On approximate geometric k-clustering*

Jiří Matoušek**

Department of Applied Mathematics
Charles University
Malostranské nám. 25, 118 00 Praha 1
Czech Republic

Abstract

For a partition of an n-point set $X \subset \mathbf{R}^d$ into k subsets (clusters) $S_1, S_2, \ldots, S_k$, we consider the cost function $\sum_{i=1}^k \sum_{x \in S_i} \|x - c(S_i)\|^2$, where $c(S_i)$ denotes the center of gravity of S_i. For $k = 2$ and for any fixed d and $\varepsilon > 0$, we present a deterministic algorithm that finds a 2-clustering with cost no worse than $(1+\varepsilon)$-times the minimum cost in time $O(n \log n)$; the constant of proportionality depends polynomially on ε. For an arbitrary fixed k, we get an $O(n \log^k n)$ algorithm for a fixed ε, again with a polynomial dependence on ε.

1 Introduction

We consider a geometric k-clustering problem: given an n-point set $X \subset \mathbf{R}^d$ and a natural number $k \geq 2$, find a partition (clustering) $\Pi = (S_1, S_2, \ldots, S_k)$ of X into k disjoint nonempty subsets that minimizes a suitable cost function among all k-clusterings of X. The cost function should show how tightly each S_i is "packed together" and how well the different S_i are separated from each other. We will consider the following cost function, based on intra-cluster variances:

$$\mathrm{cost}(\Pi) = \sum_{i=1}^{k} \mathrm{cost}(S_i), \qquad \text{where} \qquad \mathrm{cost}(S) = \sum_{x \in S} \|x - c(S)\|^2.$$

Here $\|.\|$ denotes the Euclidean norm and $c(S) = \frac{1}{|S|} \sum_{x \in S} x$ is the centroid (center of gravity) of the set S.

We say that a k-clustering Π of X is $(1+\varepsilon)$-*approximately optimal* if $\mathrm{cost}(\Pi) \leq (1+\varepsilon)\mathrm{cost}(\Pi')$ for any k-clustering Π' of X.

*Part of this research was done during a visit to the Imai Laboratory at the University of Tokyo, whose support and hospitality is gratefully acknowledged.

**Research supported by Charles University grants No. 158/99 and 159/99.

In this paper, we investigate approximate k-clustering algorithms. The space dimension d and the number of clusters k will always be regarded as (small) constants, and the constants of proportionality hidden in the $O(.)$ notation usually depend on them.

For 2-clustering (which can be used as a basis of a heuristic clustering algorithm by recursive splitting), we give a near-linear approximate algorithm for any fixed $\varepsilon > 0$:

Theorem 1.1 *Let $X \subset \mathbf{R}^d$ be an n-point set, and let $\varepsilon > 0$ be given. A $(1+\varepsilon)$-approximately optimal 2-clustering of X can be found in time*

$$O\Big(n \log n \cdot \varepsilon^{-2d} \log \tfrac{1}{\varepsilon} + n\varepsilon^{-(4d-2)} \log \tfrac{1}{\varepsilon}\Big).$$

In particular, the running time is $O(n \log n)$ for a fixed $\varepsilon > 0$.

If $f(s)$ denotes the smallest cost of a 2-clustering of X whose both clusters have size at least $s \geq 1$, then a 2-clustering with cost at most $(1+\varepsilon)f(s)$ can be computed in time

$$O\Big(n \log n + \tfrac{n}{s}(\log n)^2 \varepsilon^{-2d} \log \tfrac{1}{\varepsilon} + \tfrac{n}{s} \log n \cdot \varepsilon^{-(4d-2)} \log \tfrac{1}{\varepsilon}\Big).$$

Let us remark that in the secont part of the theorem, it is *not* guaranteed that the 2-clustering found by the algorithm has clusters of size $\geq s$. On the other hand, if we know apriori that all $(1+\varepsilon)$-approximate clusterings have clusters of size $\geq s$, then the algorithm can, of course, be used to find one.

As was noted by Kasturi Varadarajan [10], a result better than the first part of Theorem 1.1 can be obtained in a quite simple way, using an observation in this paper. His approach, leading to $O(n \log n \cdot \varepsilon^{-(d-1)})$ running time, is outlined in the Appendix. On the other hand, it is not known how to extend his method to the case of k-clustering or to the situation in the second part of Theorem 1.1. We still include the original proof first part of Theorem 1.1, since the tools developed there are used in the other results.

For an arbitrary fixed $k > 2$, we get a somewhat worse, although still near-linear, algorithm.

Theorem 1.2 *Let $X \subset \mathbf{R}^d$ be an n-point set, let $k \geq 3$ be fixed, and let $\varepsilon > 0$ be given. A $(1+\varepsilon)$-approximately optimal k-clustering of X can be found in time*

$$O\Big(n(\log n)^k \varepsilon^{-2k^2 d}\Big).$$

If we consider only clusters of size at least s, an improvement similar to Theorem 1.1 is possible as well, but we will not elaborate on this in this paper.

Geometric facts about k-clustering. Before reviewing previous work about k-clustering, let us introduce some notation and simple geometric facts.

For a point $p \in \mathbf{R}^d$ and a finite set $S \subset \mathbf{R}^d$, let us put

$$\mathrm{cost}(S,p) = \sum_{x \in S} \|x-p\|^2.$$

As is well-known, and easy to check, we have $\mathrm{cost}(S) = \mathrm{cost}(S, c(S)) = = \min_{p \in \mathbf{R}^d} \mathrm{cost}(S,p)$. Further, if $c_1, \ldots, c_k \in \mathbf{R}^d$ are given points, then $\sum_{i=1}^k \mathrm{cost}(S_i, c_i)$ is minimized, over all choices of a k-partition $(S_1, S_2, \ldots, S_k)$ of X, by letting S_i be the set of points of X for which c_i is the nearest point among $c_1, \ldots, c_k$ (ties are broken arbitrarily). Geometrically speaking, S_i are the points of X lying in the Voronoi region of c_i in the Voronoi diagram of the set $\{c_1, c_2, \ldots, c_k\}$. Let us call such a partition $(S_1, S_2, \ldots, S_k)$ the *Voronoi partition* of X according to $c_1, c_k, \ldots, c_k$, and let us denote it by $\Pi_{\mathrm{Vor}}(c_1, \ldots, c_k)$. Thus, an optimal k-clustering is a Voronoi partition, and in particular, an optimal 2-clustering is linearly separable.

Previous work. Clustering by the above cost function is frequently used in the literature and in practical applications. In practice, mostly heuristic methods have been used, such as the *k-means algorithm* (local improvements of the current clustering by moving individual points among the clusters).

Algorithms with performance guarantees for k-clustering in $\mathbf{R}^d$ have recently been considered by Hasegawa et al. [7] and by Inaba, Katoh, and Imai [8] (where also some background information and references to other papers can be found). Inaba, Katoh, and Imai [8] observe that the number of distinct Voronoi partitions of a given n-point set $X \subset \mathbf{R}^d$ induced by k points $c_1, c_2, \ldots, c_k$ is at most $O(n^{kd})$, and they can be enumerated in $O(n^{kd+1})$ time. Consequently, the optimum k-clustering under the variance-based cost defined above can be found in time polynomial in n, for any fixed d and k.

Hasegawa et al. [7] noted that if $\Pi_{opt} = (S_1, S_2, \ldots, S_k)$ is an optimal partition and if c_i is chosen as the point of S_i nearest to $c(S_i)$, then $\mathrm{cost}(\Pi_{\mathrm{Vor}}(c_1, \ldots, c_k)) \leq 2\mathrm{cost}(\Pi_{opt})$. By testing all the Voronoi partitions with $c_1, \ldots, c_k$ chosen among the n points of X, in $O(n^{k+1})$ time, one can thus find a 2-approximately optimal k-clustering.

Inaba, Katoh, and Imai [8] presented a randomized algorithm for finding a near-optimal 2-clustering among the 2-clusterings with no too small cluster. More precisely, let $\varepsilon > 0$ and $s \in [1, n]$ be parameters. Their algorithm finds, with probability at least $\frac{1}{2}$, a 2-clustering for which the cost is no worse than $(1+\varepsilon)$-times the cost of any 2-clustering with cluster size at least s. The running time is $O(nm^d)$, where m is of the order $\frac{n}{\varepsilon s} + \frac{n}{s} \log \frac{n}{s}$. They remark that their method can also be generalized for finding a k-clustering with a fixed k (in that case, they need to consider all Voronoi k-partitions of an m-point set in $\mathbf{R}^d$).

Remarks and further work. With the techniques of the present paper, it doesn't seem easy to improve the $O(n \log n)$ running time for approximate 2-clustering with ε fixed, but I do not know whether $\Omega(n \log n)$ is a lower bound or not. Improving the running time for approximate k-clustering, perhaps to $O(n \log n)$ for any fixed k and ε, is another interesting problem.

The work on this paper started by trying to find a deterministic counterpart of the 2-clustering algorithm of Inaba, Katoh, and Imai [8]. They consider a random sample T of m points from X, and for all linearly separable 2-partitions (T_1, T_2) of T, they use the centroids of T_1 and of T_2 as candidates for centroids of an approximately optimal 2-clustering of X. The usual derandomization techniques in computational geometry would suggest to replace the random sample T by a suitable deterministically computed sample. However, in this problem, it seems difficult to define suitable properties of such a good sample. If one takes the probably most natural such property, namely that for each linearly separable 2-clustering (S_1, S_2) of X there is a linearly separable 2-clustering (T_1, T_2) of T such that the centroids of T_1 and of T_2 approximate the centroids of S_1 and of S_2, respectively, then a single good small sample T need not exist at all (in the randomized algorithm, the sample T is good in this sense, with probability close to 1, for any *fixed* (S_1, S_2), but it need not be good for all 2-partitions simultaneously). Thus, instead of imitating the sampling of T, we will construct a suitable set of candidates for the centroids directly.

In this paper, we concentrate on asymptotic results. We do not make any attempt to optimize the various constants appearing in the proofs and algorithms, preferring conceptual simplicity and simplicity of exposition instead. It is possible that some of the ideas in the paper can be applied in practically efficient computations, but a significant amount of work seems to be needed for developing such practical versions of the algorithms.

The cost function considered in this paper is only one of many possibilities investigated in the literature. In particular, the following family of cost functions, parameterized by a real number $\alpha \in [0, 2]$, has geometric properties somewhat similar to those of $\mathrm{cost}(\Pi)$. Namely, the cost of a cluster S is given by

$$\mathrm{cost}_\alpha(S) = |S|^{\alpha-1} \sum_{x \in S} \|x - c(S)\|^2$$

and the total cost of a partition is again the sum of costs of its clusters. The case investigated in this paper is $\alpha = 1$; other interesting cases are $\alpha = 0$ (the cost of a cluster is its *variance*) and $\alpha = 2$ (the cost is the sum of squared distances of all pairs in the cluster). For $\alpha \in [1, 2]$, the optimum clustering can be characterized by a *weighted Voronoi diagram* (generally with both additive and multiplicative weights); see Inaba, Katoh, and Imai [8] for the cases $\alpha = 1, 2$. Methods of the present paper might be applicable to k-clustering with the cost function cost_α with $\alpha \in [1, 2]$, although further work seems to be needed to give efficient approximation algorithms. Another interesting class of cost functions arises, for example, by taking the same costs for the clusters but combining them in a different manner, say by taking the maximum over all clusters. Also, clusterings where the cost of a cluster is its diameter or the radius of its smallest containing ball have been investigated intensively in computational geometry (see e.g. [6]). The geometry of such clusterings is quite different, but some of our techniques might still be useful.

A very important stream of results on geometric approximation algorithms was initiated mainly by the paper of Arora [1], which gave a polynomial-time $(1+\varepsilon)$-approximation algorithm for the Traveling Salesman Problem in a Euclidean space of fixed dimension, for any fixed $\varepsilon > 0$. Numerous other problems were attacked successfully by a similar approach. Results somewhat related to our clustering problem were obtained by Arora et al. [2], who consider the *k-median* problem. Here, for an n-point set $X \subset \mathbf{R}^d$, a k-point set $M \subseteq \mathbf{R}^d$, the *k medians*, should be found minimizing the sum, over $x \in X$, of distance of x to its nearest neighbor in M. In other words, this is a k-clustering problem where the cost of a cluster S is $\min_{p\in\mathbf{R}^d} \sum_{x\in S} \|x-p\|$, and the total cost is the sum of the costs of the clusters. For $d = 2$, Arora el al. [2] obtain a polynomial-time randomized algorithm for any fixed $\varepsilon > 0$ (while k is regarded as a variable parameter). Very recently, Kolliopoulos and Rao [9] obtained the expected running time $O(2^{1/\varepsilon^d} n \log n \log k)$, for any fixed dimension d (also, in their formulation of the problem, the medians can only be selected among the points of X). The k-median problem looks very similar to the k-clustering with the cost function considered here, but the cost function is a sum of distances rather than of squared distances. While the geometry of the clustering problem with squared distances is simpler in some respects, the squared distances seem to make the problem rather different from those where Arora's approach has been applied so far. We also remark that the dependence on ε and on d in the Kolliopoulos and Rao result is exponential and doubly exponential, respectively. Nevertheless, it might be worth trying to use some of the ideas of Arora and others for getting a better dependence on k, say, in the k-clustering algorithm.

2 Preliminaries

Let $\mathrm{diam}(X)$ denote the diameter of a set $X \subset \mathbf{R}^d$.

Let A and M be sets in a metric space. We say that M is *η-dense* for A if each point of A is at distance at most η from some point of M. We will sometimes need η-dense sets for simple convex sets A in $\mathbf{R}^d$, such as cubes, under the Euclidean metric. An example of an η-dense set for A is the intersection of the grid $\eta d^{-1/2}\mathbf{Z}^d$ with the η-neighborhood of A. In the simple cases we will deal with, this set is easily constructed in time proportional to its size.

Well-separated pairs. For a real number $\varepsilon \geq 0$, we define a relation $\sim_\varepsilon$ on (ordered) pairs of points in $\mathbf{R}^d$: we let $(x,y) \sim_\varepsilon (x',y')$ if $\|x-x'\| \leq \varepsilon \cdot \|x-y\|$ and $\|y-y'\| \leq \varepsilon \cdot \|x-y\|$ (Fig. 1). We say that (x,y) and (x',y') are *ε-near* if $(x,y) \sim_\varepsilon (x',y')$ and $(x',y') \sim_\varepsilon (x,y)$ (note that $\sim_\varepsilon$ is not "quite" symmetric). We say that a set P of ordered pairs of points of $\mathbf{R}^d$ is *ε-separated* if no two pairs

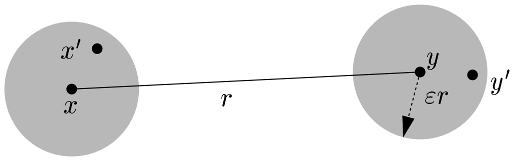


Figure 1: The relation $\sim_\varepsilon$.

in P are ε-near. An *ε-complete set of pairs* for a set X is a set P of ordered pairs such that any ordered pair of points of X is ε-near to some pair in P. (Note that we do not insist that points in the pairs in P be from X, although often it will be the case—for example in the following theorem.)

We will use a result of Callahan and Kosaraju [5], which for our purposes can be phrased as follows:

Theorem 2.1 *Let X be an n-point set in $\mathbf{R}^d$, and let $\varepsilon \in (0,1)$. Then any ε-separated set $P \subseteq X \times X$ of pairs has size at most $O(n\varepsilon^{-d})$. An ε-complete set P of pairs for X, with $|P| = O(n\varepsilon^{-d})$ and $P \subseteq X \times X$, can be computed in $O(n \log n + n\varepsilon^{-d})$ time.*

Approximate range searching. In a range searching problem, we consider an n-point set $P \subset \mathbf{R}^d$ and a class $\mathcal{R}$ of admissible ranges (such as d-dimensional axis-parallel boxes, spheres, simplices, etc.). Each point $p \in P$ is equipped by a weight $w(p) \in S$, where $(S, +)$ is a commutative semigroup (this means that the weights can be added together; computationally, one assumes that a weight can be stored in a single word of memory and that the semigroup operation can be performed in unit time). The goal is to preprocess the set P with the weights and store the results in a data structure so that, given a query range $R \in \mathcal{R}$, the total weight of points in R, i.e. $\sum_{p \in P \cap R} w(p)$, can be calculated efficiently.

We will need a result of Arya and Mount [4] on approximate range searching. Given a range $R \in \mathcal{R}$ and a number $\varepsilon > 0$, let R^+ be the set of all points of distance at most $\varepsilon \cdot \mathrm{diam}(R)$ from R, and let R^- be all points of distance at most $\varepsilon \cdot \mathrm{diam}(R)$ from the complement of R; see Fig. 2. An *ε-approximate intersection* of P with a range R is any subset $P_1 \subseteq P$ with $P \cap R^- \subseteq P_1 \subseteq P \cap R^+$. An *$\varepsilon$-approximate answer* to the query with range R is any weight of the form $\sum_{p \in P_1} w(p)$, where P_1 is an ε-approximate intersection of P with R. Arya and Mount assume that for a given d-dimensional axis-parallel cube Q, it can be decided in unit time whether $Q \subseteq R^+$ and also whether $Q \cap R^- = \emptyset$. They show that after $O(n \log n)$ preprocessing time, a data structure of size $O(n)$ can be built, such that an ε-approximate answer to any query $R \in \mathcal{R}$ can be computed in time $O(\log n + \varepsilon^d)$. The query time bound improves to $O(\log n + \varepsilon^{d-1})$ if all $R \in \mathcal{R}$ are convex. A reporting version of the query is also possible; namely, the

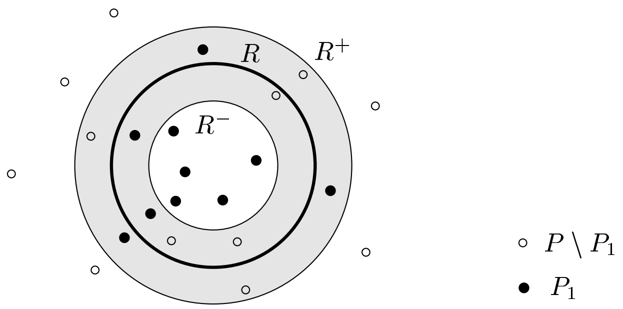


Figure 2: Approximate range searching.

list of points of a set P_1 as above can be computed in $O(\log n + \varepsilon^d + |P_1|)$ time (or in $O(\log n + \varepsilon^{d-1} + |P_1|)$ time for convex ranges).

Spanners. Let X be a set of n points in $\mathbf{R}^d$. A graph $G = (X, E)$ is called a (Euclidean) *t-spanner* of X if for any two points $x, y \in X$, there is a path from x to y in G of length at most $t\|x - y\|$, where the length of a path is the sum of the Euclidean lengths of its edges. Point sets in low-dimensional spaces admit sparse spanners. A recent strong result, subsuming most of the previous work, is due to Arya et al. [3]: for any fixed $\varepsilon > 0$ and $d > 0$ and for any n-point set in $\mathbf{R}^d$, a $(1+\varepsilon)$-spanner of maximum degree $O(1)$ can be computed in $O(n \log n)$ time. We will only need the much weaker result that a 2-spanner with $O(n)$ edges can be computed in $O(n \log n)$ time. (And, actually, spanners are not crucial in our applications; they are used mainly for convenience.)

3 Putting the points on a polynomial-size grid

Here we describe a reduction showing that for the k-clustering problem with a fixed k, it is sufficient to deal with points with polynomially large integer coordinates, after a suitable preprocessing.

Our model of computation is the *Real RAM*; we assume that the point coordinates are arbitrary real numbers. This means that ratios of the interpoint distances can be arbitrary large, say exponential in n (although this is rather unrealistic for practical inputs). A suitable preprocessing allows us to get rid of these extremely large distance ratios.

Proposition 3.1 *Let d and k_0 be fixed. Suppose that there is an algorithm A that, for a given $\varepsilon > 0$, $k \leq k_0$, and an n-point multiset $X' \subset \mathbf{R}^d$ with points lying on an integer grid of size $O(n^3/\varepsilon)$, finds a $(1+\varepsilon)$-approximately optimal k-clustering of X'. Then a $(1+\varepsilon)$-approximately optimal k_0-clustering for an arbitrary n-point set $X \subset \mathbf{R}^d$ can be computed with $O(n \log n)$ preprocessing* *and with at most C calls to algorithm A, with various at most n-point sets X', with $k \leq k_0$, and with $\alpha\varepsilon$ instead of ε, where $\alpha > 0$ and $C = C(k_0)$ are constants.*

Proof. Let G be a 2-spanner of the given set X, and let G_Δ denote G minus all edges of (Euclidean) length at least Δ. We put $\Delta = \mathrm{diam}(X)/n$, and we observe that G_Δ is necessarily disconnected.

Let X' be a (multi)set arising by moving each point of X by no more than $\delta = \alpha\varepsilon\Delta/(5n^2)$. By such a movement (and appropriate re-scaling), it can be guaranteed that the points of X' lie on an integer grid of size $O(n^3/\varepsilon)$. (Note that a multiset may result.)

We call the algorithm A on this X', obtaining a $(1 + \alpha\varepsilon)$-approximately optimal k_0-clustering Π' of X'. If $\mathrm{cost}(\Pi')$ is larger than a suitable threshold, equal to $\frac{1}{20}\Delta^2$, we show that the corresponding k_0-clustering Π of X is $(1 + \varepsilon)$-approximately optimal for X. Otherwise, if $\mathrm{cost}(\Pi')$ is below this threshold, we infer that each cluster in an optimal k_0-clustering of X is completely contained in a connected component of G_Δ. This implies that G_Δ has at most k_0 connected components. If there are exactly k_0 connected components, then the components necessarily determine the optimum clustering. Otherwise, if there are $m < k_0$ components, we recursively k-cluster each of the components, $2 \leq k \leq k_0 - m + 1$, and we combine an approximately optimal k_0-clustering from these data (in time bounded by a function of k_0).

It remains to establish the above claims. By moving the points by at most δ, the centroids are moved by no more than δ as well. By passing from X to X' or back, the square of the largest possible distance of a point from a centroid, $\mathrm{diam}(X)^2$, is changed by no more than $|\mathrm{diam}(X)^2 - (\mathrm{diam}(X) + 2\delta)^2| \leq 5\delta\,\mathrm{diam}(X)$. It follows that for any k-clustering Π of X, if Π' is the corresponding k-clustering of X', then $|\mathrm{cost}(\Pi) - \mathrm{cost}(\Pi')| \leq n \cdot 5\delta\,\mathrm{diam}(X) \leq \alpha\varepsilon\Delta^2$.

Supposing that $\mathrm{cost}(\Pi') \geq \frac{1}{20}\Delta^2$, and knowing that Π' is $(1+\alpha\varepsilon)$-approximately optimal k-clustering of X', it is easy to check that Π is $(1 + \varepsilon)$-approximately optimal clustering of X, provided that α was chosen sufficiently small.

Next, suppose that $\mathrm{cost}(\Pi') < \frac{1}{20}\Delta^2$. We infer that, for α sufficiently small, any optimal k_0-clustering Π_0 of X must satisfy $\mathrm{cost}(\Pi_0) < \frac{1}{16}\Delta^2$. This implies that the distance of any two points in the same cluster must be smaller than $\frac{1}{2}\Delta$. But by the properties of the 2-spanner G, any two points from distinct components of G_Δ have distance at least $\frac{1}{2}\Delta$. Therefore, any optimal k_0-clustering has clusters completely contained in the components of G_Δ, and we can apply the recursion as described above. This finishes the proof of Proposition 3.1. $\square$

4 Approximate centroid sets

Let S be a finite set in $\mathbf{R}^d$. We let

$$\rho(S) = \Big(\tfrac{1}{|S|}\sum_{x\in S}\|x-c(S)\|^2\Big)^{1/2}$$

be the *quadratic-mean radius* of S. For a real number $\varepsilon \geq 0$, the *ε-tolerance ball* of S is the ball centered at $c(S)$ of radius $\frac{\varepsilon}{3}\rho(S)$.

Let $X \subset \mathbf{R}^d$ and $C \subset \mathbf{R}^d$ be finite point sets. We call C an *ε-approximate centroid set for X* if C intersects the ε-tolerance ball of each nonempty $S \subseteq X$. We call C an *ε-approximate centroid set for X for cluster size $\geq s$* if it intersects the ε-tolerance ball of each cluster of size s or larger.

The following lemma uses the ideas of Inaba, Katoh, and Imai [8]:

Lemma 4.1 *Let $X \subset \mathbf{R}^d$ be a finite point set, let $k \geq 2$, and let C be an ε-approximate centroid set for X for cluster size $\geq s$. Then there are $c_1, c_2, \ldots, c_k \in C$ such that*

$$\mathrm{cost}(\Pi_{\mathrm{Vor}}(c_1, c_2, \ldots, c_k)) \leq (1+\varepsilon)\mathrm{cost}(\Pi)$$

for any k-clustering Π of X with all clusters of size at least s.

Proof. Let $(S_1, S_2, \ldots, S_k)$ be an optimal k-clustering of X with all clusters of size at least s. For $i = 1, 2, \ldots, k$, choose $c_i \in C$ lying in the ε-tolerance ball of the cluster S_i. For each i, we have

$$\begin{aligned}
\mathrm{cost}(S_i, c_i) &= \sum_{x\in S_i}\|x-c_i\|^2 \leq \sum_{x\in S_i}\Big(\|x-c(S_i)\| + \|c(S_i)-c_i\|\Big)^2 \\
&= \mathrm{cost}(S_i) + 2\|c_i - c(S_i)\|\cdot\sum_{x\in S_i}\|x-c(S_i)\| + |S_i|\cdot\|c_i-c(S_i)\|^2 \\
&\leq \mathrm{cost}(S_i) + 2\tfrac{\varepsilon}{3}\rho(S_i)\sqrt{|S_i|}\cdot\sqrt{\mathrm{cost}(S_i)} + |S_i|(\tfrac{\varepsilon}{3}\rho(S_i))^2 \\
&\leq \mathrm{cost}(S_i) + \tfrac{2}{3}\varepsilon\,\mathrm{cost}(S_i) + \tfrac{\varepsilon^2}{9}\mathrm{cost}(S_i) \\
&\leq (1+\varepsilon)\,\mathrm{cost}(S_i).
\end{aligned}$$

Let $(S'_1, S'_2, \ldots, S'_k) = \Pi_{\mathrm{Vor}}(c_1, \ldots, c_k)$. By the optimality of the Voronoi partition with respect to given centroids, we have

$$\sum_{i=1}^k \mathrm{cost}(S'_i) \leq \sum_{i=1}^k \mathrm{cost}(S'_i, c_i) \leq \sum_{i=1}^k \mathrm{cost}(S_i, c_i) \leq (1+\varepsilon)\sum_{i=1}^k \mathrm{cost}(S_i).$$

□

A simple construction for well-separated point sets. Here we give a very simple construction of an ε-approximate centroid set. This construction is suitable for a (multi)set X whose ratio of maximum and minimum interpoint distances is not extremely large. In particular, according to Proposition 3.1, for an approximate k-clustering algorithm, we may assume that X lies on an integer grid of size $O(n^4)$; then the ratio of the maximum and minimum distances of (distinct) points of X is $O(n^4)$.

The parameters of the construction are the set X, the minimum cluster size s, the number $\varepsilon > 0$, and a number $\delta > 0$ which is a lower bound for the minimum distance of two distinct points of X. (As we will see, the dependence of the quality of the construction on δ is only logarithmic, and so we can take a generous lower bound.) The set being constructed is called C.

Let us set $r = \frac{\delta}{n}$. (If X is a set, with no multiple points, it is sufficient to set $r = \frac{1}{\sqrt{2}}\delta$.) This choice guarantees that for any cluster $S \subseteq X$ with at least two distinct points, we have $\rho(S) \geq r$. This follows from the well-known equality

$$\sum_{x \in S} \|x - c(S)\|^2 = \frac{1}{2|S|} \sum_{x,y \in S} \|x - y\|^2. \tag{1}$$

In the sequel, by a cube, we will always mean an axis-parallel cube (in $\mathbf{R}^d$). The K-enlargement of a cube Q is the cube concentric with Q and with side K-times larger than the side of Q.

Let Q_0 be the 3-enlargement of the smallest cube enclosing the given set X, and let R be the side of Q_0.

Let us call a cube Q *aligned* if it arises from Q_0 by a repeated subdivision, where in each subdivision, the current cube is partitioned into 2^d equal-size cubes.

The construction begins with Q_0 as a single active cube and with $C = \emptyset$.

At each step of the construction, we consider one of the currently active cubes, denoted by Q. Let σ be the side length of Q. We choose a set C_Q that is $(\frac{\varepsilon}{18}\sigma)$-dense for the 2-enlargement of Q (Fig. 3 left); we may assume $|C_Q| = O(\varepsilon^{-d})$. We add C_Q to the constructed set C.

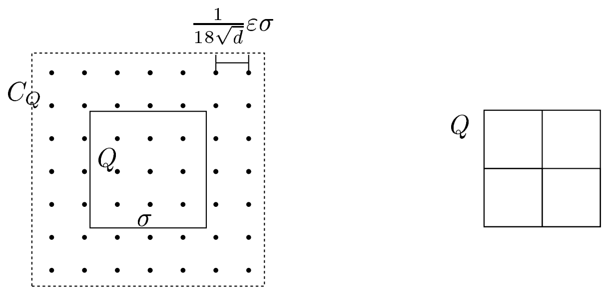


Figure 3: The set C_Q (left) and the subdivision of Q (right).

After this, the cube Q ceases to be active. If $\sigma \geq 2r$, we subdivide the cube Q into 2^d cubes of side $\frac{\sigma}{2}$ (formally, we consider the cubes to be products of semiopen intervals, so that the cubes in the subdivision are disjoint and cover Q). A cube Q' in this subdivision becomes active if and only if it contains at least $\frac{s}{2^{d+1}}$ points of X. The construction ends if there are no active cubes left.

As a final step, we add to C all the points of X of multiplicity s or larger. This takes care of all clusters S consisting of (copies of) a single point.

Lemma 4.2 *We have $|C| = O(\frac{n}{s}\,\varepsilon^{-d}\log\frac{nR}{\delta})$, and the construction can be performed in time $O((n+\frac{n}{s}\,\varepsilon^{-d})\log\frac{nR}{\delta})$. The constructed set C is an ε-approximate centroid set for X for cluster size $\geq s$.*

Proof. During the construction, we encounter active cubes with at most $O(\log\frac{R}{r})$ distinct side lengths. Since each active cube Q contains at least $\frac{s}{2^{d+1}}$ points of X, and the cubes with the same side length are disjoint, the total number of active cubes in the whole construction is $O(\frac{n}{s}\log\frac{R}{r})$, and the bound on the size of C follows. The time bound is straightforward.

Let $S \subseteq X$ be a cluster of size at least s with at least two distinct points. By Markov's inequality, the ball B of radius $\sqrt{2}\cdot\rho(S)$ centered at $c(S)$ contains at least $\frac{s}{2}$ points of S. Let $j \geq 0$ be the integer such that $\sigma = \frac{R}{2^j} \in [3\rho(S), 6\rho(S))$. Since the diameter of B is smaller than σ, the ball B intersects at most 2^d aligned cubes of side σ, and hence some of these cubes, call it Q, contains at least $\frac{s}{2^{d+1}}$ points of X. Thus, Q was an active cube sometime during the construction. The point $c(S)$ is at distance at most $\sqrt{2}\cdot\rho(S) \leq \frac{1}{2}\sigma$ from Q, and so it lies in the 2-enlargement of Q (Fig. 4). Consequently, the set $C_Q \subseteq C$ contains a point at

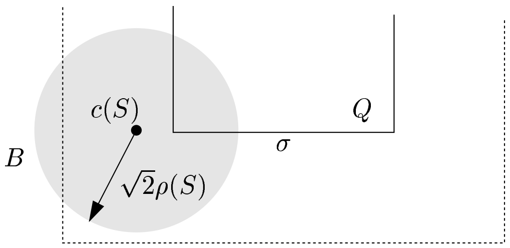


Figure 4: Illustration to the proof of Lemma 4.2.

distance at most $\frac{\varepsilon}{18}\,\sigma \leq \frac{\varepsilon}{3}\rho(S)$ from $c(S)$. Hence C intersects the ε-tolerance ball of S. $\square$

A construction for arbitrary sets. The following construction yields an ε-approximate centroid set C for an arbitrary n-point set (or multiset) X. It also gives a slightly better asymptotic bound on the size of C, but it is more complicated than the previous construction. Mainly for the sake of simplicity, we only present this construction for $s = 1$ (obtaining ε-approximate centroid set for all clusters). A generalization to an arbitrary s appears possible too, with the resulting size bound $O(\frac{n}{s}\varepsilon^{-d}\log\frac{1}{\varepsilon})$, but the proof becomes somewhat complicated.

The construction will again proceed by recursively subdividing the cube Q_0 as in the previous construction. It will be convenient to imagine that the constructed cubes are the nodes of a rooted tree (such trees are known as *quadtrees* in the literature). The cube Q_0 is the root. When a cube Q is subdivided into 2^d cubes of the same size, the cubes Q' obtained by the subdivision with $Q' \cap X \neq \emptyset$ become the sons of Q in the tree. Moreover, in order to avoid possibly infinite branches in the tree, a cube Q is not further subdivided if it contains exactly one point of X. Let $\mathcal{Q}$ denote the set of all nodes of the constructed tree.

In the previous construction, we have automatically included into C the sets C_Q for all the constructed cubes; this time we will be more selective.

Let us put $\ell = \lceil \log_2 \frac{A}{\varepsilon} \rceil$, where A is a sufficiently large constant (a suitable value can be calculated from the proof below). For a cube $Q \in \mathcal{Q}$ of side length σ, define the *periphery* of Q as the set $P_{out}(Q) \setminus P_{in}(Q)$, where $P_{in}(Q)$ is the 3-enlargement of Q and $P_{out}(Q)$ is the $2^{2\ell}$-enlargement of Q (Fig. 5). We call a cube $Q \in \mathcal{Q}$ *significant* if its periphery contains at least one point of X; otherwise, Q is *insignificant*.

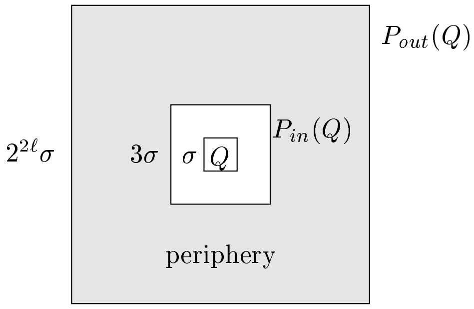


Figure 5: The periphery of a cube Q.

Now we can state the rule for constructing the set C. By saying that $Q \in \mathcal{Q}$ is *filled* we mean that the set C_Q is added to C. Each $Q \in \mathcal{Q}$ is filled *unless* there is a $\tilde{Q} \in \mathcal{Q}$ that is insignificant and lies exactly ℓ levels below Q in the tree (this means that $\tilde{Q} \subset Q$ and the side of $\tilde{Q}$ is 2^ℓ-times shorter than the side of Q). Finally, we add to C all points of X.

Lemma 4.3 *The constructed set C has size $O(n\varepsilon^{-d}\log\frac{1}{\varepsilon})$, and it is an ε-approximate centroid set for X (for all clusters).*

Proof. First we estimate the size of C. Since the leaf cubes of $\mathcal{Q}$ are disjoint and each of them contains a point of X, there are n leaves. If a cube $Q \in \mathcal{Q}$ is filled, then it is fewer than ℓ levels from a leaf or it contains a significant cube $\tilde{Q} \in \mathcal{Q}$ lying ℓ levels below it. The number of the filled cubes of the former type is at most $n\ell = O(n \log \frac{1}{\varepsilon})$. The number of the filled cubes of the latter type is no more than the number of significant cubes $\tilde{Q} \in \mathcal{Q}$, which we estimate next.

First of all, since the number of leaf cubes is at most n, there are no more than n cubes in the tree with at least two successors (the *branching cubes*). Moreover, any cube lying at most 3ℓ levels below a branching cube can be charged to that branching cube; at most $O(n\ell)$ cubes are charged in this way. Let $\mathcal{Q}_0$ be the remaining significant cubes. For $\tilde{Q} \in \mathcal{Q}_0$, we can thus assume that $\tilde{Q}$ lies in an aligned cube $\hat{Q}$ with side $2^{3\ell}\sigma$, where σ is the side of $\tilde{Q}$, such that $\hat{Q}$ contains no other points of X besides those in $\tilde{Q}$.

Let us classify the cubes in $\mathcal{Q}_0$ into $2^d 3\ell$ types, as follows. A cube $\tilde{Q} \in \mathcal{Q}_0$ lies in one of the 2^d subcubes of the corresponding $\hat{Q}$, and the cubes in $\mathcal{Q}_0$ are classified according to the subcube; for example, in the plane, the four possibilities are lower left, upper left, lower right, and upper right. Next, each $\tilde{Q} \in \mathcal{Q}_0$ has a level in the tree (distance from the root), and we classify it by the remainder of the level modulo 3ℓ. It remains to show that the number of cubes in $\mathcal{Q}_0$ of any given type (given subcube and level modulo 3ℓ) is $O(n)$. This can be easily derived from Theorem 2.1 on well-separated pairs (although a more complicated direct proof is possible too).

Let $\mathcal{Q}_1 \subseteq \mathcal{Q}_0$ be the significant cubes of the considered fixed type. For each $\tilde{Q} \in \mathcal{Q}_1$, choose a pair $(x, y) \in X \times X$, where $x \in \tilde{Q}$ and y lies in the periphery of $\tilde{Q}$. We will show that the pairs for different cubes in $\mathcal{Q}_1$ cannot be, say, 1-near, and then Theorem 2.1 will imply the claimed bound.

Let $\tilde{Q}, \tilde{Q}' \in \mathcal{Q}_1$ be cubes with sides σ and σ', respectively. If $\sigma = \sigma'$ then the distance of $\tilde{Q}$ and $\tilde{Q}'$ is at least $2^{3\ell-1}$ (because their big cubes $\hat{Q}$ and $\hat{Q}'$ are disjoint), while the diameter of their peripheries is much smaller ($2^{2\ell}\sqrt{d}$). Therefore, their pairs cannot be 1-near. And if $\sigma > \sigma'$ then $\sigma \geq 2^{3\ell}\sigma'$, and the pair assigned to $\tilde{Q}'$ must be much closer than the pair assigned to $\tilde{Q}$. This shows that the total number of significant cubes is $O(n\ell)$.

It remains to prove that the constructed set C intersects the ε-tolerance balls of all clusters S. Consider a cluster $S \subseteq X$. We may assume that $\rho(S) > 0$, for otherwise S consists of (several copies of) a single point and this point is included in C by the construction. As in the proof of Lemma 4.2, let B be the ball of radius $\sqrt{2} \cdot \rho(S)$ centered at $c(S)$, and let $Q \in \mathcal{Q}$ be a cube with side length $\sigma = \frac{R}{2^j} \in [3\rho(S), 6\rho(S))$ containing some point of $S \cap B$. If Q was filled, the set C_Q intersects the ε-tolerance ball of S (as in the proof of Lemma 4.3). It remains to consider the case when Q was not filled, which means that there is an insignificant $\tilde{Q} \in \mathcal{Q}$ lying ℓ levels below Q. We fix such a $\tilde{Q}$.

We want to show that the ε-tolerance ball of the cluster S completely contains

$\tilde{Q}$. Since C must contain a point lying in $\tilde{Q}$, this will prove that C intersects the ε-tolerance ball of S.

Let $S_{in} = S \cap P_{in}(\tilde{Q})$ and $S_{ext} = S \setminus P_{out}(\tilde{Q})$. Note that $S = S_{in} \cup S_{ext}$, since $\tilde{Q}$ is insignificant and thus it has no points in its periphery. We now prove that

$$\|c(S) - c(S_{in})\| \leq \tfrac{\varepsilon}{6}\rho(S). \tag{2}$$

Choose $c(S_{in})$ as the origin of coordinates, so that $\sum_{x \in S_{in}} x = 0$. The distance of the points of S_{ext} from $c(S)$ is at least $\frac{1}{3}2^\ell\sigma \geq \frac{A}{3\varepsilon}\sigma \geq \frac{A}{18\varepsilon}\rho(S)$, and by Markov's inequality, we obtain $|S_{ext}| \leq \frac{1}{36}\varepsilon^2|S|$ (if A is sufficiently large). We calculate, using the Cauchy-Schwarz inequality,

$$\begin{aligned} |S|^2 \cdot \|c(S)\|^2 &= \Big\|\sum_{x \in S} x\Big\|^2 = \Big\|\sum_{x \in S_{ext}} x\Big\|^2 \\ &\leq |S_{ext}| \cdot \sum_{x \in S_{ext}} \|x\|^2 \leq \tfrac{1}{36}\varepsilon^2 \cdot |S|^2 \rho(S)^2. \end{aligned}$$

This proves (2). Since $c(S_{in}) \in P_{in}(\tilde{Q})$, the distance of $c(S)$ from $P_{in}(\tilde{Q})$ is at most $\frac{\varepsilon}{6}\rho(S)$. The diameter of $P_{in}(\tilde{Q})$ is $3\sqrt{d}\,\sigma 2^{-\ell} < \frac{\varepsilon}{6}\rho(S)$, and so the ball of radius $\frac{\varepsilon}{3}\rho(S)$ centered at $c(S)$ covers $\tilde{Q}$ as claimed. □

An efficient algorithmic version of the construction. Let us modify the previous construction so that it can be performed in near-linear time.

Theorem 4.4 *Given an n-point set $X \subset \mathbf{R}^d$ and an $\varepsilon > 0$, an ε-approximate centroid set for X, of size $O(n\varepsilon^{-d}\log\frac{1}{\varepsilon})$, can be computed in time $O(n\log n + n\varepsilon^{-d}\log\frac{1}{\varepsilon})$.*

Proof. The algorithmic version of the construction has two quite independent parts. In the first part, we find all the leaf cubes in $\mathcal{Q}$ and we fill the corresponding cubes at most ℓ levels above them. In the second part, we detect a superset of the significant cubes and we fill the cubes ℓ levels above them.

To find the leaf cubes, we simulate the subdivision construction, with a suitable provision for cubes with a single successor. We start with the root cube Q_0. For each of the currently active cubes Q, we first find the smallest aligned cube Q' containing all the points of $Q \cap X$ (such a *shrinking* operation is used in several papers concerning efficient quadtree constructions; see Arya and Mount [4]). Then we subdivide Q' provided that it has more than one point. In this way, all the leaves are discovered and we only generate $O(n)$ cubes in the process.

If we know the minimum and maximum coordinates of the points in Q, the shrinking operation can be implemented in $O(1)$ time (if the floor operation is allowed; see [4] for a discussion). For an efficient implementation of the subdivision operation, we can proceed as in Callahan and Kosaraju [5]. We store d doubly-linked lists with Q, the ith list containing the points of $Q \cap X$ sorted by the ith coordinate (initially, such lists for Q_0 are obtained in $O(n \log n)$ time). In order to split Q into halves in the x_1-direction, say, we first search the list sorted by the x_1-coordinate from both ends, until we find the point of splitting. If a is the size of the smaller part, this is done in $O(a)$ time. Then we delete the a points of the smaller portion from the other $d-1$ lists, also in $O(a)$ time. Such a splitting is done for each of the coordinates in turn. A simple analysis, which we omit, shows that the total time for finding all leaf cubes in $\mathcal{Q}$ is $O(n \log n)$.

In the second part, we compute a superset of the significant cubes. From the previous step, we know the branching cubes, and we can thus generate all the cubes at most ℓ levels below them.

Next, we generate a 1-complete set P of pairs for X of size $O(n)$ as in Theorem 2.1. Let $(x, y) \in P$. For each of the $2^d 3\ell$ possible types of significant cubes, we find all aligned cubes $\tilde{Q}$ of that type such that a pair (x', y') that is 1-near to (x, y) could possibly be assigned to $\tilde{Q}$.

From the distance $\|x - y\|$ and from the type, we can read off the side of $\tilde{Q}$ uniquely. The possible location of x' is a ball B around x, whose diameter is smaller than the side of the big cube $\hat{Q}$ corresponding to $\tilde{Q}$. Hence, at most 2^d aligned cubes $\hat{Q}$ can be considered. If such a $\hat{Q}$ contains no point of X it can be disregarded. If it contains a point $x \in X$, we can output the aligned cube $\tilde{Q}$ with the appropriate side containing x (of course, this $\tilde{Q}$ need not be a significant cube in our tree, but all significant cubes are certainly included).

Thus, the following problem remains to be solved: we are given a set of $O(n)$ aligned cubes and a set of n points, and we need to detect the cubes containing at least one point. This can be done easily in $O(n \log n)$ time. For example, we can first organize the cubes into a quadtree (some nodes may be missing in the regular quadtree structure but these can be added if needed). Then we traverse the quadtree with points and the quadtree with cubes simultaneously. □

5 Approximate 2-clustering

Here is an obvious algorithm for finding a $(1+\varepsilon)$-approximately optimal 2-clustering for X using the ε-approximate centroid sets constructed in Section 4.

1. Compute an ε-approximate centroid set C for X.
2. Form the set P of all pairs (c_1, c_2) of distinct points of C.
3. For each pair $(c_1, c_2) \in P$, compute the Voronoi diagram, i.e. the hyperplane h bisecting the segment $c_1 c_2$. Compute the cost of the 2-clustering given by h, and select the 2-clustering with the smallest cost.

In a direct implementation of this algorithm, we thus consider about $|C|^2$ pairs (c_1, c_2), and for each pair, we need $O(n)$ time for computing the cost of the corresponding 2-clustering. This can be improved in two respects. First, as shown by Lemma 5.1 below, instead of all pairs (c_1, c_2), it is sufficient to consider an ε-complete set of pairs for C as in Theorem 2.1 (making the approximation factor somewhat worse). And second, it is possible to use approximate range searching to approximate the cost of the 2-clustering for each pair.

Lemma 5.1 *Let $c_1, c_2 \in \mathbf{R}^d$ be points, and let c_1', c_2' be such that the pairs (c_1, c_2) and (c_1', c_2') are ε-near, $\varepsilon \leq \frac{1}{9}$. Let $\Pi = \Pi_{\mathrm{Vor}}(c_1, c_2)$ be the clustering of X by the bisecting hyperplane of c_1 and c_2, and let $\Pi' = \Pi_{\mathrm{Vor}}(c_1', c_2')$. Then* $\mathrm{cost}(\Pi') \leq (1 + 16\varepsilon)\mathrm{cost}(\Pi)$.

Proof. Let $\Pi = (S_1, S_2)$ and $\Pi' = (S_1', S_2')$. By the optimality of the Voronoi clusterings for given centroids, it suffices to show that $\mathrm{cost}(S_1', c_1) + \mathrm{cost}(S_2', c_2) \leq (1 + 16\varepsilon)[\mathrm{cost}(S_1, c_1) + \mathrm{cost}(S_2, c_2)]$. To this end, it is enough to show that for each $x \in S_1' \setminus S_1$, we have $\|x - c_1\| \leq (1 + 6\varepsilon)\|x - c_2\|$ (a symmetric argument applies for $x \in S_2' \setminus S_2$). Since $x \in S_1'$, we know that x is closer to c_1' than to c_2', i.e. $\|x - c_1'\| \leq \|x - c_2'\|$ (Fig. 6). Put $\delta = \|c_1' - c_2'\|$. We have $\|x - c_1\| \leq$

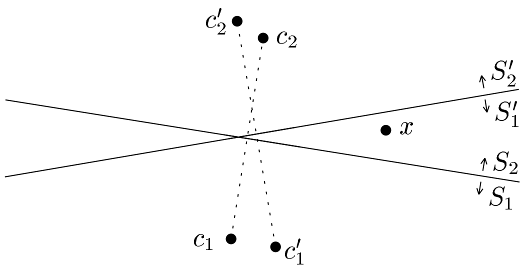


Figure 6: Illustration to the proof of Lemma 5.1.

$\|x - c_1'\| + \varepsilon\delta \leq \|x - c_2'\| + \varepsilon\delta \leq \|x - c_2\| + 2\varepsilon\delta$. Now we need to bound $\|x - c_2\|$ from below by a multiple of δ. We have $\|x - c_2\| \geq \|x - c_2'\| - \varepsilon\delta \geq \frac{1}{2}\delta - \varepsilon\delta$, and so $\delta \leq \frac{2}{1-2\varepsilon}\|x - c_2\| \leq 3\|x - c_2\|$. Thus, $\|x - c_1\| \leq (1 + 6\varepsilon)\|x - c_2\|$ as required.
□

In step 2 of the algorithm at the beginning of this section, we can thus let P be an ε-complete set of pairs for C (instead of taking all pairs). It remains to implement step 3 efficiently.

Each pair $(c_1, c_2) \in P$ defines a 2-clustering (S_1, S_2) by its bisector. If we want to approximate $\mathrm{cost}(S_1) + \mathrm{cost}(S_2)$ within the factor of $(1 + 3\varepsilon)$, say, we can afford to "misclassify" any point x whose distances to c_1 and to c_2 differ at most by the factor $(1 + \varepsilon)$. Therefore, if

$$\begin{aligned} B_1 &= \{x \in \mathbf{R}^d : (1 + \varepsilon)\|x - c_1\| \leq \|x - c_2\|\} \text{ and} \\ B_2 &= \{x \in \mathbf{R}^d : (1 + \varepsilon)\|x - c_2\| \leq \|x - c_1\|\}, \end{aligned}$$

we can use the value $\mathrm{cost}(S_1') + \mathrm{cost}(S_2')$ for any 2-clustering (S_1', S_2') such that $B_1 \cap X \subseteq S_1'$ and $B_2 \cap X \subseteq S_2'$.

Write $\delta = \|c_1 - c_2\|$. Calculation shows that B_1 is the ball of radius $r = \frac{\delta}{\varepsilon}\sqrt{1+\varepsilon}$ as in Fig 7. Moreover, the distance of the balls B_1 and B_2 is (slightly)

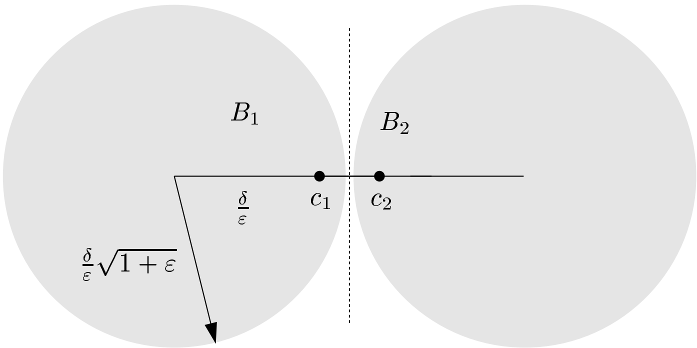


Figure 7: The balls B_1 and B_2.

smaller than $\frac{1}{2}\varepsilon\delta$. Let $\bar{B}_1$ be the ball concentric with B_1 and of radius $r + \frac{1}{4}\delta\varepsilon$. If we use η-approximate range searching (see Section 2) with $\bar{B}_1$ as the query, where η is such that $B_1 \subseteq \bar{B}_1^-$ and $B_2 \cap \bar{B}_1^+ = \emptyset$, then the answer gives us the weight of some set S_1' satisfying the above requirements. It turns out that η should be chosen of the order ε^2, which leads to the query time $O(\log n + \varepsilon^{-2(d-1)})$. (By considering the approximate range-searching algorithm in detail, one might perhaps get this down to something like $O(\log n + \varepsilon^{-(d-1)})$.)

The function cost(.) itself is not directly amenable to range searching (it is not additive). However, it can be evaluated using range searching with several auxiliary types of weights. The first weight w_1 of a point is 1, and $w_1(S) = |S|$. Further $w_2(x) = \|x\|^2$ and $w_2(S) = \sum_{x\in S}\|x\|^2$, and finally $w_3(x) = x \in \mathbf{R}^d$, $w_3(S) = \sum_{x\in S} x$. By performing approximate range searching with the composed weight $w = (w_1, w_2, w_3)$, we can calculate all of w_1, w_2, w_3 for the same set S_1'. The weights of the complementary set $S_2' = X \setminus S_1'$ can be computed from the weights of X and of S_1'. Finally, $\mathrm{cost}(S)$ for a set S can be computed from $w(S)$ using the equality (1):

$$\begin{aligned}\mathrm{cost}(S) &= \frac{1}{2|S|}\sum_{x,y\in S}\|x-y\|^2 = \frac{1}{2|S|}\Big(2|S|\cdot\sum_{x\in S}\|x\|^2 - 2\sum_{x,y\in S}\langle x,y\rangle\Big)\\ &= \sum_{x\in S}\|x\|^2 - \frac{1}{|S|}\Big\langle\sum_{x\in S} x, \sum_{x\in S} x\Big\rangle = w_2(S) - \frac{1}{w_1(S)}\|w_3(S)\|^2.\end{aligned}$$

Summarizing, it is possible to preprocess the set X in $O(n\log n)$ time in such a way that, for a given pair (c_1, c_2), $\mathrm{cost}(\Pi_{\mathrm{Vor}}(c_1, c_2))$ can be approximated within a factor of $(1+\varepsilon)$ in time $O(\log n + \varepsilon^{-2(d-1)})$.

By using this in the implementation of step 3 of the algorithm at the beginning of this section, and employing the bounds for the construction of an ε-approximate centroid set from Theorem 4.4 and from Lemma 4.2, we arrive at the total running time bounds for 2-clustering as claimed in Theorem 1.1. $\square$

6 Approximate k-clustering

In order to extend the method of the previous section to k-clustering with a fixed $k > 2$, it is natural to define that two ordered k-tuples $(c_1, c_2, \ldots, c_k)$ and $(c'_1, c'_2, \ldots, c'_k)$ are *ε-near* if for any two indices i, j, $1 \leq i < j \leq k$, the pairs (c_i, c_j) and (c'_i, c'_j) are ε-near (as in Section 2). We have the following analogue of Lemma 5.1:

Lemma 6.1 *Let $(c_1, c_2, \ldots, c_k)$ and $(c'_1, c'_2, \ldots, c'_k)$ be two k-tuples of points in $\mathbf{R}^d$ that are ε-near, $\varepsilon \leq \frac{1}{9}$. Let $\Pi = \Pi_{\mathrm{Vor}}(c_1, c_2, \ldots, c_k)$ and $\Pi' = \Pi_{\mathrm{Vor}}(c'_1, c'_2, \ldots, c'_k)$ be the respective Voronoi clusterings of a set X. Then* $\mathrm{cost}(\Pi') \leq (1+16\varepsilon)\mathrm{cost}(\Pi)$.

Sketch of proof. As in the proof of Lemma 5.1, we estimate $\sum_{i=1}^{k} \mathrm{cost}(S'_i, c_i)$, where S'_i are the clusters of Π'. Considering a point x lying in S_i and in S'_j, the calculation in the proof of Lemma 5.1 shows that $\|x - c_j\|^2 \leq (1 + 16\varepsilon)\|x - c_i\|^2$. $\square$

Well-spread k-tuples. We can proceed to define an ε-complete set of k-tuples for a set C in an obvious manner. But, heading for a near-linear approximation algorithm, we cannot afford to compute all k-tuples in such an ε-complete set, because their number can be too large. The simplest example occurs for $k = 4$, with 4-tuples (c_1, c_2, c_3, c_4) where c_1 is very close to c_2 and c_3 very close to c_4, but these two pairs lie relatively far apart (Fig. 8). One can construct an example of

Figure 8: Obtaining many well-separated 4-tuples.

an n-point set C and $\Omega(n^2)$ 4-tuples of its points such that no two of them are 1-near, say. On the other hand, the "fine structure" of the two subsets $\{c_1, c_2\}$ and $\{c_3, c_4\}$ as in Fig. 8 does not matter for their mutual interaction; we can use any of the pairs (c_1, c_3), (c_1, c_4), (c_2, c_3), and (c_2, c_4) for defining a bisector that separates the clusters of c_1 and of c_2 from the clusters of c_3 and of c_4. This motivates the following definition.

For a real number $M > 0$ and sets $Y \subseteq X \subset \mathbf{R}^d$, we say that Y is *M-isolated* in X if any point of $X \setminus Y$ has distance at least $M \cdot \mathrm{diam}(Y)$ from Y. We note that if $Y \subset X$ is $\frac{1}{\varepsilon}$-isolated in X, and $x \in X \setminus Y$, then all the pairs (y, x) with $y \in Y$ are ε-near.

Let us say that $X \subset \mathbf{R}^d$ is *ε-well-spread* if there is no proper subset $Y \subset X$, $|Y| \geq 2$, that is $\frac{1}{\varepsilon}$-isolated in X.

Lemma 6.2 *Let X be a k-point ε-well-spread set, $\varepsilon \leq \frac{1}{8}$, and let δ be the minimum distance of two (distinct) points of X. Then $\mathrm{diam}(X) \leq (\frac{2}{\varepsilon})^{k-2}\delta$.*

Proof. Consider a Euclidean minimum spanning tree T of X. Let e_1 be the shortest of its edges, and for $i \geq 1$, let e_{i+1} be the shortest edge connecting the subgraph T_i of T induced by the edges $e_1, \ldots, e_i$ to the rest of T. Since for $i = 2, 3, \ldots, k-1$, $V(T_i)$ is not $\frac{1}{\varepsilon}$-isolated in X, we get that $\|e_{i+1}\| \leq \frac{1}{\varepsilon} \mathrm{diam}(T_i) \leq \frac{1}{\varepsilon}(\|e_1\| + \|e_2\| + \cdots + \|e_i\|)$. The lemma now follows by induction on i. $\square$

Lemma 6.3 *Let $C \subset \mathbf{R}^d$ be an m-point set, let $\varepsilon < \frac{1}{8}$, and let $k \geq 2$ be fixed. Then one can compute a set $\mathcal{C}$ of ordered k-tuples with the following properties:*

(i) For any ε-well-spread k-tuple of points of C, there is a k-tuple in $\mathcal{C}$ that is ε-near to it.

(ii) $|\mathcal{C}| = O(m\varepsilon^{-k^2 d})$.

(iii) Each k-tuple in $\mathcal{C}$ is $\frac{\varepsilon}{2}$-well-spread.

(iv) At least one point in each k-tuple in $\mathcal{C}$ belongs to C.

(v) For any given k-tuple of points in $\mathbf{R}^d$, there are no more than $O(1)$ k-tuples of $\mathcal{C}$ lying ε-near to it.

(vi) The minimum and maximum distance of points in each k-tuple in $\mathcal{C}$ are bounded by constant multiples of the minimum and maximum distance in C, respectively.

The running time is $O(m \log m + m\varepsilon^{-k^2 d})$.

Proof. We use an algorithm for generating an $\frac{\varepsilon}{2}$-complete set $P \subseteq C \times C$ of pairs for C, as in Theorem 2.1. For each pair $(x, y) \in P$, we further compute k-tuples having x and y as points with the smallest (or almost the smallest) distance. We put $\delta = \|x - y\|$ and $R = 2\delta(\frac{2}{\varepsilon})^{k-2}$ and we choose a $\frac{1}{4}\varepsilon\delta$-dense set D for the ball of radius R centered at x. We output all k-tuples that consist of x, of y, and of some $k-2$ points of D (in some arbitrary order), such that: every two points of the k-tuple have distance at least $\frac{1}{4}\delta$, the k-tuple is $\frac{\varepsilon}{2}$-well-spread, and it has diameter at most $2\mathrm{diam}(C)$.

The number of generated k-tuples is

$$O\Big(m\varepsilon^{-d}(\tfrac{R}{\varepsilon\delta})^{(k-2)d}\Big) = O\Big(m\varepsilon^{-d-(k-1)(k-2)d}\Big) = O(m\varepsilon^{-k^2 d}).$$

Given an ε-well-spread k-tuple $(c_1, c_2, \ldots, c_k)$, $c_1, \ldots, c_k \in C$, with $\delta = \|c_1 - c_2\|$ being the shortest distance, we know that there is a pair $(c_1', c_2') \in P$ $\frac{\varepsilon}{2}$-near to (c_1, c_2), with $\delta' = \|c_1' - c_2'\| \in [\delta - 2\varepsilon\delta, \delta + 2\varepsilon\delta] \subseteq [\frac{1}{2}\delta, 2\delta]$. In the corresponding set D, there are points $c_3', c_4', \ldots, c_k'$, c_i' being at distance at most $\frac{1}{4}\varepsilon\delta' \leq \frac{1}{2}\varepsilon\delta$ from c_i (using Lemma 6.2). It is routine to check that the k-tuple $(c_1', c_2', \ldots, c_k')$ has all interpoint distances at least $\frac{1}{2}\delta \geq \frac{1}{4}\delta'$, diameter no larger than $2\,\mathrm{diam}(C)$, and that it is $\frac{\varepsilon}{2}$-well-spread, and so it was output by the algorithm.

Let $\mathcal{C}_0$ denote the resulting set of k-tuples. This set satisfies all the conditions in the lemma except possibly for (v): the k-tuples can sometimes be cluttered together. We run a pruning algorithm on $\mathcal{C}_0$. Let $(c_1, c_2, \ldots, c_k) \in \mathcal{C}_0$. To each point c_i, we assign an aligned cube Q_i containing c_i. The side of Q_i is determined as follows: if δ_i is the distance to c_i of the nearest neighbor of c_i among the c_j, $j \neq i$, then the diameter of Q_i should be between $\frac{1}{4}\varepsilon\delta_i$ and $\frac{1}{2}\varepsilon\delta_i$. We call two k-tuples in $\mathcal{C}_0$ equivalent if their corresponding k-tuples of aligned cubes are identical. From each equivalence class, we discard all k-tuples but one (computationally, this can be done in $O(n \log n)$ time). It can be checked that the resulting set $\mathcal{C}$ of k-tuples satisfies both (i) and (v). $\square$

The algorithm. Now we can formulate a high-level description of a recursive algorithm for k-clustering. The parameters of the recursive procedure are: a set $X \subset \mathbf{R}^d$ (the points), a set $C \subset \mathbf{R}^d$ (candidates for centroids), and the number of clusters k (ε is regarded as a global parameter). The result is a k-clustering of X and its cost. Initially, the algorithm is called with the given set X, an ε-approximate centroid set C for X, and with the given k. We need to assume that the ratio of the maximum and minimum distances in X is polynomially bounded (see Section 3); in such case, the construction of C (Theorem 4.4) can be implemented in such a way that C also has this property.

The procedure operates as follows.

1. If $k = 1$ then return X (as the single cluster) and $\mathrm{cost}(X)$. Otherwise continue with the next step.

2. For $k^* = 2, 3, \ldots, k$, generate a set $\mathcal{C}^*$ of ordered k^*-tuples for C as in Lemma 6.3.

3. For each $(c_1, c_2, \ldots, c_{k^*}) \in \mathcal{C}^*$, let $(X_1, X_2, \ldots, X_{k^*}) = \Pi_{\mathrm{Vor}}(c_1, c_2, \ldots, c_{k^*})$ be the Voronoi partition of X, and let C_i be the points of C lying in the $\varepsilon\delta$-neighborhood of c_i, where δ is the distance of the two closest points among $c_1, c_2, \ldots, c_{k^*}$ (see the schematic illustration in Fig. 9). If $C_i = \emptyset$ for some i, we disregard the current k^*-tuple $(c_1, c_2, \ldots, c_{k^*})$, otherwise we process it according to the next step.

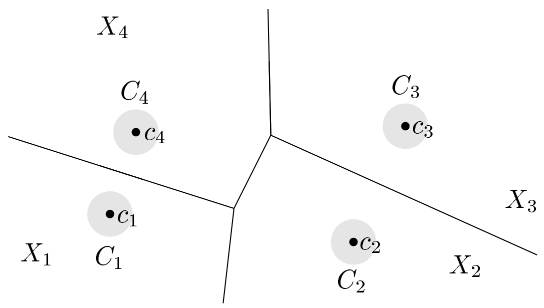


Figure 9: The recursion in the k-clustering algorithm.

4. For $i = 1, 2, \ldots, k^*$, call the procedure recursively on X_i and C_i, with the number of clusters k_i running from 1 to $k - k^* + 1$. This yields a k_i-clustering of X_i for all the considered k_i. Find the combination $(k_1, k_2, \ldots, k_{k^*})$, $k_1 + k_2 + \cdots + k_{k^*} = k$, for which the k-clustering of X obtained by combining the computed k_i-clusterings of the X_i's has the smallest cost.

5. For each k^*-tuple in $\mathcal{C}^*$ with all the C_i nonempty, we thus obtain one k-clustering. Among these, return the one with the smallest cost.

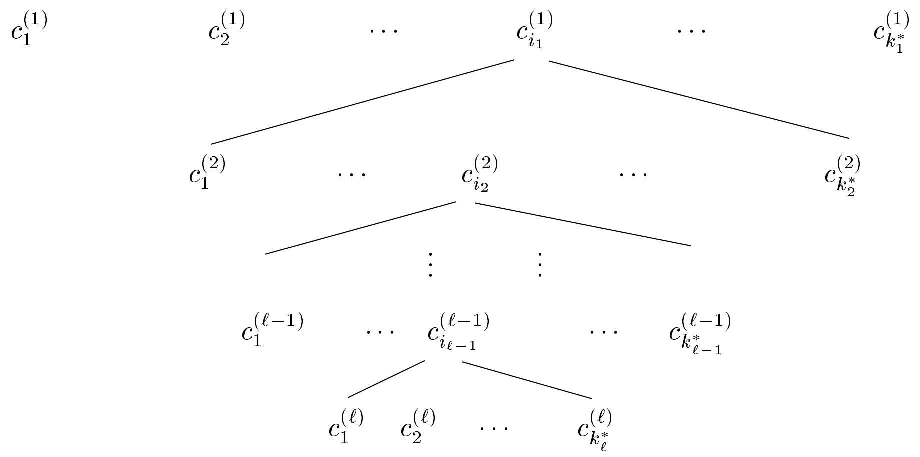
Using Lemmas 6.1 and 6.3, one can check that for ε below a suitable constant, the algorithm computes a $(1 + O(\varepsilon))$-approximately optimal k-clustering of X. It remains to specify the implementation of the various steps in more detail and bound the running time.

Implementation and running time analysis, ignoring cost computations. In the first part of the running time analysis, let us ignore the cost of step 1 (computing $\mathrm{cost}(X)$) and the cost of computing the Voronoi partition $\Pi_{\mathrm{Vor}}(c_1, c_2, \ldots, c_{k^*})$ in step 3; in the final version of the algorithm, the set X will be passed only implicitly to the recursive procedure and the cluster cost will be approximated using range searching.

The sets C_i in step 3 are computed using approximate range searching (the reporting version), say with $\varepsilon = \frac{1}{4}$ (the algorithm still works correctly if we take C_i as a $\frac{1}{4}$-approximate intersection of C with the $2\varepsilon\delta$-ball around c_i). The set C at the top level of the algorithm is preprocessed, and the queries are always made with respect to this original set; this is not exactly as specified in the algorithm, but it works just as well. Also, we do not generate C_i if $k_i = 1$. Thus, computing the C_i for one k^*-tuple in $\mathcal{C}^*$ needs $O(\log n)$ overhead, plus time proportional to the combined size of the C_i. If the queries are processed simultaneously for all i, we can make sure that whenever some C_i is empty, the total time for the queries is only $O(\log n)$. If we charge time proportional to $|C_i|$ to the recursive call with C_i, we get that the overhead for the computation of the C_i is $O(\log n)$ per k^*-tuple in $\mathcal{C}^*$.

Thus, the running time in a recursive call, excluding the time for the embedded recursive calls (and, as stated above, ignoring the computations involving X), is at most $O(|C|(\log n + \varepsilon^{-k^2 d}) + |\mathcal{C}^*| \log n)$. In order to bound the total running time, we need to consider the total size of the sets $\mathcal{C}^*$ generated by the algorithm.

Charging running time to tufts. Let us look at how a k^*-tuple $\mathbf{c}$ in some $\mathcal{C}^*$ can arise in the algorithm. At the top level of the recursion, we choose some k_1^*-tuple $\mathbf{c}^{(1)} = (c_1^{(1)}, \dots, c_{k_1^*}^{(1)})$ and an index i_1. We consider a recursive call belonging to $c_{i_1}^{(1)}$, i.e. with the sets X_{i_1} and C_{i_1}. In this second-level recursive call, we choose a k_2^*-tuple $\mathbf{c}^{(2)}$ (lying in a small neighborhood of the point $c_{i_1}^{(1)}$) and an index i_2, and we proceed with the third-level recursive call, etc., until we finally reach our k^*-tuple $\mathbf{c} = \mathbf{c}^{(\ell)}$ at some ℓth level of recursion, $\ell \le k$. Schematically:



We have $(k_1^* - 1) + (k_2^* - 1) + \cdots + (k_{\ell-1}^* - 1) + k_\ell^* \le k$.

By Lemma 6.3, for each of the k^*-tuples generated in step 2, at least of its points lies in C. Let i_ℓ be an index of a point lying in C in our k^*-tuple $\mathbf{c} = \mathbf{c}^{(\ell)}$, i.e. such that $c_{i_\ell}^{(\ell)} \in C$.

For $j = 1, 2, \dots, \ell$, let $L_j = L_j(\mathbf{c}) = \{c_i^{(j)} : i = 1, 2, \dots, k_j^*,\ i \ne i_j\}$, and let $r = r(\mathbf{c}) = c_{i_\ell}^{(\ell)}$. The total size of the L_j is at most $k - 1$, and each $L_j \cup \{r\}$ is $\frac{\varepsilon}{3}$-well-spread.

For a point x and finite sets $L_1, L_2, \dots, L_\ell$, call $(r, L_1, \dots, L_\ell)$ a (k, ε)-*tuft* if $|L_1| + |L_2| + \cdots + |L_\ell| < k$ and each $L_j \cup \{r\}$ is ε-well-spread; r is the *root* of the tuft.

In this way, each k^*-tuple $\mathbf{c}$ generated by the algorithm is assigned a $(k, \frac{\varepsilon}{3})$-tuft whose root lies in C. One tuft is assigned to at most $O(1)$ k^*-tuples (the sets in a tuft are unordered, while the algorithm formally deals with ordered k^*-tuples, and so each tuft can be ordered in a number of ways). Let $\mathcal{T}_A$ be the set of all the tufts produced by the algorithm in this manner.

The total size of the sets $\mathcal{C}^*$ generated by the algorithm is at most proportional to $|\mathcal{T}_A|$. We need to verify that also the total size of all the sets C entering the recursive calls (only those with $k > 1$) is at most proportional to $|\mathcal{T}_A|$ (note that, in some recursive calls, it might happen that $|\mathcal{C}^*|$ is much smaller than $|C|$). Suppose that C has been passed to some recursive call of the procedure, say to a call on the qth level of recursion. This call is uniquely identified by the k_j^*-tuples $\mathbf{c}^{(1)}, \mathbf{c}^{(2)}, \ldots, \mathbf{c}^{(q-1)}$ and the corresponding indices $i_1, i_2, \ldots, i_{q-1}$ generated at the previous levels of recursion. We may assume that $|C| \geq 2$ (otherwise the size of C can be charged to the calling procedure rather than to the called one). Then it can be checked that for each point $x \in C$, there is a tuft in $\mathcal{T}_A$ having x as the root and containing the points of each $\mathbf{c}^{(j)}$ (excluding $c_{i_j}^{(j)}$), $j = 1, 2, \ldots, q-1$. Indeed, a pair (x, y), where y lies close to some of the nearest neighbors of x, is generated at the qth level of recursion (for $k_q^* = 2$), and x and y together with the points of $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(q-1)}$ form the desired tuft in $\mathcal{T}_A$. The total running time of the k-clustering algorithm (still ignoring the operations with X) is thus $O(|\mathcal{T}_A|(\log n + \varepsilon^{-k^2 d}))$.

We call two (k, ε)-tufts $(r, L_1, \ldots, L_\ell)$ and $(r', L_1', \ldots, L_\ell')$ η-near if, for each $j = 1, 2, \ldots, \ell$, $|L_j| = |L_j'|$ and the $(|L_j| + 1)$-tuples $\{r\} \cup L_j$ and $\{r'\} \cup L_j'$ are η-near (in some ordering of its points, such that r is matched with r'). We note that by the construction of the sets $\mathcal{C}^*$ in the algorithm, at most $O(1)$ tufts of $\mathcal{T}_A$ are ε-near to any given $(k, \frac{\varepsilon}{3})$-tuft (by Lemma 6.3(v)).

Lemma 6.4 *Let $r \in \mathbf{R}^d$ be fixed. Let $\mathcal{T}$ be a set of (k, ε)-tufts with root r such that for each $(r, L_1, \ldots, L_\ell) \in \mathcal{T}$, $\mathrm{diam}(\{r\} \cup L_1 \cup \ldots \cup L_\ell) \leq R$ and any two points in each $L_j \cup \{r\}$ have distance at least δ ($R \geq 2\delta$). Suppose that no two tufts in $\mathcal{T}$ are ε-near. Then $|\mathcal{T}| = O((\log \frac{R}{\delta})^{k-1} \varepsilon^{-(k-1)^2 d})$.*

Proof. Consider the tufts in $\mathcal{T}$ with a given $\ell \leq k-1$ and with given sizes of the L_j. For a given tuft $(r, L_1, \ldots, L_\ell)$, let δ_j be the smallest distance in $\{r\} \cup L_j$ and let i_j be the largest integer with $2^{i_j} \leq \delta_j$. By the assumption on $\mathcal{T}$, there are at most $\log_2 \frac{2R}{\delta}$ possible values of each i_j. If i_j is fixed, by Lemma 6.2 we know that the points of L_j lie in the ball around r of radius $(\frac{2}{\varepsilon})^{|L_j|-1} 2^{i_j+1}$, and it suffices to determine their position with accuracy $\varepsilon 2^{i_j}$. It follows that the number of choices for L_j is $O(\log \frac{R}{\delta} \cdot \varepsilon^{-d|L_j|^2})$. The bound in the lemma follows (actually, it is an overestimate, since if e.g. the power of the logarithm is $k-1$ then the power of $\frac{1}{\varepsilon}$ is at most $(k-1)d$). $\square$

For the tufts in $\mathcal{T}_A$ (generated by the algorithm), we know that $\log \frac{R}{\delta} = O(\log n)$. It follows that $|\mathcal{T}_A| = O(|C|(\log |C|)^{k-1} \varepsilon^{-(k-1)^2 d})$.

If the operations concerning X, which were ignored so far, are implemented in a straightforward manner, i.e. in $O(n)$ time each, they add at most $O(n)$ as a multiplicative factor, and so the total time is slightly superquadratic. To do better, we can again use approximate range searching.

Approximate cost computation. In order to present this part of the algorithm, we need to say a little more about the approximate range-searching algorithm considered in Section 2. For our purposes, we can regard its data structure as a rooted tree T (called a *partition tree*), where each node is associated with a subset of the given set P. The root is associated with the whole P. Each node has two sons; the subsets of the sons are disjoint and together cover the subset of their parent node. Leaves of the tree are associated with one-point subsets. When processing a query with a range R, the algorithm visits the nodes of some rooted subtree T_R of T; for each visited node, either both sons are visited or none of them. Each leaf of T_R (i.e. each visited node whose sons were not visited) is labeled either IN or OUT. The approximate answer to the query is computed by summing up the weights of all the visited nodes labeled IN. The total number of visited nodes, for any convex query range, is $O(\log n + \varepsilon^{-(d-1)})$.

From this description, it is easy to see that this data structure allows us to process queries with intersections and complements of query ranges. For example, if R_1 and R_2 are convex ranges, we can compute $w(P_1 \cap P_2)$ in $O(\log n + \varepsilon^{-(d-1)})$ time, where P_1 is an ε-approximate intersection of P with R_1 and P_2 is an ε-approximate intersection with R_2. Namely, we compute the visited subtrees T_{R_1} and T_{R_2}, and we look at the leaves of their union. For the answer, we take all the leaves that are labeled IN according to both T_{R_1} and T_{R_2}; here a node of T is considered to be labeled IN according to T_{R_1} if it is a leaf of T_{R_1} labeled IN or a descendant of such a leaf of T_{R_1}.

In the recursive k-clustering algorithm, we first build the partition tree T for the original (top-level) set X. The parameter X will be passed in an implicit representation to the recursive calls: it is represented as a subtree of T plus a labeling (IN/OUT) of its leaf nodes.

It remains to describe how the Voronoi partition $\Pi_{\mathrm{Vor}}(c_1, c_2, \ldots, c_k)$ of a current set X is computed. We recall the ball B_1 introduced at the end of Section 5 (Fig. 7) for c_1 and c_2 and the slightly larger ball $\bar{B}_1$ used for the queries there. We let B_{ij} and $\bar{B}_{ij}$ denote the analogously defined balls for c_i and c_j (so B_1 is the same as B_{12}).

In order to find suitable approximations to the X_i, we first compute, for all i, j, $1 \le i < j \le k$, a set A_{ij} that is an η-approximate intersection of $\bar{B}_{ij}$ with X. (This A_{ij} is represented implicitly, by a subtree of T and an IN/OUT labeling of its leaves.) Here $\eta = \Theta(\varepsilon^2)$ is chosen so that if $x \in A_{ij}$ then $\|x - c_i\| < \|x - c_j\|$, while for $x \notin A_{ij}$ we have $\|x - c_j\| \le (1+\varepsilon)\|x - c_i\|$. Now we define

$$\begin{aligned}
X'_1 &= \bigcap_{j>1} A_{1j} \\
X'_2 &= (X \setminus X'_1) \cap \Big(\bigcap_{j>2} A_{ij} \Big) \\
&\vdots \\
X'_i &= (X \setminus (X'_1 \cup \cdots \cup X'_{i-1})) \cap \Big(\bigcap_{j>i} A_{ij} \Big)
\end{aligned}$$

$$\vdots$$
$$X'_k = X \setminus (X'_1 \cup \cdots \cup X'_{k-1}).$$

The X'_i form a partition of X, and each X'_i is represented by $O(\log n + \varepsilon^{-2(d-1)})$ nodes of the partition tree. It remains to show that if a point $x \in X$ is "misclassified", i.e. it is nearest to c_i but was placed to some X'_j, then $\|x - c_j\| \leq (1 + O(\varepsilon))\|x - c_i\|$.

First suppose that $x \in X'_j \setminus X_i$, $i > j$. This immediately leads to a contradiction, since $x \in X'_j$ implies $x \in A_{ji}$ and so $\|x - c_j\| < \|x - c_i\|$.

Next, let $x \in X'_j \setminus X_i$, $i < j$. There must be a reason why x is not placed in X'_i, namely that $x \notin A_{ij_1}$ for some $j_1 > i$, implying that $\|x - c_{j_1}\| \leq (1+\varepsilon)\|x - c_i\|$. If $j = j_1$ we have what we wanted. If $j_1 > j$ then, since x is placed to X'_j, we get $x \in A_{jj_1}$, and consequently $\|x - c_j\| \leq \|x - c_{j_1}\| \leq (1+\varepsilon)\|x - c_i\|$. Finally, if $i < j_1 < j$, we can repeat the argument with j_1 instead of i. There is a $j_2 > j_1$ with $x \notin A_{j_1 j_2}$, and so $\|x - c_{j_2}\| \leq (1+\varepsilon)\|x - c_{j_1}\|$, etc. After no more than $k-1$ such steps, we reach j or beyond it, and we thus have $\|x - c_j\| \leq (1+\varepsilon)^{k-1}\|x - c_i\| = (1 + O(\varepsilon))\|x - c_i\|$ as desired. Therefore, the k-clustering algorithm using the approximate partitions described above instead of exact Voronoi partitions correctly computes a $(1 + O(\varepsilon))$-approximately optimal k-clustering.

Each operation with X (cost computation or partitioning) is performed in time $O(\log n + \varepsilon^{-2(d-1)})$, using the implicit representations of the X'_i and the weight computations described at the end of Section 5. At the same time, each such operation can be uniquely attributed to some tuft in $\mathcal{T}_A$. For the total running time of the k-clustering algorithm we thus get the estimate

$$\begin{aligned}
O(|\mathcal{T}_A|(\log n + \varepsilon^{-2(d-1)} + \varepsilon^{-k^2 d})) &= O(|C|(\log|C|)^{k-1}\varepsilon^{-(k-1)^2 d}(\log n + \varepsilon^{-k^2 d})) \\
&= O(n\varepsilon^{-d}(\log n)^k \varepsilon^{-(k-1)^2 d - k^2 d}) \\
&= O(n(\log n)^k \varepsilon^{-2k^2 d}).
\end{aligned}$$

Theorem 1.2 is proved. □

References

[1] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–11, 1996.

[2] S. Arora, P. Raghavan, and S. Rao. Polynomial time approximation schemes for the Euclidean k-medians problem. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998.

[3] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. In *Proc. of the 27th Annual ACM Symposium on the Theory of Computing*, pages 489–498, 1995.

[4] S. Arya and D. Mount. Approximate range searching. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 172–181, 1995.

[5] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[6] V. Capoyleas, G. Rote, and G. Woeginger. Geometric clusterings. *J. Algorithms*, 12:341–356, 1991.

[7] S. Hasegawa, H. Imai, M. Inaba, N. Katoh, and J. Nakano. Efficient algorithms for variance-based k-clustering. In *Proc. First Pacific Conf. Computer Graphics Appl., Seoul, Korea, vol. 1*, pages 75–89. World Scientific Publishing Co., 1993.

[8] M. Inaba, N. Katoh, and H. Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 332–339, 1994.

[9] S. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. In *Proc. 7th Ann. Europ. Sympos. on Algorithms*, 1999.

[10] K. Varadarajan. Private communication, May 1999.

Appendix:
Approximate 2-clustering according to K. Varadarajan

Here is an outline of Varadarajan's argument, giving an improvement and simplification of the first part of Theorem 1.1.

Consider the optimal two partition $\Pi = (S_1, S_2)$ of the given n-point set X. Let h be the perpendicular bisector of the segment s connecting the centroids $c(S_1)$ and $c(S_2)$, and let m be the intersection of s. Let h' be a hyperplane passing through m, such that the angle of the normal vectors of h and of h' is at most ε. Then, as can be easily derived fom Lemma 5.1, the clustering induced by h' is $(1 + O(\varepsilon))$-approximately optimal.

Here is an algorithm for computing an approximate 2-clustering. Choose a set D of $O(\varepsilon^{-(d-1)})$ direction vectors such that any direction vector has angle at most ε with some of the chosen vectors. For each $v \in D$, sweep a hyperplane perpendicular to v across the point set X, and compute the costs of each of the n resulting 2-clusterings of X, and take the minimum. Costs of the 2-clusterings can be computed using a 1-dimensional range searching data structure (as described in the paper), or it can be updated incrementally as points are moved accross the hyperplane. The resulting running time is $O(n \log n\varepsilon^{-(d-1)})$, plus time polynomial in ε^{-1} for constructing D.