{-2}d^2 \log |P|$ operations;
3. for any $x \in E^d$, the probability that an entry, and hence NEIGHBORS(x), returns an incorrect list is at most $|P|^{-\beta}$.

We also need to apply frequently a union/find algorithm. For our purposes it is sufficient to assume simply (say using balanced trees) that any union or find operation in a universe of n elements can be performed within $g(n) = O(\log n)$ steps.

We also need to call approximate nearest (and furthest) neighbor algorithms[2] of [15]. These algorithms work for all queries and all distances. In particular, an ϵ-ANN/ϵ-AFN table for a finite set $P \subseteq E^d$ is a data structure supporting the following operations

- CONSTRUCT-ANN/CONSTRUCT-AFN(P),
  which creates a new instance of the data structure for $P \subseteq E^d$; this data structure consists of $|P|^{O(1)}$ "entries", where $O(1)$ may depend on ϵ, and every entry contains either a single element of P or a symbol indicating that the entry is empty.
- CLOSEST(x), for any $x \in E^d$, which returns a point in P (contained in one of the entries), whose distance from x is at most $1+\epsilon$ times the minimum distance of a point in P from x.
- FURTHEST(x), for any $x \in E^d$, which returns a point in P (contained in one of the entries), whose distance from x is at least $1-\epsilon$ times the maximum distance of a point in P from x.

An efficient construction of an ϵ-ANN/ϵ-AFN table is easily constructed from the approximate nearest neighbor search algorithm of [15]. These results provide the following guarantee:

Lemma 2. For every $\beta > 0$, there exists $c = c(\beta)$ such that for every $\epsilon > 0$ there exists a randomized implementation of an ϵ-ANN/ϵ-AFN table with the following properties:

1. CONSTRUCT-ANN/CONSTRUCT-AFN
   takes $T(|P|) = |P|^{c/\epsilon^2}$ arithmetic operations;
2. CLOSEST takes $\tilde{O}(\epsilon^{-2}d^2)$ operations;
3. FURTHEST takes $\tilde{O}(\epsilon^{-2}d^2)$ operations;
4. for any $x \in E^d$, the probability that CLOSEST(x) or FURTHEST(x) returns an incorrect answer is at most $|P|^{-\beta}$.

3 Agglomerative Clustering

In this section we discuss the following clustering problem: Given a set $P \subseteq E^d$ of n points and $\Delta > 0$, partition P into the connected components of the following graph $G_{P,\Delta}$. The graph $G_{P,\Delta}$

[2]The algorithms of [15] are easily adapted to finding furthest neighbors.

has node set P and an edge connecting every pair of nodes at Euclidean distance Δ or less. In the approximate version of the problem, we are given an additional parameter ϵ, and the graph $G_{P,\Delta}$ is replaced by a graph $G_{P,\Delta,\epsilon}$. The graph $G_{P,\Delta,\epsilon}$ has the same node set as $G_{P,\Delta}$. Its edge set contains all the edges of $G_{P,\Delta}$. In addition, it may contain edges connecting pairs of nodes at distance greater than Δ, but no greater than $(1+\epsilon)\Delta$. (Notice that the choice of $G_{P,\Delta,\epsilon}$ is not necessarily unique — any such graph gives a correct solution to the approximate problem.) We remark that in addition to separating the graph into connected components, our algorithm can be easily modified to output a *witness* spanning tree for each component.

The algorithm. We maintain a union/find structure with element set P and a multigraph $G(P, E)$, represented by the adjacency list for each node. In addition, we maintain several proximity tables for subsets of P. We set $\beta = 1$ in Lemma 1 and let $c > 0$ be the constant guaranteed by Lemma 1 Set $\delta = \epsilon^2/2c$ and $k = 2/\delta = 4c/\epsilon^2$.

The algorithm is shown in Figure 1.

At the end of the algorithm, the desired partition of P into clusters is the partition of the (sparse) graph G constructed by the algorithm into its connected components.

Notation. Let $x \in \mathbb{R}^d$ and let $\ell > 0$. Denote by $B(x, \ell)$ the closed ball around x with radius ℓ; i.e., the set $\{y \in \mathbb{R}^d \mid \|x - y\|_2 \leq \ell\}$.

Correctness. The correctness of the algorithm is an immediate corollary of the following claim:

Claim 3.

1. With high probability, (i.e., $\geq 1 - n^{-\delta}$) every node $u \in P$ is in the same connected component as all the nodes in $B(u, \Delta)$.
2. If $u, v \in P$ are in the same connected component of G, then there is a sequence of nodes $u = v_0, v_1, \ldots, v_t = v$, such that for all $i = 1, 2, \ldots, t$, $\text{dist}(v_{i-1}, v_i) \leq (1+\epsilon)\Delta$.

Proof. To see (1), consider an n^δ-subset P_i of P used by the algorithm and let u be some point in P. Let $\mathcal{E}_i$ be the event that u retrieves a proximity table entry containing all of $B(u, \Delta) \cap P_i$ and none of $\overline{B(u, (1+\epsilon)\Delta)} \cap P_i$ (call such an entry *good*). $\Pr[\overline{\mathcal{E}_i}] \leq p^k$, where p is the failure probability of a query. Thus $p \leq n^{-\delta}$, so $p^k \leq n^{-2}$. We have n nodes and $n^{1-\delta}$ sets P_i, so with high probability

Partition P arbitrarily into $n^{1-\delta}$ sets
P_1, P_2, ..., of size n^δ each;
Initialize $E \leftarrow \emptyset$;
Repeat k times:
    Initialize the union/find structure
        (each element is a set);
    For each $j = 1, 2, \ldots, n^{1-\delta}$:
        $T \leftarrow$ CONSTRUCT(P_j);
        Mark each entry in T by 0;
        For each node $x \in P$:
            $L \leftarrow$ NEIGHBORS(x); (say $L = \{y_1, y_2, \ldots\}$)
            If $L = \emptyset$, skip x;
            If L is marked 1 and $\text{dist}(x, y_1) \leq (1+\epsilon)\Delta$,
                UNION(x,y_1), $E \leftarrow E \cup \{(x, y_1)\}$;
            If L is marked by 0 and $\forall i$, $\text{dist}(x, y_i) \leq (1+\epsilon)\Delta$,
                $\forall i$, UNION(x,y_i), $E \leftarrow E \cup \{(x, y_i)\}$;
                Mark L by 1.

Figure 1: The agglomerative clustering algorithm.

every event $\mathcal{E}_i$ happens. Now, if $\mathcal{E}_i$ happens and u retrieves a good entry marked by 0, then we connect u to all the nodes in that entry. Otherwise (the entry is marked 1), there is another node v connected to all the nodes in the entry, and u connects to one of the nodes of the entry.

For (2), use induction on the number of union operations performed. Consider a UNION(x,y) operation which joins two components $C_1 \ni x$, $C_2 \ni y$. The only case which does not follow from the induction hypothesis is $u \in C_1$ and $v \in C_2$. But by the induction hypothesis, there is a sequence $u = v_0, v_1, \ldots, v_s = x$, and another sequence $y = v_{s+1}, v_{s+2}, \ldots, v_t = v$, such that for every $i \neq s+1$, $1 \leq i \leq t$, $\mathrm{dist}(v_{i-1}, v_i) \leq (1+\epsilon)\Delta$. By the specification of the algorithm, $\mathrm{dist}(v_s, v_{s+1}) = \mathrm{dist}(x, y) \leq (1+\epsilon)\Delta$. ■

Analysis. We analyze the complexity of one iteration of the outer "Repeat loop" and then multiply by k to obtain the total running time. For simplicity we suppress the influence of c and ϵ in the big Oh notation. The time complexity of the "Repeat loop" is upper bounded by the sum

$$D + N + B + G + U,$$

where D is the cost of building the $n^{1-\delta}$ proximity tables, N is the minimum search cost for the nodes (i.e., assuming they retrieve entries marked 1 only), B is the cost for retrieving bad entries, G is the cost induced by retrievals of good 0-marked entries, and U is the cost of performing union/find. We analyze each term separately.

Claim 4. Let $\beta = 1$ and let $c = c(\beta)$ be the constant from Lemma 1. Let $g(n) = O(\log n)$ be the worst case cost of a union/find algorithm. Then,

$$\begin{aligned}
D &= n^{1-\delta} T(n^\delta) = o(n^{3/2}) \\
N &= n^{2-\delta}\left(q(n^\delta) + d\right) = O(d^2 n^{2-\epsilon^2/2c} \log n) \\
\mathrm{E}[B] &= O(n^{2-\delta} d) = O(d n^{2-\epsilon^2/2c}) \\
G &= O(n T(n^\delta) d) = O(d n^{3/2}) \\
U &= g(n)\left(n^{2-\delta} + n T(n^\delta)\right) = O(n^{2-\epsilon^2/2c} \log n)
\end{aligned}$$

Proof.

- For D: In each iteration we have to construct $n^{1-\delta}$ proximity tables, each for n^{δ} points. The construction of each table takes $T(n^{\delta})$ steps.
- For N: In each iteration we process each of the n points. Each point requires a search in $n^{1-\delta}$ proximity tables. The search in each proximity table takes $q(n^{\delta})$ time. If the retrieved entry is not empty and is marked 1, there is an additional cost of $O(d)$ to compute the distance to the first point on the list.
- For B: In each iteration we access $n^{1-\delta}$ proximity tables, searching for n points in each table. For each table, for each point, the probability that the point retrieves a bad entry is at most $\left(n^{-\delta}\right)$. Therefore the expected number of bad entries handled in each proximity table is at most $n^{1-\delta}$. For each bad entry, we may have to compute the distance to all the points in the table. This costs $O(n^{\delta}d)$.
- For G: In each iteration we handle $n^{1-\delta}$ proximity tables. Each table has at most $T(n^{\delta})$ entries. An entry is accessed as a good entry that is marked 0 at most once. If this happens, we compute the distances to all the points listed, at most n^{δ}. Each distance costs $O(d)$.
- For U: The bound follows from an estimate on the total number of union operations performed. In each iteration we handle $n^{1-\delta}$ proximity table. Each proximity table is accessed by n points. If a point retrieves an entry marked 1, it causes at most one union operation, so there are at most n such operations per table. Good entries marked 0 cause at most n^{δ} union operations. For each entry we do this at most once. Thus, the number of these union operations per table is at most $n^{\delta}T(n^{\delta})$.

■

Theorem 5. The total running time of the agglomerative clustering algorithm is $O(d^2n^{2-\epsilon^2/2c}\log n)$

4 Sparse Partitions

In this section we discuss computing a *sparse partition* clustering of a finite set of points $P \subseteq E^d$, $P = \{x_1, \ldots, x_n\}$. Our definition of sparse partitions is the Euclidean analogue of the definitions given in [1], [2] and [17] for undirected graphs where in those papers distance refers to path length.

A sparse partition $\mathcal{C}$ of P (with parameter $r > 0$), is a collection of subsets (called clusters) $S_1, S_2, \ldots, S_m \subseteq P$ satisfying the following conditions:

1. The (Euclidean) diameter of each cluster S_i is at most $O(r \log n)$.
2. For every $x \in P$, $P \cap B(x,r)$ is contained completely in at least one of the clusters where $B(x,r)$ is the Euclidean ball of radius r centered at x.
3. The clusters can be grouped into $O(\log n)$ classes, so that the clusters in each class are disjoint. (It follows that for each $x \in P$, there are at most $O(\log n)$ clusters containing x.) Furthermore, the distance between any two clusters in the same class is greater than r; that is, if clusters S_i and S_j are in the same class with $x \in S_i$ and $y \in S_j$, then the Euclidean distance between x and y is greater than r.

(The original definition of sparse partitions is more general in that it allows tradeoffs between the values in conditions 1 and 3. The definition here uses the most common values in applications. Our results extend to these general tradeoffs.)

The algorithm. As in [17], we run $O(\log n)$ phases. In each phase we grow a new class (of clusters) which contains a constant fraction of all the nodes. In a phase, the status of each node is either FREE or USED (initially in a phase, all nodes are FREE). We grow a cluster by picking a FREE node and adding its FREE neighbors in a breadth first search manner, stopping at a graph distance chosen at random using the truncated geometric distribution of [17] (i.e., choose the distance to be i with probability 2^{-i}, for i ranging from 1 to $2\log n - 2$, and with probability $2^{-2\log n+1}$, for $i = 2\log n - 1$ or $i = 2\log n$). The main difference with [17] is in our implementation of the breadth first search, which exploits the underlying structure of the graph, giving the speedup in the running time. In particular, we need to be able (with high probability) to determine all so far FREE r-neighborhood points in an approximate ball $\tilde{B}(y,r)$ centered at y of radius r which contains $B(y,r)$ and may also include some (so far FREE) points in the slightly larger ball $B(y,(1+\epsilon)r)$. A more formal description of the algorithm appears in Figure 2.

Computing neighbors. We compute the set $\tilde{B}(y,r)$ using the proximity tables that are initialized at the beginning of the phase. We first explain the initialization: Let δ and k be as in the previous section. We partition P into $n^{1-\delta}$ disjoint sets of size n^{δ} each. For each set we construct k independent ϵ-approximate r-proximity tables. We also mark all the entries in all the tables by 0.

Repeat $O(\log n)$ times:
    Initialize proximity tables (see discussion in **Computing neighbors**) ;
    Mark all $x \in P$ as FREE;
    Choose at random $i \in \{1, 2, \ldots, 2\log n\}$
        with geometric truncated distribution;
    While there is a FREE node $x \in P$:
        Initialize a new cluster $S \leftarrow \emptyset$;
        Mark node x as USED
        Initialize $F \leftarrow \{x\}$, $F' \leftarrow \emptyset$;
        For i steps do:
            For each $y \in F$:
                For each FREE $z \in \tilde{B}(y, r)$:
                    $F' \leftarrow F' \cup \{z\}$;
                    mark z as used;
            $S \leftarrow S \cup F$, $F \leftarrow F'$, $F' \leftarrow \emptyset$

Figure 2: The sparse partitions algorithm.

This completes the initialization. We construct the set $\tilde{B}(y,r)$ as follows: We fetch NEIGHBORS(y) in each of the $kn^{1-\delta}$ tables. Let $L_1, L_2, \ldots, L_{kn^{1-\delta}}$ be the resulting lists. We ignore all lists marked by 1. For every list L marked 0 we compute dist(y,z), for all $z \in L$. If any point in L is further from y than $(1+\epsilon)r$, we leave L marked by 0. Otherwise we add the elements of L to $\tilde{B}(y,r)$ and mark L by 1.

Correctness. Here we prove that the above algorithm indeed produces the desired sparse partition clustering with high probability. The main issue is to show that $\tilde{B}(y,r)$ is computed correctly. The rest of the argument follows in general the line of argument in [17]. The added difficulty is that because we have approximate distances only, proximity between points is no longer a symmetric property.

Claim 6. With probability[3] at least $\frac{19}{20}$ the following holds for every $y \in P$: If the algorithm computes $\tilde{B}(y,r)$ then this set contains all the points in $B(y,r)$ that are still FREE, and none of the points in $\overline{B(y,(1+\epsilon)r)}$.

Proof. The latter claim is obvious because the algorithm checks the distances to the points it adds to $\tilde{B}(y,r)$. To see the former claim, we argue as in the proof of Claim 4 that the probability that the good entries retrieved by y do not contain all the points in $B(y,r)$ is at most n^{-2}, and therefore the probability that this happens for *any* y is at most n^{-1}. Finally, notice that if an entry is marked 1, then the status of all the points listed in the entry (in the current phase) is USED. ■

Claim 7. Consider any particular phase. Let $y \in P$ be the first point in $B(x,r)$ that is placed in F in that phase. If y is placed in F just before the execution of iteration $j \leq 2\log n - 2$ of the "For" loop of that phase, then $B(x,r)$ is contained in a cluster generated in the phase with probability at least $\frac{1}{5}$.

Proof. Let F_j be the set of points in $B(x,r)$ that are in F at the start of iteration j; that is, if $j > 1$ these are the points that were placed in F' during iteration $j-1$. Assume for the moment that $x \notin F_j$. Clearly, $y \in F_j \subseteq F$. Moreover, at the start of the jth iteration, all the points in $B(x,r) \setminus F_j$ (and in particular x) are FREE. As the jth iteration is executed, we know that $i \geq j$. Given that $i \geq j$, and that $j \leq 2\log n - 2$, the conditional probability that $i \geq j+2$ is $\frac{1}{4}$. So assume that

[3]The choice of constant is arbitrary

the event $i \geq j+2$ holds, and further assume that the event in Claim 6 holds. This happens with probability at least $\frac{1}{4} - \frac{1}{20} = \frac{1}{5}$. As $y \in B(x,r)$, $\text{dist}(y,x) \leq r$. Therefore, $x \in B(y,r) \subseteq \tilde{B}(y,r)$, and x is placed in F' during the jth iteration (by Claim 6). Let F_{j+1} be the set of points in $B(x,r)$ that are placed in F' during the jth iteration. At the end of the jth iteration, all the points in F_j are added to the cluster S, and all the points in F' are placed in F. In particular, x is placed in F, and so are all the other points in F_{j+1}. We have assumed that $i \geq j+2$. Therefore iteration $j+1$ is executed. Notice that at the start of this iteration, the points in $B(x,r) \setminus (F_j \cup F_{j+1})$ are FREE. Denote these points by F_{j+2}. As $x \in F$, during iteration $j+1$ the FREE points in $\tilde{B}(x,r)$ are placed in F'. These include all the points of F_{j+2}. At the end of iteration $j+1$, the points in F, and in particular the points of F_{j+1}, are placed in the cluster S. The points in F', in particular all of F_{j+2} are placed in F. As we are assuming that iteration $j+2$ is executed as well, at the end of that iteration the points of F_{j+2} are placed in the cluster S. Because $F_j \cup F_{j+1} \cup F_{j+2}$ are all the points in $B(x,r)$, the claim follows. If $x \in F_j$ then a similar argument shows that S will contain all the points in $B(x,r)$ with even larger probability. ■

Claim 8. Consider any particular phase. $\forall x$, the probability that the first $y \in B(x,r)$ to be clustered is reached in the last two iterations of the main loop for that phase is $\frac{4}{n^2}$.

Proof. i is chosen once each phase, and the probability that $i > 2\log n - 2$ is $\frac{4}{n^2}$. ■

The above claims imply

Theorem 9. With high probability the algorithm produces a sparse partitioning clustering with parameter r.

Proof. By Claims 7 and 8, for every point $x \in P$, the following holds. In any phase, the probability that $B(x,r) \cap P$ is completely contained in a cluster generated in that phase is a bit below $\frac{1}{5}$, but clearly more than, say $\frac{1}{6}$. Thus, after $O(\log n)$ phases, with high probability this property holds for all points x. The bound on the diameters of the clusters follows from the choice of i. The bound on the number of clusters containing a point follows from the fact that there are $O(\log n)$ iterations, and each iteration generates a collection of disjoint clusters. ■

Analysis. Let δ and k be as described in the implementation. The time complexity for a phase in the above algorithm is (similar to agglomerative clustering) upper bounded by the sum

$$D + N + B + G + U,$$

where D is the cost of building the $kn^{1-\delta}\log n$ r-proximity tables, N is the minimum search cost for $\tilde{\mathcal{B}}(y,r)$ (i.e. cost of retrieving all the pointers only), B is the cost for retrieving entries that are marked 0, but which contain points further then $(1+\epsilon)r$, G is the cost induced by retrievals of good 0-marked entries (i.e. such that all are at distance within $(1+\epsilon)r$, and U is the cost of performing set unions. Again, we analyze each cost separately.

Claim 10. Let $\beta = 1$ and let $c = c(\beta)$ be the constant from Lemma 1. Then,

$$\begin{aligned} D &= o(kn^{3/2}\log n) \\ N &= O(kd^2n^{2-\delta}\log^2 n) \\ \mathrm{E}[B] &= O(kn^{2-\delta}d\log n) \\ G &= O(kn^{3/2}\log n) \\ U &= O(n^{2-\delta}\log^2 n). \end{aligned}$$

Proof. Analogous to Lemma 4.

Theorem 11. The total running time of the sparse partitions algorithm is $O(d^2n^{2-\epsilon^2/2c}\log^2 n)$

■

5 Computing an approximate MST and other related problems

In this section we discuss several other geometric problems that can be approximated in subquadratic time using our methods. First, we consider the problems of computing a $1+\epsilon$ appoximation to the closest and to the furthest pair of points, an application also considered by Kleinberg [14]. We have the following result:

Lemma 12. For each constant ϵ there exists $\delta > 0$ (the same δ as in agglomerative clustering) so that in time $\tilde{O}(d^2n^{2-\delta})$ we can compute w.h.p. a pair of points which are within a factor $1+\epsilon$ of the distance of the closest (or furthest) pair of points.

Proof Partition the set P into $n^{1-\delta}$ subsets of size n^{δ} each, as before. For each subset we build an ϵ-ANN (or an ϵ-AFN) data structure. We then search each of the structures for each point in P. By repeating the search several times and taking the best answer, we can reduce the error probability sufficiently. ■

We now consider the problem of computing an approximate minimum spanning tree (MST).

Lemma 13. For each constant ϵ there exists δ so that we can compute w.h.p. a $(1+\epsilon)$-approximation to the minimum spanning tree in time $\tilde{O}(d^2 n^{2-\delta})$.

Proof Iterate over several computations of agglomerative clustering, each time reduing the parameter Δ by a factor of about $1+\epsilon$. Notice that our agglomerative clustering algorithm outputs a sparse graph G as a witness to the connectivity of each cluster. We use such a witness here (i.e., a "black-box" agglomerative clustering algorithm that outputs the partition into clusters alone is insufficient). Whenever a cluster in one iteration is split in the next, we add to the growing forest edges connecting the current fragments of the cluster. These edges are taken from the graph G computed in the previous iteration. (Notice that the graph in the current iteration may differ from the the graph in the previous iteration.) We add edges of G, that do not connect two points that are currently also in the same cluster, and do not close a cycle. (Because G is sparse enough, we can check all its edges.)

We can begin with any Δ which on the one hand guarantees a single cluster, and on the oher hand is not too big (close enough to a lower bound on the MST). For example, we can take $1+\epsilon$ times the result of an approximate furthest pair search. We can stop when Δ reaches a value small enough to allow us to take the remaining connections as we please. Because an MST contains $n-1$ edges, a value of about ϵ/n times the initial value of Δ is sufficient (recall that the initial value is close to a lower bound on the MST). Thus, the number of agglomerative clustering computations, for fixed ϵ, is $O(\log n)$. ■

As mentioned previously, we can now use such an approximate MST and any threshold value θ as the basis for approximate MST clustering by deleting from the MST any edges of distance larger than θ and viewing the resulting components as clusters. We note that this MST clustering might be different from an approximate agglomerative clustering using the same parameter θ.

Acknowledgment

We thank Andrei Broder for his motivating discussions regarding the Alta-Vista$^{\text{TM}}$ search engine.

References

[1] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. of 21st FOCS* pp. 503-513, 1990.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear cost sequental and distributed constructions of sparse neighborhood covers. In *Proc. of 34th FOCS* pp. 639-647, 1993.

[3] C. Berge and A. Ghouila-Houri. *Programming, Games, and Transportation Networks*. John Wiley, 1965.

[4] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the Sixth International World Wide Web Conference*, pages 391-404, 1997

[5] E. Cohen and D. Lewis. Approximating matrix multilication for pattern recognition tasks. In *Proc. of 8th SODA*, 1997.

[6] B. Chazelle. How to search in history. *Information and Control*, 64:77–99, 1985.

[7] D. Eppstein. Fast hierachical clustering and other applications of dynamic closest pair. In *Proc. of 9th SODA*, 1998.

[8] P. Frankl and H. Maehara. The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *J. of Combinatorial Theory B*, 44:355–362, 1988.

[9] J. Goodman and J. O'Rourke, eds. *Handbook of Decrete and Computational Geometry*. CRS press, 1997.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of 30th STOC*, 1998.

[11] P. Indyk, R. Motwani, S. Har-Peled. Private communication (including a complete version of [10]), 2002.

[12] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[13] J. Jaromczyk, and G. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 90:1502-1517, 1992.

[14] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. of 29th STOC*, pp. 599–608, 1997.

[15] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM J. Comput. 30, 2, 457–474. (Preliminary version in *Proc. of 30th STOC*, 614–623, 1998.)

[16] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[17] N. Linial and M. Saks. Decomposing graphs into regions of small diamater. *Proc. of 2nd SODA*, pp. 320–336, 1991.

[18] J. O'Rourke and G. Toussaint. Pattern recognition. In [9], pp. 797–813.

[19] A.C. Yao. On constructing minimum spannning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, November 1982.