On Coresets for k-Means and k-Median Clustering*

Sariel Har-Peled*
Department of Computer Science,
University of Illinois, Urbana-Champaign
Urbana, IL 61801
sariel@uiuc.edu

Soham Mazumdar
Department of Computer Science,
University of Illinois, Urbana-Champaign
Urbana, IL 61801
smazumda@uiuc.edu

ABSTRACT

In this paper, we show the existence of small coresets for the problems of computing k-median and k-means clustering for points in low dimension. In other words, we show that given a point set P in $\mathbb{R}^d$, one can compute a weighted set $\mathcal{S} \subseteq P$, of size $O(k\varepsilon^{-d} \log n)$, such that one can compute the k-median/means clustering on $\mathcal{S}$ instead of on P, and get an $(1+\varepsilon)$-approximation.

As a result, we improve the fastest known algorithms for $(1+\varepsilon)$-approximate k-means and k-median. Our algorithms have *linear* running time for a fixed k and ε. In addition, we can maintain the $(1+\varepsilon)$-approximate k-median or k-means clustering of a stream when points are being *only inserted*, using polylogarithmic space and update time.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Non-numerical Algorithms and Problems*

General Terms

Algorithms

Keywords

Coreset, k-median, k-means, Streaming, Clustering

1. INTRODUCTION

Clustering is a widely used technique in Computer Science with applications to unsupervised learning, classification, data mining and other fields. We study two variants of the clustering problem in the geometric setting. The *geometric k-median clustering* problem is the following: Given a set P of points in $\mathbb{R}^d$, compute a set of k points in $\mathbb{R}^d$ such that the sum of the distances of the points in P to their respective nearest median is minimized. The *k-means* problem differs from the above in that instead of the sum of distances, we minimize the sum of squares of distances. Interestingly the 1-mean is the center of mass of the points, while the 1-median problem, also known as the Fermat-Weber problem, has no such closed form. As such the problems have usually been studied separately from each other even in the approximate setting. We propose techniques which can be used for finding approximate k centers in both variants.

In the data stream model of computation, the points are read in a sequence and we desire to compute a function, clustering in our case, on the set of points seen so far. In typical applications, the total volume of data is very large and can not be stored in its entirety. Thus we usually require a data-structure to maintain an aggregate of the points seen so far so as to facilitate computation of the objective function. Thus the standard complexity measures in the data stream model are the storage cost, the update cost on seeing a new point and the time to compute the function from the aggregated data structure.

k-median clustering. The k-median problem turned out to be nontrivial even in low dimensions and achieving a good approximation proved to be a challenge. Motivated by the work of Arora [4], which proposed a new technique for geometric approximation algorithms, Arora, Raghavan and Rao [5] presented a $O\left(n^{O(1/\varepsilon)+1}\right)$ time $(1+\varepsilon)$-approximation algorithm for points in the plane. This was significantly improved by Kolliopoulos and Rao [26] who proposed an algorithm with a running time of $O(\varrho n \log n \log k)$ for the discrete version of the problem, where the medians must belong to the input set and $\varrho = \exp\left[O\left((1+\log 1/\varepsilon)/\varepsilon\right)^{d-1}\right]$. The k-median problem has been studied extensively for arbitrary metric spaces and is closely related to the un-capacitated facility location problem. Charikar *et al.* [12] proposed the first constant factor approximation to the problem for an arbitrary metric space using a natural linear programming relaxation of the problem followed by rounding the fractional solution. The fastest known algorithm is due to Mettu and Plaxton [29] who give an algorithm which runs in $O(n(k+\log n))$ time for small enough k given the distances are bound by $2^{O(n/\log(n/k))}$. It was observed that if the constraint of having exactly k-medians is relaxed, the problem becomes considerably easier [11, 24]. In particular, Indyk [22] proposed a constant factor approximation algo-

*The full version of the paper is available from the author webpage http://valis.cs.uiuc.edu/~sariel/research/papers/03/kcoreset/.

*Work on this paper was partially supported by a NSF CAREER award CCR-0132901.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

Problem	Prev. Results	Our Results
k-median	$O(\varrho n(\log n)\log k)$ [26] (*) $\varrho = \exp\left[O\left((1+\log 1/\varepsilon)/\varepsilon\right)^{d-1}\right]$	$O\left(n + \varrho k^{O(1)}\log^{O(1)} n\right)$ [Theorem 5.5]
discrete k-median	$O(\varrho n \log n \log k)$ [26]	$O\left(n + \varrho k^{O(1)}\log^{O(1)} n\right)$ [Theorem 5.7]
k-means	$O_k(n(\log n)^k \varepsilon^{-2k^2 d})$ [28] (**)	$O(n + k^{k+2}\varepsilon^{-(2d+1)k}\log^{k+1} n \log^k \frac{1}{\varepsilon})$ [Theorem 6.1]
Streaming	k-median Const factor; Any metric space $O(k\text{polylog})$ space [27]	k-means and k-median $(1+\varepsilon)$-approx; Points in $\mathbb{R}^d$. $O(k\varepsilon^{-d}\log^{2d+2} n)$ space [Theorem 7.2]

Table 1: For our results, all the running time bounds are in expectation and the algorithms succeed with high probability. (*) Getting this running time requires non-trivial modifications of the algorithm of Kolliopoulos and Rao. (**) The O_k notation hides constants that depends solely on k.

rithm which produces $O(k)$ medians in $\widetilde{O}(nk)$ time. In the streaming context, Guha *et al.* [17] propose an algorithm which uses $O(n^{\varepsilon})$ memory to compute $2^{1/\varepsilon}$ approximate k-medians. Charikar *et al.* [27] improve the algorithm by reducing the space requirement to $O(k \cdot \text{polylog}(n))$.

k-means clustering. Inaba *et al.* [21] observe that the number of Voronoi partitions of k points in $\mathbb{R}^d$ is n^{kd} and can be done exactly in time $O(n^{kd+1})$. They also propose approximation algorithms for the 2-means clustering problem with time complexity $O(n^{O(d)})$. de la Vega *et al.* [13] proposes a $(1+\varepsilon)$-approximation algorithm, for high dimensions, with running time $O(n^{k/\varepsilon^2})$. Matoušek [28] proposed a $(1+\varepsilon)$-approximation algorithm for the geometric k-means problem with running time $O\left(n\varepsilon^{-2k^2 d}\log^k n\right)$.

Our Results.. We propose fast algorithms for the approximate k-means and k-medians problems in Euclidean metrics. The central idea behind our algorithms is computing a weighted point set which we call a (k,ε)-*coreset*. For an optimization problem, a coreset is a subset of input, such that we can get a good approximation to the original input by solving the optimization problem directly on the coreset. As such, to get good approximation, one needs to compute a coreset, as small as possible from the input, and then solve the problem on the coreset using known techniques. Coresets have been used for geometric approximation mainly in low-dimension [1, 18, 2], although a similar but weaker concept was also used in high dimensions [9, 8, 20]. In low dimensions coresets yield approximation algorithm with linear or near linear running time with an additional term that depends only on the size of the coreset.

In the present case, the property we desire of the (k,ε)-coreset is that the clustering cost of the coreset for any arbitrary set of k centers is within $(1\pm\varepsilon)$ of the cost of the clustering for the original input. To facilitate the computation of the coreset, we first show a linear time algorithm (for $k = O(n^{1/4})$), that constructs a $O(k\text{polylog})$ sized set of centers such that the induced clustering gives a constant factor approximation to both the optimal k-means and the optimal k-medians clustering. We believe that the technique used for this fast algorithm is of independent interest. Note that it is faster than previous published fast algorithms for this problem (see [29] and references therein), since we are willing to use more centers. Next, we show how to construct a suitably small coreset from the set of approximate centers. We compute the k clusterings for the coresets using weighted variants of known clustering algorithms. Our results are summarized in Table 1.

One of the benefits of our new algorithms is that in the resulting bounds, on the running time, the term containing 'n' is decoupled from the "nasty" exponential constants that depend on k and $1/\varepsilon$. Those exponential constants seems to be inherent to the clustering techniques currently known for those problems.

Our techniques extend very naturally to the streaming model of computation. The aggregate data-structure is just a (k,ε)-coreset of the stream seen so far. The size of the maintained coreset is $O(k\varepsilon^{-d}\log n)$, and the overall space used is $O((\log^{2d+2} n)/\varepsilon^d)$. The amortized time to update the data-structure on seeing a new point is $O(k^5 + \log^2(k/\varepsilon))$.

As a side note, our ability to get linear time algorithms for fixed k and ε, relies on the fact that our algorithms need to solve a batched version of the nearest neighbor problem. In our algorithms, the number of queries is considerably larger than the number of sites, and the distances of interest arise from clustering. Thus, a small additive error which is related to the total cost of the clustering is acceptable. In particular, one can build a data-structure that answers nearest neighbor queries in $O(1)$ time per query, see Appendix B. Although this is a very restricted case, this result may nevertheless be of independent interest, as this is the first data-structure to offer nearest neighbor queries in constant time, in a non-trivial settings.

The paper is organized as follows. In Section 3, we describe a fast constant factor approximation algorithm which generates more than k means/medians. In Section 4, we prove the existence of coresets for k-median/means clustering. In Section 5 and Section 6, we combine the results of the two preceding sections, and present an $(1+\varepsilon)$-approximation algorithm for k-means and k-median respectively. In Section 7, we show how to use coresets for space efficient streaming. We conclude in Section 8.

2. PRELIMINARIES

Definition 2.1 For a point set X, and a point p, both in $\mathbb{R}^d$, let $\mathbf{d}(p,X) = \min_{x\in X}\|xp\|$ denote the *distance of p from X*.

Definition 2.2 (Clustering) For a weighted point set P with points from $\mathbb{R}^d$, with an associated weight function $w : P \to \mathbb{Z}^+$ and any point set C, we define $\nu_C(P) = \sum_{p \in P} w(p)\mathbf{d}(p, C)$ as the *cost* of the k-median clustering provided by C. Further let $\nu_{\text{opt}}(P,k) = \min_{C \subseteq \mathbb{R}^d, |C|=k} \nu_C(P)$ denote the cost of the *optimal k-median* clustering for P.

Similarly, let $\mu_C(P) = \sum_{p \in P} w(p) (\mathbf{d}(p, C))^2$ denote the cost of the k-means clustering of P as provided by the set of centers C. Let $\mu_{\text{opt}}(P,k) = \min_{C \subseteq \mathbb{R}^d, |C|=k} \mu_C(P)$ denote the cost of the *optimal k-means clustering* of P.

We refer to $\mathrm{R}^{\nu}_{\text{opt}}(P,k) = \nu_{\text{opt}}(P,k)/|P|$ as the *average radius* of P. And to $\mathrm{R}^{\mu}_{\text{opt}}(P,k) = \sqrt{\mu_{\text{opt}}(P,k)/|P|}$ as the *average means radius* of P.

Remark 2.3 We only consider positive integer weights. A regular point set P may be considered as a weighted set with weight 1 for each point, and total weight $|P|$.

Definition 2.4 (Discrete Clustering) In several cases, it is convenient to consider the centers to be restricted to lie in the original point set. Let $\nu^D_{\text{opt}}(P,k) = \min_{C \subseteq P, |C|=k} \nu_C(P)$ denote the cost of the *optimal discrete k-median* clustering for P and let $\mu^D_{\text{opt}}(P,k) = \min_{C \subseteq P, |C|=k} \mu_C(P)$ denote the cost of the *optimal discrete k-means* clustering of P.

Observation 2.5 *For any point set P, the following hold (i) $\mu_{\text{opt}}(P,k) \le \mu^D_{\text{opt}}(P,k) \le 4\mu_{\text{opt}}(P,k)$ and (ii) $\nu_{\text{opt}}(P,k) \le \nu^D_{\text{opt}}(P,k) \le 2\nu_{\text{opt}}(P,k)$.*

3. FAST CONSTANT FACTOR APPROXIMATION WITH MORE CENTERS

Let P be the given point set in $\mathbb{R}^d$. We want to quickly compute a constant factor approximation to the k-means clustering of P, while using more than k centers. The number of centers output by our algorithm is $O\left(k \log^3 n\right)$. Surprisingly, the set of centers computed by the following algorithm is a good approximation for both k-median and k-means. To be consistent, throughout this section, we refer to k-means, although everything holds nearly verbatim for k-median as well.

Definition 3.1 (bad points) For a point set X, define a point $p \in P$ as *bad* with respect to X, if the cost it pays in using a center from X is prohibitively larger than the cost C_{opt} pays for it; more precisely $\mathbf{d}(p,X) \ge 2\mathbf{d}(p, C_{\text{opt}})$. A point $p \in P$ which is not bad, is by necessity, if not by choice, *good*. Here $C_{\text{opt}} = C_{\text{opt}}(P,k)$ is a set of optimal k-means centers realizing $\mu_{\text{opt}}(P,k)$.

3.1 Construction of the Set X of Centers

For $k = O(n^{1/4})$, we can compute a 2-approximate k-center clustering of P in linear time [18], or alternatively, for $k = \Omega(n^{1/4})$, in $O(n \log k)$ time, using the algorithm of Feder and Greene [14]. This is the *min-max clustering* where we cover P by a set of k balls such the radius of the largest ball is minimized. Let V be the set of k centers computed, together with the furthest point in P from those k centers.

Let L be the radius of this 2-approximate clustering. Since both those algorithms are simulating the (slower) algorithm of Gonzalez [16], we have the property that the minimal distance between any points of V is at least L. Thus, any k-means clustering of P, must have cost at least $(L/2)^2$, and is at most nL^2, and as such L is a rough estimate of $\mu_{\text{opt}}(P,k)$. In fact, this holds even if we restrict out attention only to V; explicitly $(L/2)^2 \le \mu_{\text{opt}}(V,k) \le \mu_{\text{opt}}(P,k) \le nL^2$.

Next, we pick a random sample Y from P of size $\rho = \gamma k \log^2 n$, where γ is a large enough constant whose value would follow from our analysis. Let $X = Y \cup V$ be the required set of cluster centers. In the extreme case where $\rho > n$, we just set X to be P.

3.2 A Large Good Subset for X

3.2.1 Bad points are few

Consider the set C_{opt} of k optimal centers for the k-means, and place a ball b_i around each point of $c_i \in C_{\text{opt}}$, such that b_i contain $\eta = n/(20k \log n)$ points of P. If γ is large enough, it is easy to see that with high probability, there is at least one point of X inside every ball b_i. Namely, $X \cap b_i \neq \emptyset$, for $i = 1, \ldots, k$.

Let P_{bad} be the set of all bad points of P. Assume, that there is a point $x_i \in X$ inside b_i, for $i = 1, \ldots, k$. Observe, that for any $p \in P \setminus b_i$, we have $\|px_i\| \le 2\|pc_i\|$. In particular, if c_i is the closest center in C_{opt} to p, we have that p is good. Thus, with high probability, the only bad points in P are the one that lie inside the balls $b_1, \ldots, b_k$. But every one of those balls, contain at most η points of P. It follows, that with high probability, the number of bad points in P with respect to X is at most $\beta = k \cdot \eta = n/(20 \log n)$.

3.2.2 Keeping Away from Bad Points

Although the number of bad points is small, there is no easy way to determine the set of bad points. We instead construct a set P' ensuring that the clustering cost of the bad points in P' does not dominate the total cost. For every point in P, we compute its approximate nearest neighbor in X. This can be easily done in $O(n \log |X| + |X| \log |X|)$ time using appropriate data structures [6], or in $O(n + n\,|X|^{1/4} \log n)$ time using Corollary B.4 (with $D = nL$). This stage takes $O(n)$ time, if $k = O(n^{1/4})$, else it takes $O(n \log |X| + |X| \log |X|) = O(n \log(k \log n))$ time, as $|X| \le n$.

In the following, to simplify the exposition, we assume that we compute exactly the distance $r(p) = \mathbf{d}(p, X)$, for $p \in P$.

Next, we partition P into classes in the following way. Let $P[a,b] = \left\{ p \in P \;\middle|\; a \le r(p) < b \right\}$. Let $P_0 = P[0, L/(4n)]$, $P_\infty = P[2Ln, \infty]$ and $P_i = P\left[2^{i-1}L/n, 2^i L/n\right]$, for $i = 1, \ldots, M$, where $M = 2\lceil \lg n \rceil + 3$. This partition of P can be done in linear time using the log and floor function.

Let P_α be the last class in this sequence that contains more than $2\beta = 2(n/(20 \log n))$ points. Let $P' = V \cup \bigcup_{i \le \alpha} P_i$. We claim that P' is the required set. Namely, $|P'| \ge n/2$ and $\mu_X(P') = O(\mu_{C_{\text{opt}}}(P'))$, where $C_{\text{opt}} = C_{\text{opt}}(P,k)$ is the optimal set of centers for P.

3.2.3 Proof of Correctness

Clearly, P' contains at least $(n - |P_\infty| - M \cdot (2n/20 \log n))$ points. Since $P_\infty \subseteq P_{\text{bad}}$ and $|P_{\text{bad}}| \le \beta$, hence $|P'| > n/2$.

If $\alpha > 0$, we have $|P_\alpha| \ge 2\beta = 2(n/(20 \log n))$. Since P' is the union of all the classes with distances smaller than the distances in P_α, it follows that the worst case scenario is when all the bad points are in P_α. But with high probability the number of bad points is at most β, and since the cost

of all the points in P_α is roughly the same, it follows that we can charge the cost of the bad points in P' to the good points in P_α.

Formally, let $Q' = P_\alpha \setminus P_{\text{bad}}$. For any point $p \in P' \cap P_{\text{bad}}$ and $q \in Q'$, we have $\mathbf{d}(p, X) \leq 2\mathbf{d}(q, X)$. Further $|Q'| > |P_{\text{bad}}|$. Thus, $\mu_X(P' \cap P_{\text{bad}}) \leq 4\mu_X(Q') \leq 16\mu_{C_{\text{opt}}}(Q') \leq 16\mu_{C_{\text{opt}}}(P')$. Thus,

$$\begin{aligned}\mu_X(P') &= \mu_X(P' \cap P_{\text{bad}}) + \mu_X(P' \setminus P_{\text{bad}}) \\ &\leq 16\mu_{C_{\text{opt}}}(P') + 4\mu_{C_{\text{opt}}}(P') = 20\mu_{C_{\text{opt}}}(P').\end{aligned}$$

If $\alpha = 0$ then for any point $p \in P'$, we have $(\mathbf{d}(p, X))^2 \leq n(L/4n)^2 \leq L^2/(4n)$ and thus $\mu_X(P') \leq L^2/4 \leq \mu_{C_{\text{opt}}}(V) \leq \mu_{C_{\text{opt}}}(P')$, since $V \subseteq P'$.

In the above analysis we assumed that the nearest neighbor data structure returns the exact nearest neighbor. If we were to use an approximate nearest neighbor instead, the constants would slightly deteriorate.

Lemma 3.2 *Given a set P of n points in $\mathbb{R}^d$, and parameter k, one can compute sets P' and $X \subseteq P$ such that, with high probability, $|P'| \geq n/2$, $|X| = O(k \log^2 n)$, and $\mu_{C_{\text{opt}}}(P') \geq \mu_X(P')/32$, where C_{opt} is the optimal set of k-means centers for P. The running time of the algorithm is $O(n)$ if $k = O(n^{1/4})$, and $O(n \log(k \log n))$ otherwise.*

Now, finding a constant factor k-median clustering is easy. Apply Lemma 3.2 to P, remove the subset found, and repeat on the remaining points. Clearly, this would require $O(\log n)$ iterations. We can extend this algorithm to the weighted case, by sampling $O(k \log^2 W)$ points at every stage, where W is the total weight of the points. Note however, that the number of points no longer shrink by a factor of two at every step, as such the running time of the algorithm is slightly worse.

Theorem 3.3 (Clustering with more centers) *Given a set P of n points in $\mathbb{R}^d$, and parameter k, one can compute a set X, of size $O(k \log^3 n)$, such that $\mu_X(P) \leq 32\mu_{\text{opt}}(P, k)$. The running time of the algorithm is $O(n)$ if $k = O(n^{1/4})$, and $O(n \log(k \log n))$ otherwise.*

Furthermore, the set X is a good set of centers for k-median. Namely, we have that $\nu_X(P) \leq 32\nu_{\text{opt}}(P, k)$.

If the point set P is weighted, with total weight W, then the size of X becomes $O(k \log^3 W)$, and the running time becomes $O(n \log^2 W)$.

4. CORESET FROM APPROXIMATE CLUSTERING

Definition 4.1 (Coreset) For a weighted point set $P \subseteq \mathbb{R}^d$, a weighted set $\mathcal{S} \subseteq \mathbb{R}^d$, is a (k, ε)-*coreset* of P for the k-median clustering, if for any set C of k points in $\mathbb{R}^d$, we have $(1 - \varepsilon)\nu_C(P) \leq \nu_C(\mathcal{S}) \leq (1 + \varepsilon)\nu_C(P)$.

Similarly, $\mathcal{S}$ is a (k, ε)-*coreset* of P for the k-means clustering, if for any set C of k points in $\mathbb{R}^d$, we have $(1-\varepsilon)\mu_C(P) \leq \mu_C(\mathcal{S}) \leq (1 + \varepsilon)\mu_C(P)$.

4.1 Coreset for k-Median

Let P be a set of n points in $\mathbb{R}^d$, and $A = \{x_1, \ldots, x_m\}$ be a point set, such that $\nu_A(P) \leq c\nu_{\text{opt}}(P, k)$, where c is a constant. We give a construction for a (k, ε)-coreset using A. Note that we do not have any restriction on the size of A, which in subsequent uses will be taken to be $O(k\text{polylog})$.

4.1.1 The construction

Let P_i be the points of P having x_i as their nearest neighbor in A, for $i = 1, \ldots, m$. Let $R = \nu_A(P)/(cn)$ be a lower bound estimate of the average radius $\mathrm{R}^{\nu}_{\text{opt}}(P, k) = \nu_{\text{opt}}(P, k)/n$. For any $p \in P_i$, we have $\|px_i\| \leq cnR$, since $\|px_i\| \leq \nu_A(P)$, for $i = 1, \ldots, m$.

Next, we construct an appropriate exponential grid around each x_i, and snap the points of P to those grids. Let $Q_{i,j}$ be an axis-parallel square with side length $R2^j$ centered at x_i, for $j = 0, \ldots, M$, where $M = \lceil 2 \lg(cn) \rceil$. Next, let $V_{i,0} = Q_{i,0}$, and let $V_{i,j} = Q_{i,j} \setminus Q_{i,j-1}$, for $j = 1, \ldots, M$. Partition $V_{i,j}$ into into a grid with side length $r_j = \varepsilon R2^j/(10cd)$, and let G_i denote the resulting exponential grid for $V_{i,0}, \ldots, V_{i,M}$. Next, compute for every point of P_i, the grid cell in G_i that contains it. For every non empty grid cell, pick an arbitrary point of P_i inside it as a representative point for the coreset, and assign it a weight equal to the number of points of P_i in this grid cell. Let $\mathcal{S}_i$ denote the resulting weighted set, for $i = 1, \ldots, m$, and let $\mathcal{S} = \cup_i \mathcal{S}_i$.

Note that $|\mathcal{S}| = O\left((|A| \log n)/\varepsilon^d\right)$. As for computing $\mathcal{S}$ efficiently. Observe that all we need is a constant factor approximation to $\nu_A(P)$ (i.e., we can assign a $p \in P$ to P_i if $\|p, x_i\| \leq 2\mathbf{d}(p, A)$). This can be done in a naive way in $O(nm)$ time, which might be quite sufficient in practice. Alternatively, one can use a data-structure that answers constant approximate nearest-neighbor queries in $O(\log m)$ when used on A after $O(m \log m)$ preprocessing [6]. Another option for computing those distances between the points of P and the set A is using Theorem B.3 that works in $O(n+mn^{1/4} \log n)$ time. Thus, for $i = 1, \ldots, m$, we compute a set P'_i which consists of the points of P that x_i (approximately) serves. Next, we compute the exponential grids, and compute for each point of P'_i its grid cell. This takes $O(1)$ time per point, with a careful implementation, using hashing, the floor function and the log function. Thus, if $m = O(\sqrt{n})$ the overall running time is $O(n+mn^{1/4} \log n) = O(n)$ and $O(m \log m + n \log m + n) = O(n \log m)$ otherwise.

4.1.2 Proof of Correctness

Lemma 4.2 *The weighted set $\mathcal{S}$ is a (k, ε)-coreset for P and $|\mathcal{S}| = O\left(|A|\varepsilon^{-d} \log n\right)$.*

PROOF. Let Y be an arbitrary set of k points in $\mathbb{R}^d$. For any $p \in P$, let p' denote the image of p in $\mathcal{S}$. The error is $\mathcal{E} = |\nu_Y(P) - \nu_Y(\mathcal{S})| \leq \sum_{p \in P} |\mathbf{d}(p, Y) - \mathbf{d}(p', Y)|$.

Observe that $\mathbf{d}(p, Y) \leq \|pp'\| + \mathbf{d}(p', Y)$ and $\mathbf{d}(p', Y) \leq \|pp'\| + \mathbf{d}(p, Y)$ by the triangle inequality. Implying that $|\mathbf{d}(p, Y) - \mathbf{d}(p', Y)| \leq \|pp'\|$. It follows that

$$\begin{aligned}\mathcal{E} &\leq \sum_{p \in P} \|pp'\| = \sum_{p \in P, \mathbf{d}(p,A) \leq R} \|pp'\| + \sum_{p \in P, \mathbf{d}(p,A) > R} \|pp'\| \\ &\leq \frac{\varepsilon}{10c} nR + \frac{\varepsilon}{10c} \sum_{p \in P} \mathbf{d}(p, A) \\ &\leq \frac{2\varepsilon}{10c} \nu_A(P) \leq \varepsilon \nu_{\text{opt}}(P, k),\end{aligned}$$

since $\|pp'\| \leq \frac{\varepsilon}{10c}\mathbf{d}(p, A)$ if $\mathbf{d}(p, A) \geq R$, and $\|pp'\| \leq \frac{\varepsilon}{10c}R$, if $\mathbf{d}(p, A) \leq R$, by the construction of the grid. This implies $|\nu_Y(P) - \nu_Y(\mathcal{S})| \leq \varepsilon\nu_Y(P)$, since $\nu_{\text{opt}}(P, k) \leq \nu_Y(P)$. □

It is easy to see that the above algorithm can be easily extended for weighted point sets.

Theorem 4.3 *Given a point set P with n points, and a point set A with m points, such that $\nu_A(P) \leq c\nu_{\text{opt}}(P,k)$, where c is a constant. Then, one can compute a weighted set $\mathcal{S}$ which is a (k,ε)-coreset for P, and $|\mathcal{S}| = O\left((|A|\log n)/\varepsilon^d\right)$. The running time is $O(n)$ if $m = O(\sqrt{n})$ and $O(n\log m)$ otherwise.*

For a weighted point set P, with total weight W, the size of the coreset is $|\mathcal{S}| = O\left((|A|\log W)/\varepsilon^d\right)$.

4.2 Coreset for k-Means from Approximate Clustering

The construction of the k-means coreset follows the k-median with a few minor modifications. Let P be a set of n points in $\mathbb{R}^d$, and a A be a point set $A = \{x_1, \ldots, x_m\}$, such that $\mu_A(P) \leq c\mu_{\text{opt}}(P,k)$. Let $R = \sqrt{(\mu_A(P)/(cn))}$ be a lower bound estimate of the average mean radius $\mathrm{R}^{\mu}_{\text{opt}}(P,k) = \sqrt{\mu_{\text{opt}}(P,k)/n}$. For any $p \in P_i$, we have $\|px_i\| \leq \sqrt{cn}R$, since $\|px_i\|^2 \leq \mu_A(P)$, for $i = 1, \ldots, m$.

Next, we construct an exponential grid around each point of A, as in the k-median case, and snap the points of P to this grid, and we pick a representative point for such grid cell. See Section 4.1.1 for details. We claim that the resulting set of representatives $\mathcal{S}$ is the required coreset.

Theorem 4.4 *Given a set P with n points, and a point set A with m points, such that $\mu_A(P) \leq c\mu_{\text{opt}}(P,k)$, where c is a constant. Then, can compute a weighted set $\mathcal{S}$ which is a (k,ε)-coreset for P, and $|\mathcal{S}| = O\left((m\log n)/(c\varepsilon)^d\right)$. The running time is $O(n)$ if $m = O(n^{1/4})$ and $O(n\log m)$ otherwise.*

If P is a weighted set with total weight W, then the size of the coreset is $O\left((m\log W)/\varepsilon^d\right)$.

We omit the proof of this theorem. It goes along the same lines as the proof for the medians case but is much more tedious.

5. PUTTING THINGS TOGETHER: A $(1+\varepsilon)$-APPROXIMATION FOR K-MEDIANS

We now present the approximation algorithm using exactly k centers. Assume that the input is a set of n points. We use the set of centers computed in Theorem 3.3 to compute a constant factor coreset using the algorithm of Theorem 4.3. The resulting coreset $\mathcal{S}$, has size $O(k\log^4 n)$. Next we compute a $O(n)$ approximation to the k-median for the coreset using the k-center (min-max) algorithm [16]. Let $C_0 \subseteq \mathcal{S}$ be the resulting set of centers. Next we apply the local search algorithm, due to Arya *et al.* [7], to C_0 and $\mathcal{S}$, where the set of candidate points is $\mathcal{S}$. This local search algorithm, at every stage, picks a center c from the current set of centers C_{curr}, and a candidate center $s \in \mathcal{S}$, and swaps c out of the set of centers and s into the set of centers. Next, if the new set of centers $C'_{curr} = C_{curr} \setminus \{c\} \cup \{s\}$ provides a considerable improvement over the previous solution (i.e., $\nu_{C_{curr}}(\mathcal{S}) \leq (1-\varepsilon/k)\nu_{C'_{curr}}(\mathcal{S})$ where ε here is an arbitrary small constant), then we set C_{curr} to be C'_{curr}. Arya *et al.* [7] showed that the algorithm terminates, and it provides a constant factor approximation to $\nu^D_{\text{opt}}(\mathcal{S},k)$, and as hence to $\nu_{\text{opt}}(P,k)$. It is easy to verify that it stops after $O(k\log n)$ such swaps. Every swap, in the worst case, requires considering $|\mathcal{S}|\,k$ sets. Computing the cost of clustering for every such candidate set of centers takes $O\left(|\mathcal{S}|\,k\right)$ time. Thus, the running time of this algorithm is $O\left(|\mathcal{S}|^2 k^3 \log n\right) = O\left(k^5 \log^9 n\right)$. Finally, we use the new set of centers with Theorem 4.3, and get a (k,ε)-coreset for P. It is easy to see that the algorithm works for weighted point-sets as well. Putting in the right bounds from Theorem 3.3 and Theorem 4.3 for weighted sets, we get the following.

Lemma 5.1 (coreset) *Given a set P of n points in $\mathbb{R}^d$, one can compute a k-median (k,ε)-coreset $\mathcal{S}$ of P, of size $O\left((k/\varepsilon^d)\log n\right)$, in time $O\left(n + k^5\log^9 n\right)$.*

If P is a weighted set, with total weight W, the running time of the algorithm is $O(n\log^2 W + k^5\log^9 W)$.

We would like to apply the algorithm of Kolliopoulos and Rao [26] to the coreset, but unfortunately, their algorithm only works for the discrete case, when the medians are part of the input points. Thus, the next step is to generate from the coreset, a small set of candidate points in which we can assume all the medians lie, and use the (slightly modified) algorithm of [26] on this set.

Definition 5.2 (Weber-Centroid Set) Given a set P of n points in $\mathbb{R}^d$, a set $\mathcal{D} \subseteq \mathbb{R}^d$ is an (k,ε)-*approximate weber-centroid set* for P, if there exists a subset $C \subseteq \mathcal{D}$ of size k, such that $\nu_C(P) \leq (1+\varepsilon)\nu_{\text{opt}}(P,k)$.

Lemma 5.3 *Given a set P of n points in $\mathbb{R}^d$, one can compute an (k,ε)-approximate weber-centroid set $\mathcal{D}$ of size $O(k^2\varepsilon^{-2d}\log^2 n)$ in time $O\left(n + k^5\log^9 n + k^2\varepsilon^{-2d}\log^2 n\right)$.*

For a weighted point set P, with total weight W, the running time is $O\left(n\log^2 W + k^5\log^9 W + k^2\varepsilon^{-2d}\log^2 W\right)$, and the weber-centroid set is of size $O(k^2\varepsilon^{-2d}\log^2 W)$.

Proof. Compute a $(k,\varepsilon/12)$-coreset $\mathcal{S}$ using Lemma 5.1. Retain the set B of k centers, for which $\nu_B(P) = O(\nu_{\text{opt}}(P,k))$, which is computed during the construction of $\mathcal{S}$. Further let $R = \nu_B(P)/n$.

Next, compute around each point of $\mathcal{S}$, an exponential grid using R, as was done in Section 4.1.1. This results in a point set $\mathcal{D}$ of size of $O(k^2\varepsilon^{-2d}\log^2 n)$. We claim that $\mathcal{D}$ is the required weber-centroid set. The proof proceeds on similar lines as the proof of Theorem 4.3 and is given in Appendix A. □

We are now in the position to get a fast approximation algorithm. We generate the centroid set, and then we modify the algorithm of Kolliopoulos and Rao so that it considers centers only from the centroid set in its dynamic programming stage. For the weighted case, the depth of the tree constructed in [26] is $O(\log W)$ instead of $O(\log n)$. Further since their algorithm works in expectation, we run it independently $O(\log(1/\delta)/\varepsilon)$ times to get a guarantee of $(1-\delta)$.

Theorem 5.4 ([26]) *Given a weighted point set P with n points in $\mathbb{R}^d$, with total weight W, a centroid set $\mathcal{D}$ of size at most n, and a parameter $\delta > 0$, one can compute $(1+\varepsilon)$-approximate k-median clustering of P using only centers from $\mathcal{D}$. The running time is $O\left(\varrho n(\log k)(\log W)\log(1/\delta)\right)$, where $\varrho = \exp\left[O\left((1+\log 1/\varepsilon)/\varepsilon\right)^{d-1}\right]$. The algorithm succeeds with probability $\geq 1-\delta$.*

The final algorithm is the following: Using the algorithms of Lemma 5.1 and Lemma 5.3 we generate a (k,ε)-coreset

$\mathcal{S}$ and an ε-centroid set $\mathcal{D}$ of P, where $|\mathcal{S}| = O(k\varepsilon^{-d}\log n)$ and $|\mathcal{D}| = O(k^2\varepsilon^{-2d}\log^2 n)$. Next, we apply the algorithm of Theorem 5.4 on $\mathcal{S}$ and $\mathcal{D}$.

Theorem 5.5 ($(1+\varepsilon)$-approx k-median) *Given a set P of n points in $\mathbb{R}^d$, and parameter k, one can compute a $(1+\varepsilon)$-approximate k-median clustering of P (in the continuous sense) in $O\left(n + k^5\log^9 n + \varrho k^2\log^5 n\right)$ time, where $\varrho = \exp\left[O\left((1+\log 1/\varepsilon)/\varepsilon\right)^{d-1}\right]$ and c is a constant. The algorithm outputs a set X of k points, such that $\nu_X(P) \le (1+\varepsilon)\nu_{\mathrm{opt}}(P,k)$. If P is a weighted set, with total weight W, the running time of the algorithm is $O(n\log^2 W + k^5\log^9 W + \varrho k^2\log^5 W)$.*

We can extend our techniques to handle the discrete median case efficiently as follows.

Lemma 5.6 *Given a set P of n points in $\mathbb{R}^d$, one can compute a discrete (k,ε)-approximate weber-centroid set $\mathcal{D} \subseteq P$ of size $O(k^2\varepsilon^{-2d}\log^2 n)$. The running time of this algorithm is $O\left(n + k^5\log^9 n + k^2\varepsilon^{-2d}\log^2 n\right)$ if $k \le \varepsilon^d n^{1/4}$ and $O\left(n\log n + k^5\log^9 n + k^2\varepsilon^{-2d}\log^2 n\right)$ otherwise.*

PROOF. Omitted. Will appear in the full-version. □

Combining Lemma 5.6 and Theorem 5.5, we get the following.

Theorem 5.7 (Discrete k-medians) *One can compute an $(1+\varepsilon)$-approximate discrete k-median of a set of n points in time $O\left(n + k^5\log^9 n + \varrho k^2\log^5 n\right)$, where ϱ is the constant from Theorem 5.4.*

PROOF. The proof follows from the above discussion. As for the running time bound, it follows by considering separately the case when $1/\varepsilon^{2d} \le 1/n^{1/10}$, and the case when $1/\varepsilon^{2d} \ge 1/n^{1/10}$, and simplifying the resulting expressions. We omit the easy but tedious computations. □

6. A $(1+\varepsilon)$-APPROXIMATION ALGORITHM FOR K-MEANS

We only state the result, the details are in Appendix C. Note that despite of the messy looking running time, our algorithm performs significantly better than Matoušek's algorithm [28], which runs in $O(n(\log n)^k\varepsilon^{-2k^2d})$ time.

Theorem 6.1 (k-means) *Given a n point set P in $\mathbb{R}^d$, one can compute $(1+\varepsilon)$-approximate k-means clustering of P in time $O(n+k^5\log^9 n+k^{k+2}\varepsilon^{-(2d+1)k}\log^{k+1} n\log^k(1/\varepsilon))$.*

For a weighted set with total weight W, the running time is $O(n\log^2 W+k^5\log^9 W+k^{k+2}\,\varepsilon^{-(2d+1)k}\,\log^{k+1} W\log^k(1/\varepsilon))$.

7. STREAMING

A consequence of our ability to compute quickly a (k,ε)-coreset for a point set, is that we can maintain the coreset under insertions quickly.

Observation 7.1 *(i) If C_1 and C_2 are the (k,ε)-coresets for disjoint sets P_1 and P_2 respectively, then $C_1 \cup C_2$ is a (k,ε)-coreset for $P_1 \cup P_2$.*

(ii) If C_1 is (k,ε)-coreset for C_2, and C_2 is a (k,δ)-coreset for C_3, then C_1 is a $(k,\varepsilon+\delta)$-coreset for C_3.

The above observation allows us to use Bentley and Saxe's technique [10] as follows. Let $P = (p_1, p_2, \ldots, p_n)$ be the sequence of points seen so far. We partition P into sets $P_0, P_1, P_2, \ldots, P_t$ such that each either P_i empty or $|P_i| = 2^i M$, for $i > 0$ and $M = O(k/\varepsilon^d)$. We refer to i as the rank of i.

Define $\rho_j = \varepsilon/\left(c(j+1)^2\right)$ where c is a large enough constant, and $1+\delta_j = \prod_{l=0}^{j}(1+\rho_l)$, for $j = 1,\ldots,\lceil \lg n\rceil$. We store a (k,δ_j)-coreset Q_j for each P_j. It is easy to verify that $1+\delta_j \le 1+\varepsilon/2$ for $j = 1,\ldots,\lceil\lg n\rceil$ and sufficiently large c. Thus the union of the Q_is is a $(k,\varepsilon/2)$-coreset for P.

On encountering a new point p_u, the update is done in the following way: We add p_u to P_0. If P_0 has less than M elements, then we are done. Note that for P_0 its corresponding coreset Q_0 is just itself. Otherwise, we set $Q_1' = P_0$, and we empty Q_0. If Q_1 is present, we compute a (k,ρ_2) coreset to $Q_1 \cup Q_1'$ and call it Q_2', and remove the sets Q_1 and Q_1'. We continue the process until we reach a stage r where Q_r did not exist. We set Q_r' to be Q_r. Namely, we repeatedly merge sets of the same rank, reduce their size using the coreset computation, and promote the resulting set to the next rank. The construction ensures that Q_r is a (k,δ_r) coreset for a corresponding subset of P of size $2^r M$. It is now easy to verify, that Q_r is a $(k, \prod_{l=0}^{j}(1+\rho_l)-1)$-coreset for the corresponding points of P.

We modify the above construction, by computing a $(k,\varepsilon/6)$-coreset R_i for Q_i, whenever we compute Q_i. The time to do this is dominated by the time to compute Q_i. Clearly, $\cup R_i$ is a (k,ε)-coreset for P at any point in time, and $|\cup R_i| = O(k\varepsilon^{-d}\log^2 n)$.

Streaming k-means. In this case, the Q_is are coresets for k-means clustering. Since Q_i has a total weight equal to $2^i M$ (if it is not empty) and is generated as a $(1+\rho_i)$ approximation, by Theorem C.2, $|Q_i| = O\left(k\varepsilon^{-d}(i+1)^{2d}(i+\log M)\right)$. Thus the total storage requirement is $O\left(\left(k\log^{2d+2} n\right)/\varepsilon^d\right)$.

Specifically, a (k,ρ_j) approximation of a subset P_j of rank j is constructed after every $2^j M$ insertions, therefore using Theorem C.2 the amortized time spent for an update is

$$\begin{aligned}
&\sum_{i=0}^{\lceil\log(n/M)\rceil} \frac{1}{2^i M} O\left(|Q_i|\log^2|P_i| + k^5\log^9|P_i|\right)\\
&= \sum_{i=0}^{\lceil\log\frac{n}{M}\rceil} \frac{1}{2^i M} O\left(\frac{k}{\varepsilon^d} i^{2d}(i+\log M)^2 + k^5(i+\log M)^9\right)\\
&= O\left(\log^2(k/\varepsilon) + k^5\right).
\end{aligned}$$

Further, we can generate an approximate k-means clustering from the (k,ε)-coresets, by using the algorithm of Theorem 6.1 on $\cup_i R_i$, with $W = n$. The resulting running time is $O(k^5\log^9 n + k^{k+2}\varepsilon^{-(2d+1)k}\log^{k+1} n\log^k(1/\varepsilon))$.

Streaming k-medians. We use the algorithm of Lemma 5.1 for the coreset construction. Further we use Theorem 5.5 to compute an $(1+\varepsilon)$-approximation to the k-median from the current coreset. The above discussion can be summarized as follows.

Theorem 7.2 *Given a stream P of n points in $\mathbb{R}^d$ and $\varepsilon > 0$, one can maintain a (k,ε)-coresets for k-median and k-*

means efficiently and use the coresets to compute a $(1+\varepsilon)$-approximate k-means/median for the stream seen so far. The relevant complexities are:

- *Space to store the information:* $O\left(k\varepsilon^{-d}\log^{2d+2} n\right)$.
- *Size and time to extract coreset of the current set:* $O(k\varepsilon^{-d}\log^2 n)$.
- *Amortized update time:* $O\left(\log^2(k/\varepsilon)+k^5\right)$.
- *Time to extract $(1+\varepsilon)$-approximate k-means clustering:* $O\left(k^5\log^9 n + k^{k+2}\varepsilon^{-(2d+1)k}\log^{k+1} n\log^k(1/\varepsilon)\right)$.
- *Time to extract $(1+\varepsilon)$-approximate k-median clustering:* $O\left(\varrho k\log^7 n\right)$, *where* $\varrho = \exp\left[O\left((1+\log 1/\varepsilon)/\varepsilon\right)^{d-1}\right]$.

Interestingly, once an optimization problem has a coreset, the coreset can be maintained under both insertions and deletions, using linear space. The following result follows in a plug and play fashion from [1, Theorem 5.1], and we omit the details.

Theorem 7.3 *Given a point set P in $\mathbb{R}^d$, one can maintain a (k,ε)-coreset of P for k-median/means, using linear space, and in time $O(k\varepsilon^{-d}\log^{d+2} n\log\frac{k\log n}{\varepsilon} + k^5\log^{10} n)$ per insertion/deletions.*

8. CONCLUSIONS

In this paper, we showed the existence of small coresets for the k-means and k-median clustering. At this point, there are numerous problems for further research. In particular:

1. Can the running time of approximate k-means clustering be improved to be similar to the k-median bounds? Can one do FPTAS for k-median and k-means (in both k and $1/\varepsilon$)? Currently, we can only compute the (k,ε)-coreset in fully polynomial time, but not extracting the approximation itself from it.
2. Can the $\log n$ in the bound on the size of the coreset be removed?
3. Does a coreset exist for the problem of k-median and k-means in high dimensions? There are some partial relevant results [9].
4. Can one do efficiently $(1+\varepsilon)$-approximate streaming for the discrete k-median case?
5. Recently, Piotr Indyk [23] showed how to maintain a $(1+\varepsilon)$-approximation to k-median under insertion and deletions (the number of centers he is using is roughly $O(k\log^2\Delta)$ where Δ is the spread of the point set). It would be interesting to see if one can extend our techniques to maintain coresets also under deletions. It is clear that there is a linear lower bound on the amount of space needed, if one assume nothing. As such, it would be interesting to figure out what are the minimal assumptions for which one can maintain (k,ε)-coreset under insertions and deletions.

Acknowledgments

The authors would like to thank Piotr Indyk and Satish Rao for useful discussions of problems studied in this paper and related problems.

9. REFERENCES

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 2003. to appear.

[2] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for k-line center. In *Proc. 10th Annu. European Sympos. Algorithms*, pages 54–63, 2002.

[3] A. Amir, A. Efrat, P. Indyk, and H. Samet. Efficient algorithms and regular data structures for dilation, location and proximity problems. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 160–170, 1999.

[4] S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, Sep 1998.

[5] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-median and related problems. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 106–113, 1998.

[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6), 1998.

[7] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *Proc. 33rd Annu. ACM Sympos. Theory Comput.*, pages 21–29, 2001.

[8] M. Bădoiu and K. L. Clarkson. Optimal core-sets for balls. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*, pages 801–802, 2003.

[9] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 250–257, 2002.

[10] J. L. Bentley and J. B. Saxe. Decomposable searching problems i: Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[11] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 378–388, 1999.

[12] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, pages 1–10, 1999.

[13] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. Approximation schemes for clustering problems. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 50–58, 2003.

[14] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[15] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic J. Comput.*, 2:3–27, 1995.

[16] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[17] S. Guha, R. M. N. Mishra, and L. O'Callaghan. Clustering data streams. In *Proc. 41th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 359–366, 2000.

[18] S. Har-Peled. Clustering motion. In *Proc. 42nd Annu.*

IEEE Sympos. Found. Comput. Sci., pages 84–93, 2001.

[19] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

[20] S. Har-Peled and K. R. Varadarajan. Projective clustering in high dimensions using core-sets. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 312–318, 2002.

[21] 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.

[22] P. Indyk. Sublinear time algorithms for metric space problems. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, pages 154–159, 1999.

[23] P. Indyk. $(1+\varepsilon)$-approximate bi-criterion algorithm for k-median on a stream. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, 2004.

[24] K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–13, 1999.

[25] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 10–18, 2002.

[26] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the euclidean κ-median problem. In *Proc. 7th Annu. European Sympos. Algorithms*, pages 378–389, 1999.

[27] L. O. M. Charikar and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 30–39, 2003.

[28] J. Matoušek. On approximate geometric k-clustering. *Discrete Comput. Geom.*, 24:61–84, 2000.

[29] R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. In *In Proceedings of the 18th Conference on Uncertainty in Artifical Intelligence*, pages 344–351, 2002.

APPENDIX

A. WEBER-CENTROID SET CONSTRUCTION

Proof of Lemma 5.3 Let C_{opt} be the optimal set of k medians. We snap each point of C_{opt} to its nearest neighbor in $\mathcal{D}$, and let X be the resulting set. Arguing as in the proof of Theorem 4.3, we have that $\left|\nu_X(\mathcal{S}) - \nu_{C_{\text{opt}}}(\mathcal{S})\right| \leq (\varepsilon/12)\nu_{C_{\text{opt}}}(\mathcal{S})$. On the other hand, from the definition of a coreset, we have that $\left|\nu_{C_{\text{opt}}}(P) - \nu_{C_{\text{opt}}}(\mathcal{S})\right| \leq (\varepsilon/12)\ \nu_{C_{\text{opt}}}(P)$ and $|\nu_X(P) - \nu_X(\mathcal{S})| \leq (\varepsilon/12)\ \nu_X(P)$. As such, $\nu_{C_{\text{opt}}}(\mathcal{S}) \leq (1+\varepsilon/12)\nu_{C_{\text{opt}}}(P)$ and it follows

$$\begin{aligned}\left|\nu_X(\mathcal{S}) - \nu_{C_{\text{opt}}}(\mathcal{S})\right| &\leq (\varepsilon/12)(1+\varepsilon/12)\nu_{C_{\text{opt}}}(P)\\ &\leq (\varepsilon/6)\nu_{C_{\text{opt}}}(P).\end{aligned}$$

As such,

$$\begin{aligned}\nu_X(P) &\leq \frac{1}{1-\varepsilon/12}\nu_X(\mathcal{S}) \leq 2\nu_X(\mathcal{S})\\ &\leq 2\left(\nu_{C_{\text{opt}}}(\mathcal{S}) + \frac{\varepsilon}{6}\nu_{C_{\text{opt}}}(P)\right)\\ &\leq 2\left(\left(1+\frac{\varepsilon}{12}\right)\nu_{C_{\text{opt}}}(P) + \frac{\varepsilon}{6}\nu_{C_{\text{opt}}}(P)\right)\\ &\leq 3\nu_{C_{\text{opt}}}(P),\end{aligned}$$

for $\varepsilon < 1$. Thus we have $|\nu_X(P) - \nu_X(\mathcal{S})| \leq (\varepsilon/12)\nu_X(P) \leq (\varepsilon/3)\nu_{C_{\text{opt}}}(P)$. Putting things together, we have

$$\begin{aligned}\left|\nu_X(P) - \nu_{C_{\text{opt}}}(P)\right| &\leq |\nu_X(P) - \nu_X(\mathcal{S})|\\ &\quad + \left|\nu_X(\mathcal{S}) - \nu_{C_{\text{opt}}}(\mathcal{S})\right|\\ &\quad + \left|\nu_{C_{\text{opt}}}(\mathcal{S}) - \nu_{C_{\text{opt}}}(P)\right|\\ &\leq \left(\frac{\varepsilon}{3} + \frac{\varepsilon}{6} + \frac{\varepsilon}{12}\right)\nu_{C_{\text{opt}}}(P)\\ &\leq \varepsilon\nu_{C_{\text{opt}}}(P).\end{aligned}$$

■

B. FUZZY NEAREST-NEIGHBOR SEARCH IN CONSTANT TIME

Let X be a set of m points in $\mathbb{R}^d$, such that we want to answer ε-approximate nearest neighbor queries on X. However, if the distance of the query point q to its nearest neighbor in X is smaller than δ, then it is legal to return any point of X in distance smaller than δ from q. Similarly, if a point is in distance larger than Δ from any point of X, we can return any point of X. Namely, we want to do nearest neighbor search on X, when we care only for an accurate answer if the distance is in the range $[\delta, \Delta]$.

Definition B.1 Given a point set X and parameters δ, Δ and ε, a data structure D answers $(\delta, \Delta, \varepsilon)$-*fuzzy nearest neighbor* queries, if for an arbitrary query q, it returns a point $x \in X$ such that

1. If $\mathbf{d}(q, X) > \Delta$ then x is an arbitrary point of X.
2. If $\mathbf{d}(q, X) < \delta$ then x is an arbitrary point of X in distance smaller than δ from q.
3. Otherwise, $\|qx\| \leq (1+\varepsilon)\mathbf{d}(q, X)$.

In the following, let $\rho = \Delta/\delta$ and assume that $1/\varepsilon = O(\rho)$. First, we construct a grid G_Δ of size length Δ, using hashing and the floor function, we throw the points of X into their relevant cells in G_Δ. We construct a NN data structure for every non-empty cell in G_Δ. Given a query point q, we will compute its cell c in the grid G_Δ, and perform NN queries in the data-structure associated with c, and the data-structures associated with all its neighboring cells, returning the best candidate generated. This would imply $O(3^d)$ queries into the cell-level NN data-structure.

Consider Y to be the points of X stored in a cell c of G_Δ. We first filter Y so that there are no points in Y that are too close to each other. Namely, let G be the grid of side length $\delta\varepsilon/(10d)$. Again, map the points of Y into this grid G, in linear time. Next, scan over the nonempty cells of G, pick a representative point of Y from such a cell, and add it to the output point set Z. However, we do not add a representative point x to Z, if there is a neighboring cell to c_x, which already has a representative point in Z, where c_x is the cell in G containing x. Clearly, the resulting set

$Z \subseteq Y$ is well spaced, in the sense that there is no pair of points of Z that are in distance smaller than $\delta\varepsilon/(10d)$ from each other. As such, the result of a $(\delta, \Delta, \varepsilon)$-fuzzy NN query on Z is a valid answer for a equivalent fuzzy NN query done on Y, as can be easily verified. This filtering process can be implemented in linear time.

The point set Z has a bounded stretch; namely, the ratio between the diameter of Z and the distance of the closet pair is bounded by $\Delta/(\delta\varepsilon/(10d)) = O(\rho^2)$. As such, we can use a data structure on Z for nearest neighbors on point set with bounded stretch [19, Section 4.1]. This results in a quadtree T of depth $O(\log(\rho)) \leq c \log \rho$, where c is constant. Answering NN queries, is now done by doing a point-location query in T, and finding the leaf of T that contains the query point q, as every leaf v in T store a point of Z which is a $(1+\varepsilon)$-approximate nearest neighbor for all the points in c_v, where c_v is the region associated with v. The construction time of T is $O(|Z|\, \varepsilon^{-d} \log \rho)$, and this also bound the size of T.

Doing the point-location query in T in the naive way, takes $O(\text{depth}(T)) = O(\log \rho)$ time. However, there is a standard technique to speed up the nearest neighbor query in this case to $O(\log \text{depth}(T))$ [3]. Indeed, observe that one can compute for every node in T a unique label, and furthermore given a query point $q = (x, y)$ (we use a 2d example to simplify the exposition) and a depth i, we can compute in constant time the label of the node of the quadtree T of depth i that the point-location query for q would go through. To see that, consider the quadtree as being constructed on the unit square $[0, 1]^2$, and observe that if we take the first i bits in the binary representation of x and y, denoted by x_i and y_i respectively, then the tuple (x_i, y_i, i) uniquely define the required node, and the tuple can be computed in constant time using bit manipulation operators.

As such, we hash all the nodes in T with their unique tuple id into a hash table. Given a query point q, we can now perform a binary search along the path of q in T, to find the node where this path "falls of" T. This takes $O(\log \text{depth}(T))$ time.

One can do even better. Indeed, we remind the reader that the depth of T is $c \log \rho$, where c is a constant. Let $\alpha = \lceil (\log \rho)/(20dr) \rceil \leq (\log \rho)/(10dr)$, where r is an arbitrary integer parameter. If a leaf v in T is of depth u, we continue to split and refine it till all the resulting leaves of v lie in level $\alpha \lceil u/\alpha \rceil$ in T. This would blow up the size of the quadtree by a factor of $O((2^d)^\alpha) = O(\rho^{1/r})$. Furthermore, by the end of this process, the resulting quadtree has leaves only on levels with depth which is an integer multiple of α. In particular, there are only $O(r)$ levels in the resulting quadtree T' which contain leaves.

As such, one can apply the same hashing technique described above to T', but only for the levels that contains leaves. Now, since we do a binary search over $O(r)$ possibilities, and every probe into the hash table takes constant time, it follows that a NN query takes $O(\log r)$ time.

We summarize the result in the following theorem.

Theorem B.2 *Given a point set X with m points, and parameters δ, Δ and $\varepsilon > 0$, then one can preprocess X in $O(m\rho^{1/r}\varepsilon^{-d} \log(\rho/\varepsilon))$ time, so that one can answer $(\delta, \Delta, \varepsilon)$-fuzzy nearest neighbor queries on X in $O(\log r)$ time. Here $\rho = \Delta/\delta$ and r is an arbitrary integer number fixed in advance.*

Theorem B.3 *Given a point set X of size m, and a point set P of size n both in $\mathbb{R}^d$, one can compute in $O(n + mn^{1/4}\varepsilon^{-d} \log(n/\varepsilon))$ time, for every point $p \in P$, a point $x_p \in X$, such that $\|px_p\| \leq (1+\varepsilon)\mathbf{d}(p, X) + \tau/n^3$, where $\tau = \max_{p \in P} \mathbf{d}(p, X)$.*

PROOF. The idea is to quickly estimate τ, and then use Theorem B.2. To estimate τ, we use a similar algorithm to the closet-pair algorithm of Golin *et al.* [15]. Indeed, randomly permute the points of P, let $p_1, \ldots, p_n$ be the points in permuted order, and let l_i be the current estimate of r_i, where $r_i = \max_{j=1}^{i} \mathbf{d}(p_i, X)$ is the maximum distance between $p_1, \ldots, p_i$ and X. Let G_i be a grid of side length l_i, where all the cells contains points of X, or their neighbors are marked. For p_{i+1} we check if it contained inside one of the marked cells. If so, we do not update the current estimate, and set $l_{i+1} = l_i$ and $G_{i+1} = G_i$. Otherwise, we scan the points of X, and we set $l_{i+1} = 2\sqrt{d}\mathbf{d}(p_{i+1}, X)$, and we recompute the grid G_{i+1}. It is easy to verify that $r_{i+1} \leq l_{i+1}$ in such a case, and $r_{i+1} \leq 2\sqrt{d} l_{i+1}$ if we do not rebuild the grid.

Thus, by the end of this process, we get l_n, for which $l_n/(2\sqrt{d}) \leq \tau \leq 2\sqrt{d} l_n$, as required. As for the expected running time, note that if we rebuild the grid and compute $\mathbf{d}(p_{i+1}, X)$ explicitly, this takes $O(k)$ time. Clearly, if we rebuild the grid at stage i, and the next time at stage $j > i$, it must be that $r_i \leq l_i < r_j \leq l_j$. However, in expectation, the number of different values in the series $r_1, r_2, \ldots, r_n$ is $\sum_{i=1}^{n} 1/i = O(\log n)$. Thus, the expected running time of this algorithm is $O(n + k \log n)$, as checking whether a point is in a marked cell, takes $O(1)$ time by using hashing.

We know that $l_n/(2\sqrt{d}) \leq \tau \leq 2\sqrt{d} l_n$. Set $\delta = l_n/(4d^2n^5)$, $\Delta = 2\sqrt{d} l_n$ and build the $(\delta, \Delta, \varepsilon)$-fuzzy nearest neighbor data-structure of Theorem B.2 for X. We can now answer the nearest neighbor queries for the points of P in $O(1)$ per query. □

Corollary B.4 *Given a point set X of size m, a point set P of size n both in $\mathbb{R}^d$, and a parameter D, one can compute in $O(n + mn^{1/10}\varepsilon^{-d} \log(n/\varepsilon))$ time, for every point $p \in P$ a a point $x_p \in P$, such that:*

- *If $\mathbf{d}(p, X) > D$ then x_p is an arbitrary point in X.*
- *If $\mathbf{d}(p, X) \leq D$ then $\|px_p\| \leq (1+\varepsilon)\mathbf{d}(p, X) + D/n^4$.*

C. A $(1+\varepsilon)$-APPROXIMATION ALGORITHM FOR K-MEANS

C.1 Constant Factor Approximation

In this section we reduce the number of centers to be exactly k. We use the set of centers computed by Theorem 3.3 to compute a constant factor coreset using the algorithm of Theorem 4.4. The resulting coreset $\mathcal{S}$, has size $O(k \log^4 n)$. Next we compute a $O(n)$ approximation to the k-means for the coreset using the k-center (min-max) algorithm [16]. Let $C_0 \subseteq \mathcal{S}$ be the resulting set of centers. Next we apply the local search algorithm, due to Kanungo *et al.* [25], to C_0 and $\mathcal{S}$, where the set of candidate points is $\mathcal{S}$. This local search algorithm, at every stage, picks a center c from the current set of centers C_{curr}, and a candidate center $s \in \mathcal{S}$, and swaps c out of the set of centers and c into the set of centers. Next, if the new set of centers $C'_{curr} = C_{curr} \setminus \{c\} \cup \{s\}$

provides a considerable improvement over the previous solution (i.e., $\mu_{C_{curr}}(\mathcal{S}) \leq (1-\varepsilon/k)\mu_{C'_{curr}}(\mathcal{S})$ where ε here is an arbitrary small constant), then we set C_{curr} to be C'_{curr}. Extending the analysis of Arya *et al.* [7], for the k-means algorithm, Kanungo *et al.* [25] showed that the algorithm terminates, and it provides a constant factor approximation to $\mu^D_{\text{opt}}(\mathcal{S}, k)$, and as hence to $\mu_{\text{opt}}(P, k)$. It is easy to verify that it stops after $O(k \log n)$ such swaps. Every swap, in the worst case, requires considering $|\mathcal{S}|\, k$ sets. Computing the cost of clustering for every such candidate set of centers takes $O(|\mathcal{S}|\, k)$ time. Thus, the running time of this algorithm is $O\left(|\mathcal{S}|^2 k^3 \log n\right) = O\left(k^5 \log^9 n\right)$.

Theorem C.1 *Given a point set P in $\mathbb{R}^d$ and parameter k, one can compute a set $X \subseteq P$ of size k, such that $\mu_X(P) = O(\mu_{\text{opt}}(P, k))$. The algorithm succeeds with high probability. The running time is $O(n + k^5 \log^9 n)$ time.*

If P is weighted, with total weight W, then the algorithm runs in time $O(n + k^5 \log^4 n \log^5 W)$.

C.2 The $(1+\varepsilon)$-Approximation

Combining Theorem C.1 and Theorem 4.4, we get the following result for coresets.

Theorem C.2 (coreset) *Given a set P of n points in $\mathbb{R}^d$, one can compute a k-means (k, ε)-coreset $\mathcal{S}$ of P, of size $O\left((k/\varepsilon^d) \log n\right)$, in time $O\left(n + k^5 \log^9 n\right)$.*

If P is weighted, with total weight W, then the coreset is of size $O\left((k/\varepsilon^d) \log W\right)$, and the running time is $O(n \log^2 W + k^5 \log^9 W)$.

Proof. We first compute a set A which provides a constant factor approximation to the optimal k-means clustering of P, using Theorem C.1. Next, we feed A into the algorithm Theorem 4.4, and get a $(1+\varepsilon)$-coreset for P, of size $O((k/\varepsilon^d) \log W)$. □

We now use techniques from Matoušek [28] to compute the $(1+\varepsilon)$-approximate k-means clustering on the coreset.

Definition C.3 (Centroid Set) Given a set P of n points in $\mathbb{R}^d$, a set $T \subseteq \mathbb{R}^d$ is an *ε-approximate centroid set* for P, if there exists a subset $C \subseteq T$ of size k, such that $\mu_C(P) \leq (1+\varepsilon)\mu_{\text{opt}}(P, k)$.

Matoušek showed that there exists an ε-approximate centroid set of size $O(n\varepsilon^{-d} \log(1/\varepsilon))$. Interestingly enough, his construction is weight insensitive. In particular, using an $(k, \varepsilon/2)$-coreset $\mathcal{S}$ in his construction, results in a ε-approximate centroid set of size $O\left(|\mathcal{S}|\, \varepsilon^{-d} \log(1/\varepsilon)\right)$.

Lemma C.4 *For a weighted point set P in $\mathbb{R}^d$, with total weight W, there exists an ε-approximate centroid set of size $O(k\varepsilon^{-2d} \log W \log(1/\varepsilon))$.*

The algorithm to compute the $(1+\varepsilon)$-approximation now follows naturally. We first compute a coreset $\mathcal{S}$ of P of size $O\left((k/\varepsilon^d) \log W\right)$ using the algorithm of Theorem C.2. Next, we compute in $O\left(|\mathcal{S}| \log |\mathcal{S}| + |\mathcal{S}|\, e^{-d} \log \frac{1}{\varepsilon}\right)$ time a ε-approximate centroid set U for $\mathcal{S}$, using the algorithm from [28]. We have $|U| = O(k\varepsilon^{-2d} \log W \log(1/\varepsilon))$. Next we enumerate all k-tuples in U, and compute the k-means clustering cost of each candidate center set (using $\mathcal{S}$). This takes $O\left(|U|^k \cdot k\, |\mathcal{S}|\right)$ time. And clearly, the best tuple provides the required approximation.

Theorem C.5 (k-means clustering) *Given a point set P in $\mathbb{R}^d$ with n points, one can compute $(1+\varepsilon)$-approximate k-means clustering of P in time*

$$O\left(n + k^5 \log^9 n + k^{k+2} \varepsilon^{-(2d+1)k} \log^{k+1} n \log^k(1/\varepsilon)\right).$$

For a weighted set, with total weight W, the running time is

$$O\ (\ n \log^2 W + k^5 \log^4 n \log^5 W + k^{k+2} \varepsilon^{-(2d+1)k} \log^{k+1} W \log^k(1/\varepsilon)\ ).$$