Clustering Problems on Sliding Windows

Vladimir Braverman[*] Harry Lang[†] Keith Levin[‡] Morteza Monemizadeh[§]

July 8, 2015

Abstract

We explore clustering problems in the streaming sliding window model in both general metric spaces and Euclidean space. We present the first polylogarithmic space $O(1)$-approximation to the metric k-median and metric k-means problems in the sliding window model, answering the main open problem posed by Babcock, Datar, Motwani and O'Callaghan [5], which has remained unanswered for over a decade. Our algorithm uses $O(k^3 \log^6 n)$ space and $\text{poly}(k, \log n)$ update time. This is an exponential improvement on the space required by the technique due to Babcock, et al. We introduce a data structure that extends smooth histograms as introduced by Braverman and Ostrovsky [8] to operate on a broader class of functions. In particular, we show that using only polylogarithmic space we can maintain a summary of the current window from which we can construct an $O(1)$-approximate clustering solution.

Merge-and-reduce is a generic method in computational geometry for adapting offline algorithms to the insertion-only streaming model. Several well-known coreset constructions are maintainable in the insertion-only streaming model using this method, including well-known coreset techniques for the k-median and k-means problems in both low-and high-dimensional Euclidean spaces [29, 13]. Previous work [25] has adapted coreset techniques to the insertion-deletion model, but translating them to the sliding window model has remained a challenge. We give the first algorithm that, given an insertion-only streaming coreset of space s (maintained using merge-and-reduce method), maintains this coreset in the sliding window model using $O(s^2 \epsilon^{-2} \log n)$ space.

For clustering problems, our results constitute the first significant step towards resolving problem number 20 from the List of Open Problems in Sublinear Algorithms [37].

[*]Department of Computer Science, Johns Hopkins University. This material is based upon work supported in part by the National Science Foundation under Grant No. 1447639, by the Google Faculty Award and by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense. Email: vova@cs.jhu.edu

[†]Department of Mathematics, Johns Hopkins University. Email: hlang8@jhu.edu.

[‡]Department of Computer Science, Johns Hopkins University. Email: klevin@jhu.edu.

[§]Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic. Partially supported by the project 14-10003S of GA ČR. Part of this work was done when the author was at Department of Computer Science, Goethe-Universität Frankfurt, Germany and supported in part by MO 2200/1-1. Email: monemi@iuuk.mff.cuni.cz.

1 Introduction

Over the past two decades, the streaming model of computation [35] has emerged as a popular framework in which to develop algorithms for large data sets. In the streaming model, we are restricted to using space sublinear in the size of the input, and this input typically must be processed in a single pass. While the streaming model is broadly useful, it is inadequate for domains in which data is time-sensitive such as network monitoring [14, 15, 17] and event detection in social media [36]. In these domains, elements of the stream appearing more recently are in some sense more relevant to the computation being performed. The sliding window model was developed to capture this situation [20]. In this model, the goal is to maintain a computation on only the most recent W elements of the stream, rather than on the stream in its entirety.

We consider the problem of clustering in the sliding window model. Algorithms have been developed for a number of streaming clustering problems, including k-median [26, 12, 29, 25], k-means [13, 23] and facility location [19]. However, while the sliding window model has received renewed attention recently [18, 6], no major clustering results in this model have been published since Babcock, Datar, Motwani and O'Callaghan [5] presented a solution to the k-median problem. Polylogarithmic space k-median algorithms exist in the insertion-only streaming model [12, 29] and the insertion-deletion model [30, 25, 31], but no analogous result has appeared to date for the sliding window model. Indeed, the following question by Babcock, et al. [5] has remained open for more than a decade:

> *Whether it is possible to maintain approximately optimal medians in polylogarithmic space (as Charikar et al. [12] do in the stream model without sliding windows), rather than polynomial space, is an open problem.*

Much progress on streaming clustering problems in Euclidean space has been due to coresets[29, 13, 25, 21, 24]. But, similarly to the metric case, methods for maintaining coresets on sliding windows for Euclidean clustering problems have been hard to come by. Streaming insertion-only coreset techniques exist for the Euclidean k-median and k-means problems in low-and high-dimensional spaces, but to our knowledge no such results exist for the sliding window model. We present the first such technique, a general framework in which one can build coresets for a broad class of clustering problems in Euclidean space.

1.1 Our Contribution

Metric clustering problems in the sliding window model. We present the first polylogarithmic space $O(1)$-approximation for the metric k-median problem in the sliding window model, answering the question posed by Babcock et al. [5]. Our algorithm uses $O(k^3 \log^6 n)$-space and requires update time $O(\text{poly}(k, \log n))$, with the exact update time depending on which of the many existing offline $O(1)$-approximations for k-median one chooses. We also demonstrate how this result extends to a broad class of related clustering problems, including k-means. The one requirement of our result is a polynomial bound on the ratio of the optimal cost to the minimum inter-point distance on any window of the stream.

Braverman and Ostrovsky [11] introduced smooth histograms as a method for adapting insertion-only streaming algorithms to the sliding window model for a certain class of functions, which Braverman and Ostrovsky call *smooth* (see Definition 4). Unfortunately, the k-median and k-means costs are not smooth functions (for a simple counterexample see Lemma 7). Our main technical contribution is constructing a histogram data structure for the non-smooth function of k-median. We do this by first maintaining an $O(1)$-approximate insertion-only instances at a set of indices such that the cost of an optimal solution does not vary too much between indices. For each one of these instances, we additionally maintain $O(k \log n)$ instances such that the size of each of the k clusters do not vary too much between any two subindices. Altogether, we maintain $O(\log n)$ indices with bounded variation in cost, and for each index we maintain $O(k \log n)$ subindices with bounded variation in cluster size. However, the difficulty arises when we need to delete an index (to maintain polylogarithmic space), where in order to maintain the index/subindex bounds we must store a set of facilities for the subset of the window in between each pair of indices. To overcome this problem, we prove that the set of facilities between the two indices around the deleted index can be used to determine the relation of their approximate cluster sizes. Therefore we are able to ensure an $O(1)$-approximate solution over the window while incurring a polylogarithmic multiplicative-factor in space.

Euclidean clustering problems in the sliding window model. Merge-and-reduce is a generic method in computational geometry to implement offline algorithms in the insertion-only streaming model. Several well-known coreset constructions are conducive to this method, including well-known coreset techniques for the k-median and k-means problems in both low-and high-dimensional Euclidean spaces [29, 13]. We develop a sliding window algorithm that, given one of these insertion-only streaming coresets of size s, maintains this coreset in the sliding window model using $O(s^2\epsilon^{-2}\log n)$ space.

To develop our generic framework, we consider a sequence X of indices of arrival times of points. For each index $x_i \in X$ we maintain a coreset (using merge-and-reduce) of points whose arrival times are between x_i and current time N. In particular, as the points in the interval $[x_i, N]$ arrive, we compute coresets for small subsets of points. Once the number of these coresets is big enough, we *merge* these coresets and *reduce* them by computing a coreset on their union. This yields a tree whose root is a coreset of its leaves, which contain subsets of the points in the interval $[x_i, N]$. The well-known coreset techniques of [29, 13, 21, 24] mostly partition the space into small set of regions and from each region take a small number of points either randomly or deterministically. Hence, at the root of the merge-and-reduce tree, we have a partition from whose regions we take weighted coreset points.

We would like to use the smooth histograms [8] data structure in order to maintain a small set of indices X such that for every two consecutive indices x_i and x_{i+1} all intervals $[x_i + 1, N], \ldots, [x_{i+1} - 1, N]$ have clustering cost which are within $(1 + \epsilon)$-fraction of each other. In other words, by keeping only indices x_i and x_{i+1}, we would like to smooth the cost between these two indices. To this end, we look at the partition of the root of the merge-and-reduce tree corresponding to an arrival time $t \in [x_i, x_{i+1}]$. If there is no region in this partition whose at least ϵ-fraction of its weight (, where the weight of a region is the sum of the weights of the coreset points in that region) is in interval $[x_i, x_{i+1}]$, then we ignore the interval $[t, N]$; otherwise we add index t to X and keep the coreset of weighted points for the interval $[t, N]$. We show using a novel application of VC-dimension and ϵ-sample theory [3] that if we take small random samples from every region of the partitions in the intermediate nodes of the merge-and-reduce tree, then the weighted coreset points inside every region of the partition of the root of this tree is a good approximation of the original points in that region. Thus, testing whether ϵ-fraction of overall weight of coreset points of a region is in the interval $[x_i, x_{i+1}]$ is a good approximation for testing whether ϵ-fraction of the original points of the region are in this interval. We also show that if for every region in the partition at most ϵ-fraction of overall weight of coreset points of that region is in the interval $[x_i, x_{i+1}]$, then ignoring the interval $[t, N]$ and keeping only intervals $[x_i, N]$ and $[x_{i+1}, N]$ loses at most ϵ-fraction of the clustering cost of the points in interval $[t, N]$.

Frahling and Sohler [25] developed a coreset technique for k-median and k-means problems in low-dimensional spaces in the dynamic geometric stream (i.e., a stream of insertions and deletions of points) using the heavy hitters algorithm [16] and a logarithmic sampling rate. We observe that we can maintain their coreset in the sliding window model using the heavy hitter algorithm and sampling techniques proposed for the sliding window model due to Braverman, Ostrovsky and Zaniolo [9]. However, their approach does not work for other well-known coreset techniques for Euclidean spaces [29, 13, 21, 24], motivating the need for a different technique, which we develop in this paper.

1.2 Related Work

Guha, Mishra, Motwani and O'Callaghan [26] presented the first insertion-only streaming algorithm for the k-median problem. They gave a $2^{O(1/\epsilon)}$-approximation using $O(n^\epsilon)$ space, where $\epsilon < 1$. Charikar, O'Callaghan, and Panigrahy [12], subsequently developed an $O(1)$-approximation insertion-only streaming algorithm using $O(k\log^2 n)$ space. Their approach operates in phases, similarly to [26], maintaining a set of $O(k \log n)$ candidate centers that are reduced to exactly k centers using an offline k-median algorithm after the entire stream has been observed.

Slightly stronger results hold when the elements of the stream are points in d-dimensional Euclidean space $\mathbb{R}^d$. Har-Peled and Mazumdar [29] developed a $(1+\epsilon)$-approximation for k-median and k-means in the insertion-only streaming model using (strong) coresets. Informally, a strong (k,ϵ)-coreset for k-median is a weighted subset S from some larger set of points P that enables us to (approximately) evaluate the quality of a candidate solution on P using small space. The coresets presented by Har-Peled and Mazumdar [29] required $O(k\epsilon^{-d}\log n)$ space, yielding a streaming solution using $O(k\epsilon^{-d}\log^{2d+2} n)$ space via the famous merge-and-reduce approach [7, 1]. Har-Peled and Kushal [28] later developed coresets of size $O(k^2\epsilon^{-d})$ for k-median and k-means problems. Feldman, Fiat and Sharir [21] later extended this type of coreset to the case where centers can be lines or flats.

In high-dimensional spaces, Chen [13] presented a technique for building (k,ϵ)-coresets of size $O(k^2d\epsilon^{-2}\log^2 n)$, yielding via merge-and-reduce a streaming algorithm requiring $O(k^2d\epsilon^{-2}\log^8 n)$ space. Chen [13] also presented a technique for general metric spaces, which, with probability of success $1-\delta$, produces a coreset of size $O(k\epsilon^{-2}\log n(k\log n+\log(1/\delta)))$.

To the best of our knowledge, there do not exist any results to date for the k-median problem on arbitrary metric spaces (often simply called the *metric k-median* problem) in the insertion-deletion streaming model. In the geometric case, introduced by Indyk [30] under the name *dynamic geometric data streams*, Frahling and Sohler [25] have shown (via a technique distinct from that in [29, 28]) that one can build a (k,ϵ)-coreset for k-median or k-means using $O(k^2\epsilon^{-2d-4}\log^7 n)$ or $O(k\epsilon^{-d-2}\log n)$ space, respectively.

In comparison to the insertion-only and dynamic geometric streaming models, little is known about the metric k-median problem in the sliding window model, where the goal is to maintain a solution on the most recent W elements of the data stream. To our knowledge, the only existing solution under this model is the $O(2^{O(\frac{1}{\tau})})$-approximation given in [5], where $\tau \in (0,1/2)$ is a user-specified parameter. The solution presented therein requires $O(\frac{k}{\tau^4}W^{2\tau}\log^2 W)$ space and yields an initial solution using $2k$ centers, which is then pared down to k centers with no loss to the approximation factor.

Outline. The remainder of the paper is organized as follows: Section 2 establishes notation and definitions. Our main results are presented in Section 3, which gives an algorithm for the k-median problem on sliding windows, and Section 4, which presents an algorithm for maintaining coresets for Euclidean clustering problems on sliding windows. Additional results and proof details are included in the Appendix.

2 Preliminaries

We begin by defining the clustering problems of interest and establishing notation.

2.1 Metric and Geometric k-Median Problems

Let (X,d) be a metric space where X is a set of points and $d : X\times X \to \mathbb{R}$ is a distance function defined over the points of X. For a set $Q \subseteq X$, we let $d(p,Q) = \min_{q\in Q} d(p,q)$ denote the distance between a point $p \in X$ and set Q and we denote by $\rho(Q) = \min_{p,q\in Q, p\neq q} d(p,q)$ the minimum distance between distinct points in Q. We define $[a] = \{1,2,3,\cdots a\}$ and $[a,b] = \{a, a+1, a+2, \cdots b\}$ for natural numbers $a \le b$. When there is no danger of confusion, we denote the set of points $\{p_a, p_{a+1}, \ldots, p_b\} \subset X$ by simply $[a,b]$. For example, for function f defined on sets of points, we denote $f(\{p_a, p_{a+1}, \ldots, p_b\})$ by simply $f([a,b])$.

Definition 1 (Metric k-median) *Let P be a set of n points in metric space (X,d) and let $C = \{c_1,\ldots,c_k\} \subseteq X$ be a set of k points called* centers. *A* clustering *of point set P using C is a partition of P such that a point $p \in P$ is in partition P_i if $c_i \in C$ is the nearest center in C to p, with ties broken arbitrarily. We call each P_i a* cluster. *The k-median cost using centers C is $\textit{COST}(P,C) = \sum_{p\in P} d(p,C)$. The metric k-median problem is to find a set $C^* \subset P$ of k centers satisfying $\textit{COST}(P,C^*) = \min_{C\subset P:|C|=k}\textit{COST}(P,C)$. We let $\textit{OPT}(P,k) = \min_{C\subset P:|C|=k}\textit{COST}(P,C)$ denote this optimal k-median cost for P.*

Definition 2 ((Euclidean) k-median Clustering) *Let P be a set of n points in a d-dimensional Euclidean Space $\mathbb{R}^d$ and k be a natural number. In the k-median problem, the goal is to find a set $C = \{c_1, \cdots, c_k\} \subset \mathbb{R}^d$ of k centers, that minimizes the cost $\textit{COST}(P, C) = \sum_{p \in P} d(p, C)$, where $d(p, C) = \min_{c_i \in C} d(p, c_i)$ is the Euclidean distance between p and c_i.*

Definition 3 ((k, ϵ)-Coreset for k-Median Clustering) *Let P be a set of n points in d-dimensional Euclidean Space $\mathbb{R}^d$ and let k be a natural number. A set $S \subseteq \mathbb{R}^d$ is a (k, ϵ)-coreset for k-median clustering if for every set $C = \{c_1, \cdots, c_k\} \subset \mathbb{R}^d$ of k centers we have $|\textit{COST}(P, C) - \textit{COST}(S, C)| \leq \epsilon \cdot \textit{COST}(P, C)$.*

2.2 The Sliding Window Model

Let (X, d) be a metric space and $P \subseteq X$ a point set of size $|P| = n$. In the insertion-only streaming model [2, 29, 12], we think of a (possibly adversarial) permutation $p_1, p_2, \cdots, p_n$ of P, presented as a data stream. We assume that we have some function f, defined on sets of points. The goal is then to compute the (approximate) value of f evaluated on the stream, using $o(n)$ space. We say that point p_N *arrives* at time N.

The *sliding window model* [20] is a generalization of the insertion-only streaming model in which we seek to compute function f over only the most recent elements of the stream. Given a current time N, we consider a window $\mathcal{W}$ of size W consisting of points $p_s, p_{s+1}, \ldots, p_N$, where $s = \max\{1, N - W + 1\}$. We assume that W is such that we cannot store all of window $\mathcal{W}$ in memory. A point p_i in the current window $\mathcal{W}$ is called *active*. At time N, point p_i for which $i < N - W + 1$ is called *expired*.

2.3 Smooth Functions and Smooth Histograms

Definition 4 ((ϵ, ϵ')-smooth function [8]) *Let f be a function defined on sets of points, and let $\epsilon, \epsilon' \in (0, 1)$. We say f is an (ϵ, ϵ')*-smooth function *if f is non-negative (i.e., $f(A) \geq 0$ for all sets A), non-decreasing (i.e., for $A \subseteq B$, $f(A) \leq f(B)$), and polynomially bounded (i.e., there exists constant $c > 0$ such that $f(A) = O(|A|^c)$) and for all sets A, B, C*

$$f(B) \geq (1 - \epsilon) f(A \cup B) \quad \textit{implies} \quad f(B \cup C) \geq (1 - \epsilon') f(A \cup B \cup C).$$

Interestingly, a broad class of functions fit this definition. For instance, sum, count, minimum, frequency moments and the length of the longest subsequence are smooth functions. Braverman and Ostrovsky [8] proposed a data structure called *smooth histograms* to maintain smooth functions on sliding windows.

Definition 5 (Smooth histogram [8]) *Let $0 < \epsilon < 1, 0 < \epsilon' < 1$ and $\alpha > 0$, and let f be an (ϵ, ϵ')-smooth function. Suppose that there exists an insertion-only streaming algorithm $\mathcal{A}$ that computes an α-approximation f' of f. The* smooth histogram *consists of an increasing set of indices $X_N = \{x_1, x_2, \cdots, x_t = N\}$ and t instances $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_t$ of $\mathcal{A}$ such that*

(1) Either p_{x_1} is expired and p_{x_2} is active or $x_1 = 0$.

(2) For $1 < i < t - 1$ one of the following holds

(a) $x_{i+1} = x_i + 1$ and $f'([x_{i+1}, N]) \leq (1 - \epsilon') f'([x_i, N])$,

(b) $f'([x_{i+1}, N]) \geq (1 - \epsilon) f'([x_i, N])$ and if $i \in [t - 2]$, $f'([x_{i+2}, N]) \leq (1 - \epsilon') f'([x_i, N])$.

(3) $\mathcal{A}_i = \mathcal{A}([x_i, N])$ maintains $f'([x_i, N])$.

Observe that the first two elements of sequence X_N always sandwich the current window $\mathcal{W}$, in the sense that $x_1 \leq N - W \leq x_2$. Braverman and Ostrovsky [8] used this observation to show that at any time N, one of either $f'([x_1, N])$ or $f'([x_2, N])$ is a good approximation to $f'([N - W, N])$, and is thus a good approximation to $f([N - W, N])$. In particular, they proved the following theorem.

Theorem 6 ([8]) *Let $0 < \epsilon, \epsilon' < 1$ and $\alpha, \beta > 0$, and let f be an (ϵ, ϵ')-smooth function. Suppose there exists an insertion-only streaming algorithm $\mathcal{A}$ that calculates an α-approximation f' of f using $g(\alpha)$ space and $h(\alpha)$ update time. Then there exists a sliding window algorithm that maintains a $(1 \pm (\alpha + \epsilon))$-approximation of f using $O(\beta^{-1} \cdot \log n \cdot (g(\alpha) + \log n))$ space and $O(\beta^{-1} \cdot \log n \cdot h(\alpha))$ update time.*

However, many clustering problems including the k-median clustering (see Lemma 7) are not smooth functions and so, Theorem 6 does not give a sliding windows algorithm for these problems.

Lemma 7 *k-median clustering is not a smooth function.*

Proof : As a counterexample, consider distinct points $p, q, r \in X$. Then $\text{OPT}(\{p, q\}, 2) = 0$ and $\text{OPT}(\{q\}, 2) = 0$. However, $\text{OPT}(\{q, r\}, 2) = 0$ and $\text{OPT}(\{p, q, r\}, 2) = \min(d(p, q), d(p, r), d(q, r))$, which can be arbitrarily large. □

VC-Dimension and ϵ-Sample. We briefly review the definition of VC-dimension and ϵ-sample as presented in Alon and Spencer's book [3] in Appendix B.

3 Metric k-Median Clustering in Sliding Windows

We introduce the first polylogarithmic-space $O(1)$-approximation for metric k-median clustering in the sliding window model. Our algorithm requires $O(k^3 \log^6 n)$ space and $O(\text{poly}(k, \log n))$ update time. We note that our algorithm is easily modified to accommodate k-means clustering. This modified algorithm will have the same time and space bounds with a larger approximation ratio that is nevertheless still $O(1)$.

3.1 Smoothness

k-median and k-means are not smooth (see the Appendix for an example), so the techniques of [8] do not apply directly, but Lemma 10 shows that k-median clustering does possess a property similar to smoothness.

Definition 8 (λ-approximate Triangle Inequality) *Non-negative symmetric function $d : X \times X \rightarrow \mathbb{R}_{\geq 0}$ satisfies the* λ-approximate triangle inequality *if $d(a, c) \leq \lambda\,(d(a, b) + d(b, c))$ for every $a, b, c \in X$.*

We note that a metric d satisfies the 1-approximate triangle inequality by definition and that for any $p \geq 1$, d^p obeys the 2^{p-1}-approximate triangle inequality, since $(x+y)^p \leq 2^{p-1}(x^p+y^p)$ for all non-negative x, y and $p \geq 1$. Thus, if $p = O(1)$, Theorem 16 provides an $O(1)$-approximation for the clustering objective $\sum d^p(x, C)$. The case $p = 2$ yields an $O(1)$-approximate solution for k-means.

Definition 9 *Let P and $C = \{c_1, \ldots, c_k\}$ be sets of points from metric space $(\mathcal{X}, d)$. A map $t : P \rightarrow C$ is called a* clustering map *of P for set C. If $\sum_{x \in P} d(x, t(x)) \leq \beta \cdot \text{OPT}(P, k)$, then we say that t is a β-*approximate clustering map *of P for set C. The difference between a clustering map $t(x)$ and the intuitive map $\arg\min_{c \in \{c_1, \ldots, c_k\}} d(x, c)$ is that t need not map each point x to its nearest center.*

Lemma 10 *Let d be a non-negative symmetric function on $X \times X$ satisfying the λ-approximate triangle inequality and let $A, B \subset X$ be two disjoint sets of points such that*

(1) $\text{OPT}(A \cup B, k) \leq \gamma \text{OPT}(B, k)$.

(2) There exists β-approximate clustering map t of $A \cup B$ such that $\forall i \in [k] : |t^{-1}(c_i) \cap A| \leq |t^{-1}(c_i) \cap B|$.

Then for any $C \subseteq X$ we have $\text{OPT}(A \cup B \cup C, k) \leq (1 + \lambda + \beta\gamma\lambda)\text{OPT}(B \cup C, k)$.

Proof : The proof is provided in the Appendix A.2. □

If we could ensure that the inequality conditions required by Lemma 10 hold, then we could apply the ideas from smooth histograms. The following two lemmas suggest a way to do this.

Lemma 11 *Let $A \cup B$ be a set of n points received in an insertion-only stream, appearing in two phases so that all points in A arrive before all points in B, and assume that the algorithm is notified when the last point from A arrives. Using $O(k \log^2 n)$ space, it is possible to compute an $O(1)$-approximate clustering map t for $A \cup B$ as well as the exact values of* $\{(|t^{-1}(c_i) \cap A|, |t^{-1}(c_i) \cap B|)\}_{i \in [k]}$.

Proof : Given a set of points P presented in a stream D, the PLS algorithm presented in [12] uses $O(k \log^2 n)$ space to compute a weighted set S such that $\text{COST}(D, S) \leq \alpha \text{OPT}(D, k)$ for some constant α ([12], Theorem 1). Using a theorem from [26], it is shown in [12] that $\text{OPT}(S, k) \leq 2(1 + \alpha)\text{OPT}(D, k)$. It follows immediately that running an offline ξ-approximation for k-median on S yields a set of k centers that constitutes an $(\alpha + 2\xi(1 + \alpha))$-approximate k-median solution for the original stream D.

The PLS algorithm uses as a subroutine the online facility location algorithm due to Meyerson [33]. Thus, each point in S can be viewed as a facility serving one or more points in P. Therefore, running the PLS algorithm on stream D yields a map $r : P \to S$ such that $r(p) \in S$ is the facility that serves point $p \in P$. Running a ξ-approximation on the set S to obtain centers $\{c_1, \ldots, c_k\}$, yields a map $q : S \to \{c_1, \ldots, c_k\}$ such that point $s \in S$ is connected to $q(s)$.

Given disjoint multisets A and B, PLS yields maps $r_A : A \to S_A$ and $r_B : B \to S_B$. Running the ξ-approximation on $S_A \cup S_B$, we obtain a map $q : S_A \cup S_B \to \{c_1, \ldots, c_k\}$, from which we have a β-approximate clustering map of $A \cup B$ given by $t(x) = q(r_A(x))$ if $x \in A$ and $t(x) = q(r_B(x))$ if $x \in B$, from which we can directly compute $|t^{-1}(c_i) \cap A| = |q^{-1}(c_i) \cap S_A|$ and similarly for $|t^{-1}(c_i) \cap B|$. □

The previous lemma showed how we can check whether condition 2 in Lemma 10 holds over two phases. We now extend this to the case where the stream has an arbitrary number of phases.

Lemma 12 *Let $A_1 \cup \cdots \cup A_Z$ be a set of n points in an insertion-only stream, arriving in phases $A_1, A_2, \ldots, A_Z$, and assume that the algorithm is notified at the end of each phase A_i. Using $O(Z^2 k \log^2 n)$ space, one can compute for every $1 \leq j < \ell \leq Z$ a β-approximate clustering map $t_{j,\ell}$ for $A_1 \cup \cdots \cup A_Z$ and the exact values of* $\{|t_{j,\ell}^{-1}(c_i) \cap (A_j \cup \cdots \cup A_{\ell-1})|, |t_{j,\ell}^{-1}(c_i) \cap (A_j \cup \cdots \cup A_Z)|\}_{i \in [k]}$ *for that map.*

Proof : This lemma is a natural extension of Lemma 11. Details of the proof are in the Appendix. □

Lemma 12 suggests one way to ensure that the conditions of Lemma 10 are met– simply treat every point as a phase– but this would require running $O(W^2)$ instances of PLS, which would be infeasible. We would like to ensure that the conditions 1 and 2 in 10 hold, while running at most $T \ll W$ many instances of PLS. We can achieve this by starting a new phase only when one of these conditions would otherwise be violated. We will show in Lemma 14 that this strategy incurs only polylogarithmically many phases.

3.2 Algorithm for Sliding Windows

Algorithm 1 produces an approximate k-median solution on sliding windows. The remainder of the section will establish properties of this algorithm, culminating in the main result given in Theorem 16.

The bulk of the bookkeeping required by Algorithm 1 is performed by the UPDATE subroutine, defined in Algorithm 2. In the spirit of [11], central to our approach is a set of indices $\{X_1, X_2, \ldots, X_T\}$. Each X_i is the arrival time of a certain point from the stream. Algorithm 1 runs $O(T)$ instances of PLS on the stream, with the i-th instance running starting with point p_{X_i}. Denote this instance by $\mathcal{A}(X_i)$. The PLS algorithm on input $P = \{p_i, p_{i+1}, \ldots, p_j\}$ constructs a weighted set S and a score R such that $\text{OPT}(P, k) \leq \text{COST}(P, S) \leq R \leq \alpha \text{OPT}(P, k)$. To check the smoothness conditions in Lemma 10 we will use the solutions built up by certain instances of PLS. We keep an array of $O(T^2)$ buckets, indexed as $B(X_i, X_j)$ for $1 \leq i < j \leq T$. In each bucket we store $B(X_i, X_j) = (S_{ij}, R_{ij})$ where $S_{ij} = S(X_i, X_j)$ and $R_{ij} = R(X_i, X_j)$ are, respectively, the weighted set and the cost estimate produced by an instance of PLS running on the substream $\{p_{X_i}, \ldots, p_{X_j}\}$.

Concretely, we run instance $\mathcal{A}(X_i)$ on the stream starting with point p_{X_i}. At certain times, say time N, we will need to store the solution currently built up by this instance. By this we mean that we copy the weighted set and cost estimate as constructed by $\mathcal{A}(X_i)$ on points $\{p_{X_i}, \dots, p_N\}$ and store them in bucket $B(X_i, N)$. We denote this by $B(X_i, N) \leftarrow \text{store}(\mathcal{A}(X_i))$. Instance $\mathcal{A}(X_i)$ continues running after this operation. We can view each $B(X_i, X_j)$ as a snapshot of the PLS algorithm as run on points $\{p_{X_i}, \dots, p_{X_j}\}$. As necessary, we terminate PLS instances and initialize new ones over the course of the algorithm. We assume an offline k-median $O(1)$-approximation algorithm $\mathcal{M}$, and denote by $\mathcal{M}(P)$ the centers returned by running this algorithm on point set P.

Algorithm 1 Metric k-median in Sliding Windows

Input: A stream of points $D = \{p_1, p_2, \dots, p_n\}$ from metric space $(\mathcal{X}, d)$, window size W
Update Process, upon the arrival of new point p_N:

```
1: for $i = 1, 2, \dots, T$ do
2:     $B(X_i, N) \leftarrow \text{store}(\mathcal{A}(X_i))$
3: Begin running $\mathcal{A}(N)$
4: UPDATE()
```

Output: Return the centers and cost estimate from bucket $B(X_1, N)$

Algorithm 2 UPDATE : prevents Algorithm 1 from maintaining too many buckets.

```
1: If $X_2 > N - W$, then $i \leftarrow 1$. Otherwise, $i \leftarrow 2$.
2: while $i \leq T$ do
3:     $j \leftarrow$ the maximal $j'$ such that $\beta R(X_i, N) \leq \gamma R(X_{j'}, N)$. If none exist, $j \leftarrow i + 1$
4:     $C \leftarrow \mathcal{M}(S(X_i, X_j))$
5:     while $i < j$ do
6:         mark$(X_i)$
7:         $i \leftarrow$ the maximal $\ell$ such that $|t^{-1}(c) \cap \mathcal{S}_{i,\ell}| \leq |t^{-1}(c) \cap \mathcal{S}_{\ell,T}|$ for all $c \in C$. If none exist, $i \leftarrow i + 1$
8:     mark$(X_j)$
9:     $i \leftarrow j + 1$
10: For all unmarked $X_i$, terminate instance $\mathcal{A}(X_i)$
11: Delete all buckets $B(X_i, X_j)$ for which either $X_i$ or $X_j$ is unmarked.
12: Delete all unmarked indices $X_i$; relabel and unmark all remaining indices.
```

Lemma 13 *For any index $m \leq N$, let s be the maximal index such that $[m, N] \subseteq [X_s, N]$. Then $\mathit{OPT}([m, N]) \leq (2 + \beta\gamma)\mathit{OPT}([X_s, N])$.*

Proof : If $X_s = m$, there is nothing to prove, so assume $X_s < m$. This implies that index m was deleted at some previous time Q. The result follows by considering the state of the algorithm at this time. Algorithm 1 maintains the conditions required by Lemma 10 to ensure that for any suffix C, $\text{OPT}([X_i, Q] \cup C) \leq (2 + \beta\gamma)\text{OPT}([m, Q] \cup C)$. Letting $C = [Q + 1, N]$ yields the result. Details are given in the Appendix. □

In what follows, let $\text{OPT}' = \text{OPT}(W, k)/\rho(W)$ and $n = |W|$, the size of the window.

Lemma 14 *Algorithm 1 maintains $O(k \log n \log \mathit{OPT}')$ buckets.*

Proof : Each iteration of the loop on Line 2 decreases $R(X_i)$ by a factor of γ/β, so this loop is executed $O(\log_{\gamma/\beta} \text{OPT}')$ times. In each iteration of the loop on Line 5, the size of at least one of the k clusters decreases by half. Each set has size at most n, so this loop is executed $O(k \log_2 n)$ times. Each execution of each loop stores one bucket, so in total $O(k \log n \log \text{OPT}')$ buckets are stored in these nested loops. □

Lemma 15 *Assuming* $\textsf{OPT}' = \text{poly}(n)$*, Algorithm 1 requires* $O(\text{poly}(k, \log n))$ *update time.*

Proof : The runtime of Algorithm 1 is dominated by the $O(Tk^2 \log^2 n)$ time required to partition the buckets and that $T = O(k \log^2 n)$ by Lemma 14. A more detailed proof is given in the Appendix. □

Theorem 16 *Assuming* $\textsf{OPT}' = \text{poly}(n)$*, there exists an* $O(1)$*-approximation for the metric* k*-median problem on sliding windows using* $O(k^3 \log^6 n)$ *space and* $O(\text{poly}(k, \log n))$ *update time.*

Proof : Using Algorithm 1, we output a β-approximation for $[X_1, N]$, which includes the current window $\mathcal{W}$. By Lemma 13, $\textsf{OPT}([X_1, N], k) \leq (2+\beta\gamma)\textsf{OPT}(\mathcal{W}, k)$, and thus $R(X_1, N) \leq \beta(1+\lambda+\beta\gamma\lambda)\textsf{OPT}(\mathcal{W}, k)$. Let C be the approximate centers for $[X_1, N]$. We have the following inequalities:

$$\begin{aligned} \textsf{COST}([X_1, N], C) &\leq \beta\textsf{OPT}([X_1, N]) \\ \textsf{OPT}([X_1, N]) &\leq (1 + \lambda + \beta\gamma\lambda)\textsf{OPT}(\mathcal{W}, k) \\ \textsf{COST}(\mathcal{W}, C) &\leq \textsf{COST}([X_1, N], C), \end{aligned}$$

where the last equation follows from the fact that $[X_1, N]$ contains the current window. Connecting these inequalities, we have $\textsf{COST}(\mathcal{W}, C) \leq \beta(1 + \lambda + \beta\gamma\lambda)\textsf{OPT}(\mathcal{W}, k)$, as desired.

For the space bound, note that for each $1 \leq i < j \leq T$, bucket $B(X_i, X_j)$ contains the output of an instance of PLS. Each of these $O(T^2)$ instances requires $O(k \log^2 n)$ space, and $T = O(k \log n \log \textsf{OPT}')$ by Lemma 14, so our assumption that $\textsf{OPT}' = \text{poly}(n)$ implies that we use $O(k^3 \log^6 n)$ space in total. □

4 Euclidean Coresets on Sliding Windows

In this section we first explain a coreset technique that unifies many of the known coreset techniques. Then we explain the merge-and-reduce method. Finally we develop our sliding window algorithm for coresets.

A Unified Coreset Technique Algorithm. Many coreset techniques for Euclidean spaces partition the point set into small *regions* (sometimes called *cells*) and take a small number of points from each region of the partition either randomly or deterministically. For each region, each of the chosen points is assigned a weight, which is often the number of points in that region divided by the number of chosen points from that region. Some of the well-known coreset techniques that are in this class are (1) the coreset technique due to Har-Peled and Mazumdar for the k-median and the k-means in low dimensional spaces [29]; (2) the coreset technique due to Chen for the k-median and the k-means problems in high-dimensional spaces [13]; (3) the coreset technique due to Feldman, Fiat and Sharir for the j-subspace problem in low dimensional spaces [21]; (4) the coreset technique due to Feldman, Monemizadeh, Sohler and Woodruff for the j-subspace problem in high-dimensional spaces [24]. We unify this class of coreset techniques in Algorithm 3. In the sequel, we will use this unified view to develop a sliding window streaming algorithm for this class of coreset techniques. We will give the proofs for k-median and the k-means in low-and high-dimensional spaces and we defer the proofs for the j-subspace problem in low-and high-dimensional spaces to the full version of this paper.

Lemma 17 *Let* P *be a point set of size* n *in a* d*-dimensional Euclidean space* $\mathbb{R}^d$ *and* $0 < \epsilon \leq 1$ *be a parameter. Suppose we invoke one of the* (k, ϵ)*-coreset techniques of [29] or [13] and let* $\mathcal{K}$ *be the reported coreset and* $\Lambda_{\mathcal{K}}$ *be the corresponding partition of* P*. Suppose that for every region* $R \in \Lambda_{\mathcal{K}}$ *containing* n_R *points from* P*, we delete or insert up to* ϵn_R *points. Let* $\mathcal{K}'$ *be the coreset reported by Algorithm 3 after these deletions or insertions. Then,* $\mathcal{K}'$ *is a* (k, ϵ)*-coreset of* $\mathcal{K}$*.*

Algorithm 3 Algorithm $\mathcal{A}$: A unified coreset technique

Input: A set P of n points, a constant c and two parameters $0 < \epsilon, \delta \le 1$.
Algorithm:

```
1: Suppose we have a $(k, \epsilon)$-coreset technique $\mathcal{CC}$ that returns a partition $\Lambda_{\mathcal{K}}$ of $\mathbb{R}^d$.
2: Let $d_{VC}$ be the VC-D. of range space $(P, \mathcal{R})$ s.t. $\mathcal{R}$ is the set of shapes in $\mathbb{R}^d$ similar to regions in $\Lambda_{\mathcal{K}}$.
3: Suppose $\mathcal{CC}$ samples a set of size $s_{\mathcal{CC}} = f(n, d, \epsilon, \delta)$ from R where $s_{\mathcal{CC}}$ is a function of $n, d, \epsilon, \delta$.
4: For each region $R \in \Lambda_{\mathcal{K}}$ we treat a weighted point $p$ of weight $w_p$ as $w_p$ points at coordinates of $p$.
5: Sample $r = \min\left(|R|, \max\left(s_{\mathcal{CC}}, O(d_{VC}\epsilon^{-2}\log(n) \cdot \log(\frac{d_{VC}\log(n)}{\epsilon\delta}))\right)\right)$ points uniformly at random.
6: Each such a sampled point receives a weight of $n_R/r$ where $n_R$ is the number of points in region R.
7: Let $\mathcal{K}$ be the union of all (weighted) sampled points that are taken from regions in partition $\Lambda_{\mathcal{K}}$.
```

Output: A coreset $\mathcal{K}$ of P and its partition $\Lambda_{\mathcal{K}}$.

Proof : Proofs for [29] and [13] are in Sections B.1 and B.2, respectively. □

Merge and Reduce Operation. The merge and reduce method inspired by a complete binary tree is a generic method in computational geometry to implement non-streaming algorithms in the insertion-only streaming model. Let P be a set of n points [1], presented in a streaming fashion. The pseudocode of merge and reduce operation is given below. In this pseudocode, we use buckets $\{B_1, B_1', B_2, B_2', \cdots, B_i, B_i', \cdots, B_{\log(n)-1}, B'_{\log(n)-1}, B_{\log(n)}\}$, where buckets B_i and B_i' can be considered as buckets in level i of the merge-and-reduce tree and are of size x_i, which will be determined for each concrete problem. All buckets B_i, B_i' for $i \in [\log n]$ are initialized to zero in the beginning of the stream.

Algorithm 4 MERGEREDUCE Operation

Input: A stream $S = [p_r, p_{r+1}, p_{i+2}, \cdots, p_{N-1}]$ of length $|S| = n^c$ for a constant c and a point p_N.
Update Process, upon the arrival of new point p_N:

```
1: Let $i = 1$ and add $p_N$ to $B_i$ if $B_i$ is not full, otherwise to $B_i'$.
2: while $B_i$ and $B_i'$ are both full and $i \le \log(n)$ do
3:     Compute coreset $Z$ and partition $\Lambda_Z$ using Algorithm $\mathcal{A}(B_i \cup B_i', \epsilon_i = \epsilon/(2\log n), \delta/n^c)$.
4:     Delete the points of buckets $B_i$ and $B_i'$.
5:     Let $B_{i+1}$ be $Z$ if $B_{i+1}$ is empty, otherwise let $B'_{i+1}$ be $Z$.
6:     Let $i = i + 1$.
```

Output: Return coreset $S_X = \cup_{i=1}^{\log(n)} B_i \cup B_1'$ and partition $\Lambda_{S_X} = \cup_{i=1}^{\log(n)} \Lambda_{B_i} \cup \Lambda_{B_1'}$.

The next lemma shows that the (k, ϵ)-coreset maintained by Algorithm MERGEREDUCE well-approximates the density of subsets of point set P within every region of partition Λ_{B_i} for $i \in [\log n]$. The proof of this lemma is given in Appendix B.3.

Lemma 18 *Let B_i be the bucket at level i of Algorithm* MERGEREDUCE *with (k, ϵ)-coreset B_i and partition Λ_{B_i}. Suppose the original points in the subtree B_i is subset $P_i \subseteq P$. For every region $R_i \in \Lambda_{B_i}$,*

$$\big||P_i \cap R_i| - |B_i \cap R_i|\big| \le \sum_{\textit{level } j=2}^{i-1} \sum_{\textit{node } x_j \textit{ in level } j} \epsilon_j \Big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \Big) ,$$

where O_{x_j} is a multi-set of points at node x_j in level i of the merge-and-reduce tree such that for every point $p \in O_{x_j}$ with weight w_p, we add w_p copies of p to O_{x_j}.

[1] Here we assume n is known in advance. The case where n is not known in advance can be accommodated using repeated guesses for n. See, for example [29].

Next we show the error of Lemma 18 is small.

Lemma 19 *Let* B_i *be the bucket at level* i *of Algorithm* MERGEREDUCE *with* (k, ϵ)*-coreset* B_i *and partition* Λ_{B_i}. *Suppose the original points in the subtree* B_i *is subset* $P_i \subseteq P$. *For every* $j \in [\log n]$, *if we replace* ϵ_j *by* $\epsilon/(2j)$, *the error* $\sum_{level\ j=2}^{i-1} \sum_{node\ x_j\ in\ level\ j} \epsilon_j \big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \big)$ *of* $\big||P_i \cap R_i| - |B_i \cap R_i|\big|$ *is* ϵ*-fraction of the cost of* P_i *in terms of* k *arbitrary* j*-dimensional subspaces and so can be ignored.*

Proof : Let us look at the sub-terms in the error term $\sum_{\text{level } j=2}^{i-1} \sum_{\text{node } x_j \text{ in level } j} \epsilon_j \big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \big)$. For fixed node x_j in level j, $\epsilon_j \big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \big)$ is the ϵ_j-fraction change in region $R \in \Lambda_{x_j}$ of Algorithm 3. Using Lemma 17, for each one of the coreset techniques in [29] and [13], the new (k, ϵ)-coreset after these changes in every region is again a $(k, 2\epsilon)$-coreset of point set P_i. Here we use this fact that a (k, ϵ)-coreset of a (k, ϵ)-coreset of P is a $(k, 2\epsilon)$-coreset of P. We have i levels, each one of which is a (k, ϵ)-coreset of P_i. Thus, the error term is i times the error of one of one (k, ϵ)-coreset of P_i. If we replace ϵ_j by $\epsilon/(2j)$, the overall error would be the same the error of one (k, ϵ)-coreset of P_i, which can be ignored. In Algorithm MERGEREDUCE we replace $\epsilon_i = \frac{\epsilon}{2 \log n}$ for all levels $i \in [\log n]$. □

Coreset Maintenance in Sliding Windows. In this section we develop Algorithm SWCORESET, a sliding window streaming algorithm for the class of coreset techniques (including the coreset techniques of [29], [13], [21], and [24]) that fit into Algorithm 3. We show that the number of (k, ϵ)-coresets that we maintain is upper bounded by the size of one (k, ϵ)-coreset times $O(\epsilon^{-2} \log n)$.

Algorithm 5 SWCORESET

```
Input: A stream $S = [p_1, p_2, p_3, \ldots, p_N, \ldots, p_n]$ of points $\mathbb{R}^d$.
Output: A coreset $\mathcal{K}_{x_1}$ for window $W$, i.e., points $\{p_{N-W+1}, \ldots, p_N\}$.
Update Process, upon the arrival of new point $p_N$:
 1: for $x_i \in X = [x_1, x_2, \ldots, x_t]$ where $x_i \in \{1, \ldots, N\}$ do
 2:     Let $(\mathcal{K}_{x_i}, \Lambda_{\mathcal{K}_{x_i}})$=MERGEREDUCE($[x_i, N] = \{p_{x_i}, \ldots, p_{N-1}\}, p_N$) be the coreset and its partition.
 3: Let $t = t + 1$, $x_t = N$.
 4: Let $(\mathcal{K}_{x_t}, \Lambda_{\mathcal{K}_{x_t}})$=MERGEREDUCE($\{\}, p_N$) be the coreset and the partition of single point $p_N$.
 5: for  $i = 1$ to $t - 2$ do
 6:     Find greatest $j > i$ such that for every region $R$ in partition $\Lambda_{\mathcal{K}_{x_j}}$ at most $\epsilon w_R$ weight is in $[x_i, x_j]$,
        where $w_R$ is the sum of the weights of the coreset points in $R$.
 7:     for $i < r < j$ do
 8:         Delete $x_r$, coreset $\mathcal{K}_{x_r}$ and partition $\Lambda_{\mathcal{K}_{x_r}}$. Update the indices in sequence $X$ accordingly.
 9: Let $i$ be the smallest index such that $p_{x_i}$ is expired and $p_{x_{i+1}}$ is active.
10: for $r < i$ do
11:     Delete $x_r$ and coreset $\mathcal{K}_{x_r}$ and partition $\Lambda_{\mathcal{K}_{x_r}}$, and update the indices in sequence $X$.
Output Process:
 1: Return coreset $\mathcal{K}_{x_1}$ maintained by MERGEREDUCE($[x_1, N] = \{p_{x_1}, \ldots, p_{N-1}\}, p_N$).
```

Theorem 20 *Let* $P \subseteq \mathbb{R}^d$ *be a point set of size* n. *Suppose the optimal cost of clustering of point set* P *is* $\textsc{opt}_P = n^{O(c)}$ *for some constant* c. *Let* s *be the size of a coreset (constructed using one of the coreset techniques [29, 13]) that merge-and-reduce method maintains for* P *in the insertion-only streaming model. There exists a sliding window algorithm that maintains this coreset using* $O(s^2 \epsilon^{-2} \log n)$ *space.*

The proof is given in appendix B.4.

References

[1] P. K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[3] N. Alon and J. Spencer. *The Probabilistic Method*. J. Wiley and Sons, 2000.

[4] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[5] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the 9th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 234–243, 2003.

[6] P. Beame, R. Clifford, and W. Machmouchi. Element distinctness, frequency moments, and sliding windows. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 290–299, 2013.

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

[8] V. Braverman and R. Ostrovsky. Effective computations on sliding windows. *SIAM Journal on Computing*, 39(6):2113–2131, 2010.

[9] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.

[10] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 26–40. SIAM, 2011.

[11] Vladimir Braverman and Rafail Ostrovsky. Smooth histograms on sliding windows. In *FOCS*, 2007.

[12] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003.

[13] K. Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

[14] G. Cormode. The continuous distributed monitoring model. *SIGMOD Record*, 42(1):5–14, 2013.

[15] G. Cormode and M. N. Garofalakis. Streaming in a connected world: querying and tracking distributed data streams. In *EDBT*, page 745, 2008.

[16] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[17] G. Cormode and S. Muthukrishnan. What's new: finding significant differences in network data streams. *IEEE/ACM Transactions on Networking*, 13(6):1219–1232, 2005.

[18] M. S. Crouch, A. McGregor, and D. Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of the 21st Annual European Symposium on Algorithms (ESA)*, pages 337–348, 2013.

[19] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler. $(1 + \varepsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013.

[20] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[21] D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 315–324, 2006.

[22] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 569–578, 2011.

[23] D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd Annual Symposium on Computational Geometry (SoCG)*, pages 11–18, 2007.

[24] D. Feldman, M. Monemizadeh, C. Sohler, and D. Woodruff. Coresets and sketches for high dimensional subspace problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–649, 2010.

[25] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

[26] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

[27] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society Boston, MA, USA, 2011.

[28] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the 21st Annual Symposium on Computational Geometry (SoCG)*, pages 126–134, 2005.

[29] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.

[30] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

[31] P. Indyk and E. Price. k-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 627–636, 2011.

[32] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. FOCS*, 1999.

[33] A. Meyerson. Online facility location. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 426–, Washington, DC, USA, 2001. IEEE Computer Society.

[34] M. Monemizadeh. Non-uniform sampling in clustering and streaming. *PhD Thesis, Available at: http://www.tks.informatik.uni-frankfurt.de/data/doc/Thesis.pdf*, September, 2010.

[35] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[36] M. Osborne, S. Moran, R. McCreadie, A. Von Lunen, M. Sykora, E. Cano, N. Ireson, C. MacDonald, I. Ounis, Y. He, T. Jackson, F. Ciravegna, and A. O'Brien. Real-time detection, tracking and monitoring of automatically discovered events in social media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, USA, 2014.

[37] List of open problems in sublinear algorithms: Problem 20. `http://sublinear.info/20`.

A Missing Proofs and Further Results from Metric k-Median Clustering in Sliding Windows

A.1 k-median and k-means are not smooth functions

Claim 21 *k-median and k-means clustering are not smooth functions. That is, there exist sets of points A, B and C such that for any $\gamma, \beta > 0$, $\mathit{OPT}(A \cup B, k) \le \gamma \mathit{OPT}(B, k)$ but $\mathit{OPT}(A \cup B \cup C) > \beta \mathit{OPT}(B \cup C)$.*

Proof : Let $A \cup B$ consist of k distinct points, with $A \neq \emptyset$, $B \neq \emptyset$ and A contains one or more points not in B. Then B contains at most $k-1$ distinct points, and $\mathsf{OPT}(A \cup B, k) = 0 \le \gamma\mathsf{OPT}(B, k) = 0$ for any γ. Consider a set C consisting of a single point and satisfying $C \cap (A \cup B) = \emptyset$. Then $A \cup B \cup C$ has at least $k+1$ distinct points, so that $\mathsf{OPT}(A \cup B \cup C, k) > 0$, while $\mathsf{OPT}(B \cup C, k) = 0$. Then for any β, we have that $\mathsf{OPT}(A \cup B \cup C, k) > 0 = \beta\mathsf{OPT}(B \cup C, k)$. □

A.2 Proof of Lemma 10

Proof of Lemma 10: Let t be the β-approximate clustering map of $A \cup B$ in the hypothesis. By assumption, t induces partitions $A_1, A_2, \ldots, A_k$ and $B_1, B_2, \ldots, B_k$ of A and B, respectively, given by $A_i = t^{-1}(c_i) \cap A$ and $B_i = t^{-1}(c_i) \cap B$. Since $|A_i| \le |B_i|$ for all $i \in [k]$ by assumption, for each $i \in [k]$ there exists a one-to-one mapping g_i from A_i to a subset of B_i. Letting $a \in A_i$, we have

$$\lambda^{-1} d(a, g_i(a)) \le d(a, t(a)) + d(g_i(a), t(a)) = d(a, t(a)) + d(g_i(a), t(g_i(a))),$$

where the first inequality is the approximate triangle inequality and the second inequality follows from the fact that $t(g_i(a)) = c_i = t(a)$ by definition of g_i. Thus,

$$\begin{aligned}\lambda^{-1} \sum_{i=1}^{k} \sum_{a \in A_i} d(a, g_i(a)) &\le \sum_{i=1}^{k} \sum_{a \in A_i} \left[d(a, t(a)) + d(g_i(a), t(g_i(a)))\right] \\ &\le \sum_{x \in A \cup B} d(x, t(x)) \le \beta\mathsf{OPT}(A \cup B, k) \le \beta\gamma\mathsf{OPT}(B, k), \end{aligned} \tag{1}$$

where the first inequality follows from the approximate triangle inequality, the second inequality follows from the definition of g_i, the third inequality follows since t is β-approximate, and the fourth inequality holds by assumption. Let s be an optimal clustering map for $B \cup C$ (i.e., s is 1-approximate). Then we have

$$\sum_{i=1}^{k} \sum_{a \in A_i} d(g_i(a), s(g_i(a))) \le \sum_{i=1}^{k} \sum_{b \in B_i} d(b, s(b)) \le \mathsf{OPT}(B \cup C, k). \tag{2}$$

Bounding the cost of connecting A to the optimal centers of $B \cup C$, we obtain

$$\begin{aligned}\sum_{a \in A} d(a, s(a)) &= \sum_{i=1}^{k} \sum_{a \in A_i} d(a, s(a)) \le \sum_{i=1}^{k} \sum_{a \in A_i} d(a, s(g_i(a))) \\ &\le \sum_{i=1}^{k} \sum_{a \in A_i} \lambda\left[d(a, g_i(a)) + d(g_i(a), s(g_i(a)))\right] \le \lambda\mathsf{OPT}(B \cup C, k) + \beta\gamma\lambda\mathsf{OPT}(B, k),\end{aligned}$$

where the first inequality follows from the fact that $s(a)$ is the closest center to a by definition, the second inequality follows from the approximate triangle inequality, and the third inequality follows from equations (1) and (2). Thus we conclude that

$$\begin{aligned}\mathsf{OPT}(A \cup B \cup C, k) &\leq \sum_{a \in A} d(a, s(a)) \ + \sum_{x \in B \cup C} d(x, s(x)) \\ &\leq (1+\lambda)\mathsf{OPT}(B \cup C, k) + \beta\gamma\lambda\mathsf{OPT}(B, k) \leq (1+\lambda+\beta\gamma\lambda)\mathsf{OPT}(B \cup C, k).\end{aligned}$$

□

A.3 Proof of Lemma 12

Proof of Lemma 12: Using our algorithm from Lemma 11, we proceed as before until we are notified of the first point of A_2. Here, we store $\mathcal{S}_{1,2} \leftarrow \text{PLS}(A_2)$, but we continue running this instance of the PLS algorithm. As before, we commence a new instance of the PLS algorithm beginning with the first point of A_2. In general, whenever a transition occurs to A_j, for all $i < j$ we store $\mathcal{S}_{i,j} \leftarrow \text{PLS}(A_i \cup \cdots \cup A_{j-1})$ and continue running all instances. As a result, we maintain sets $\mathcal{S}_{i,j}$ for $i \in [Z]$ and $j > i$. There are $O(Z^2)$ such sets, each of size $O(k \log^2 n)$. When we wish to compute the β-approximate map, we run an offline $O(1)$-approximation on $\mathcal{S}_{j,\ell} \cup \mathcal{S}_{\ell,Z}$. The cluster sizes are computed as in Lemma 11. □

A.4 Proof of Lemma 13

Proof of Lemma 13 : If $X_s = m$, we have equality, so suppose $X_s < m$, implying that index m was previously deleted at some time Q, when p_Q the most recent point to arrive. Let z be the index (assigned before deletion) such that $X_z = m$. During some iteration of the loop on Lines 2-9, it must have held after Line 3 that $X_i \leq X_s < X_z < X_j$. This is because both X_i and X_j are stored, so $s \geq i$ by maximality of s.

Line 3 guarantees $\beta R([X_i, Q]) \leq \gamma R([X_j, Q])$. Since the β-approximation ensures that $\mathsf{OPT}(\cdot) \leq R(\cdot) \leq \beta\mathsf{OPT}(\cdot)$, this implies that $\mathsf{OPT}([X_i, Q]) \leq \gamma\mathsf{OPT}([X_z, Q])$. Let t denote the (uncalculated, but existent) β-approximate map of $[X_i, Q]$ as in Lemma 11. The loop on Line 5 ensures that $|t^{-1}(c_w) \cap S_{i,z}| \leq |t^{-1}(c_w) \cap S_{i,T}|$ for every $w \in [k]$. Therefore, Lemma 10 guarantees that for any suffix C, $\mathsf{OPT}([X_i, Q] \cup C) \leq (2+\beta\gamma)\mathsf{OPT}([m, Q] \cup C)$. By letting $C = [Q+1, N]$, the result is obtained. □

A.5 Proof of Lemma 15

Proof of Lemma 15: Let $h(\cdot)$ be the update time for PLS and $g(\cdot)$ be the time for the offline c-approximation. Feeding new point p_N to all T instances of PLS requires $Th(n)$ time and computing the c-approximation for all $O(\log \mathsf{OPT}) = O(\log n)$ iterations of the loop on Line 2 requires $O(\log n \cdot g(k \log^2 n))$ time. Partitioning each bucket requires $O(Tk^2 \log^2 n)$ time, and finding the maximal index on Line 7 requires $O(T^2 k)$ time. In total, an update takes $O(h(n) \cdot T + \log n \cdot g(k \log^2 n) + Tk^2 \log^2 n + T^2 k)$ time. By Lemma 14, $T = O(k \log^2 n)$. By [12], the update time of PLS is polynomial in its argument, and using any of a number of offline $O(1)$-approximations for k-median, for example, [4, 32]. Moreover, $g(\cdot)$ and $h(\cdot)$ are such that the last two terms are the largest factors, resulting in an update time of $O(k^3 \log^4 n)$. □

A.6 Algorithm for Metric k-MEANS

The metric k-means problem is defined similarly.

Definition 22 (Metric k-means) *Let* $P \subseteq X$ *be a set of* n *points in a metric space* (X, d) *and let* $k \in \mathbb{N}$ *be a natural number. Suppose* $C = \{c_1, \ldots, c_k\}$ *is a set of* k *centers. The clustering of point set* P *using* C *is the partitioning of* P *such that a point* $p \in P$ *is in cluster* C_i *if* $c_i \in C$ *is the nearest center to* p *in* C*, that is point* p *is assigned to its nearest center* $c_i \in C$*. The cost of* k*-median clustering by* C *is*

$\textsc{cost}^2(P, C) = \sum_{p \in P} d^2(p, C)$. *The metric* k*-median problem is to find a set* $C \subset P$ *of* k *centers that minimizes the cost* $\textsc{cost}^2(P, C)$*, that is*

$$\begin{aligned}\textsc{cost}^2(P, C) = \sum_{p \in P} d^2(p, C) &= \min_{C' \subset P : |C'| = k} \textsc{cost}^2(P, C') \\ &= \min_{C' \subset P : |C'| = k} \sum_{p \in P} d^2(p, C') \ ,\end{aligned}$$

where $d^2(p, C) = \min_{c \in C} d^2(p, c)$ *and* $d^2(p, C') = \min_{c \in C'} d^2(p, c)$

A concept used by our algorithm for the metric k-means is the notion of σ-separability [10]. Intuitively, data which is separable for a high value of σ is well-clusterable into k-clusters (i.e. removing one center greatly increases the optimal cost).

Definition 23 (σ-separable dataset) *[10] A set of input data is said to be* σ*-separable if the ratio of the optimal* k*-means cost to the optimal* $(k-1)$*-means cost is at most* σ^2.

We now turn to the modifications necessary for k-MEANS. We state the main theorem and explain the necessary changes in the remainder of this section.

Theorem 24 *Assuming* $\textsc{opt}' = \mathrm{poly}(n)$ *and every window is* σ*-separable for some* $\sigma = O(1)$*, there exists a sliding windows algorithm which maintains a* $O(1)$*-approximation for the metric* k-MEDIAN *problem using* $O(k^3 \log^6 n)$ *space and* $O(k^3 \log^4 n)$ *update time.*

In Lemma 11, for k-MEDIAN we had used the **PLS** algorithm to maintain a weighted set $\mathcal{S}$ such that $\mathrm{COST}(\mathcal{P}, \mathcal{S}) \leq \alpha \mathrm{OPT}(\mathcal{P}, k)$ for some constant α. Instead, we now use the insertion-only k-MEANS algorithm of [10]. This algorithm also works by providing a weighted set $\mathcal{S}$ such that $\mathrm{COST}(\mathcal{P}, \mathcal{S}) \leq \alpha \mathrm{OPT}(\mathcal{P}, k)$ for some constant α. Here, the space required is again $O(k \log^2 n)$. The approximation-factor α is now $O(\sigma^2)$ where the data is σ-separable as defined in Definition 23. The second modification is in Algorithm 1. For k-MEDIAN, we had used Lemma 10 with $\lambda = 1$ since the k-MEDIAN function satisfied the 1-approximate triangle inequality (i.e. the standard triangle inequality). For k-MEANS, we now satisfy the 2-approximate triangle inequality, so we use the lemma with $\lambda = 2$. Theorem 16 still holds without modification, so we result in a $O(\sigma^4)$-approximation.

B Missing Proofs and Further Results from Euclidean Coresets in Sliding Windows

Here, we briefly review the definition of VC-dimension and ϵ-sample as presented in Alon and Spencer's book [3]. A *range space* S is a pair (X, R), where X is a set and R is a family of subsets of X. The elements of X are called *points* and the subsets in R are called *ranges*. For $A \subseteq X$, we define the *projection* of R on A as $P_R(A) = \{r \cap A : r \in R\}$. We say A is *shattered* if $P_R(A)$ contains all subsets of A. The *Vapnik-Chervonenkis* dimension (VC-D) of S, which we denote by $d_{VC}(S)$, is the maximum cardinality of a shattered subset of X. If there exist arbitrarily large shattered subsets of X, then $d_{VC}(S) = \infty$. Let (X, R) be a range space with VC-D d, $A \subseteq X$ with $|A|$ finite, and let $0 < \epsilon < 1$ be a parameter. We say that $B \subset A$ is an ϵ*-sample* for A if for any range $r \in R$, $|\frac{|A \cap r|}{|A|} - \frac{|B \cap r|}{|B|}| \leq \epsilon$.

Lemma 25 ([3]) *Let* $S = (X, R)$ *be a range space of VC-D* $d_{VC}(S)$*,* $A \subseteq X$ *with* $|A|$ *finite, and let* $0 < \epsilon < 1$ *be a parameter. Let* $c > 1$ *be a constant and* $0 < \delta < 1$*. Then a random subset* B *of* A *of size* $s = \min(|A|, \frac{c}{\epsilon^2} \cdot (d_{VC}(S) \log(d_{VC}(S)/\epsilon) + \log(1/\delta)))$ *is an* ϵ*-sample for* A *with probability at least* $1 - \delta$.

Lemma 26 ([27] (Chapter 5)) *Let* $S = (X, R)$ *and* $T = (X, R')$ *be two range spaces of VC-D* $d_{VC}(S)$ *and* $d'_{VC}(T)$*, respectively, where* $d_{VC}(S), d'_{VC}(T) > 1$. *Let* $U = \{r \cup r' | r \in R, r' \in R'\}$ *and* $I = \{r \cup r' | r \in R, r' \in R'\}$. *Then the range spaces* $\hat{S} = (X, U)$ *and* $\hat{S}' = (X, I)$ *have* $d_{VC}(\hat{S}) = d_{VC}(\hat{S}') = O((d_{VC}(S) + d'_{VC}(T)) \log(d_{VC}(S) + d'_{VC}(T)))$. *In general, unions, intersections and any finite sequence of combining ranges spaces with finite VC-D results in a range space with a finite VC-D.*

In $\mathbb{R}^d$, the VC-D of a half space is d [27]. Balls, ellipsoids and cubes in $\mathbb{R}^d$ have VC-D $O(d^2)$ as can be easily verified by lifting the points into $O(d^2)$ dimensions, where each one of these shapes in the original space is mapped into a half space.

Next, we first review 2 well-known coreset techniques that fit into the framework of Section 4. These coreset techniques are
(1) Coreset technique of [29] due to Har-Peled and Mazumdar for the k-median and the k-means in low dimensional Euclidean spaces, i.e., when dimension d is constant.
(2) Coreset technique of [13] due to Chen for the k-median and the k-means problems in high dimensional Euclidean spaces, i.e., when dimension d is not constant.

Next, we prove Lemma 17 for each one of these coreset techniques. Interestingly, the coreset technique of [29] is the basis of almost all follow-up coresets for the k-median and the k-means in low dimensional Euclidean spaces. This includes the coreset techniques of [21, 28]. The same is true for the coreset technique of [13] which is the basis of almost all follow-up coresets for the k-median and the k-means in high dimensional Euclidean spaces. This includes the coreset techniques of [23, 24, 34, 22].

Finally we prove Lemma 18.

B.1 Coreset Technique of [29] due to Har-Peled and Mazumdar

We explain their coreset technique for the k-median problem. The same coreset technique works for the k-means problem. We first invoke a α-approximation algorithm on a point set $P \in \mathbb{R}^d$ that returns a set $C = \{c_1, \cdots, c_k\} \subset \mathbb{R}^d$ of k centers. Let C_i be the set of points that are in the cluster of center c_i, i.e. $C_i = \{p \in P : d(p, c_i) \leq \min_{c_j \in C} d(p, c_j)\}$. We consider $\log n + 1$ balls $\mathrm{Ball}_{i,j}(c_i, 2^j \cdot \frac{\mathsf{COST}(P,C)}{n})$ centered at center c_i of radii $2^j \cdot \frac{\mathsf{COST}(P,C)}{n}$ for $j \in \{0, 1, 2, \cdots, \log n\}$. In $\mathrm{Ball}_{i,j}$, we impose a grid $G_{i,j}$ of side length $\frac{\epsilon}{10\sqrt{d}\alpha} \cdot 2^j \cdot \frac{\mathsf{COST}(P,C)}{n}$ and for every non-empty cell c in grid $G_{i,j}$ we replace all points in c with one point of weight n_c, where n_c is the number of points in c. Let $\mathcal{K}_P$ be the set of cells returned by this coreset technique, that is $\mathcal{K}_P = \cup_{c \in G_{i,j}} c$. We also let partition Λ_P to be the set of cells $\Lambda_P = \{G_{i,j} \cap \mathrm{Ball}_{i,j} : i \in [k], j \in [\log n + 1]\}$.

In Step 3 of Algorithm 3 we let $s_{CC} = f(n, d, \epsilon, \delta) = 1$ (which is the number of points that this coreset technique samples from each cell) be 1. Moreover, we let parameter s in Lemma 20 (which is the size of this coreset maintained by merge-and-reduce approach) be $O(k\epsilon^{-d} \log^{2d+2} n)$. We now prove a variant of Lemma 17 for this coreset. Observe that to adjust the parameter ϵ in Lemma 17 for Lemma 27, we replace ϵ by $\frac{\epsilon^2}{5\sqrt{d}}$.

Lemma 27 *Let* $P \subset \mathbb{R}^d$ *be a point set of size* n *and* $k \in \mathbb{N}$ *be a parameter. Let* $\Lambda_P = \{c_1, c_2, \cdots, c_x\}$ *be the partition set of cells returned by the coreset technique of Har-Peled and Mazumdar [29]. Let* $\mathcal{B} = \{b_1, \cdots, b_k\} \subset \mathbb{R}^d$ *be an arbitrary set of* k *centers. Suppose for every cell* $c \in \Lambda_P$ *we delete up to* $\frac{\epsilon^2}{5\sqrt{d}} \cdot n_c$ *points and let* c' *be cell* c *after deletion of these points. We then have*

$$\sum_{c \in \Lambda_P} |\mathit{COST}(c', \mathcal{B}) - \mathit{COST}(c, \mathcal{B})| \leq \epsilon \cdot \mathit{COST}(P, \mathcal{B}) \ ,$$

where $\mathit{COST}(c, \mathcal{B}) = \sum_{p \in c} d(p, \mathcal{B})$ *and* $\mathit{COST}(c', \mathcal{B}) = \sum_{p \in c'} d(p, \mathcal{B})$.

Proof : Let us fix a cell $c \in \Lambda_P$. Suppose $c \in G_{i,j}$ for a particular cluster P_i and $j \in \{0, 1, 2, \cdots, \log n\}$. Assume the nearest center to cell c is $b_c \in \mathcal{B}$. We either have $d(c, b_c) \geq \frac{\ell_c}{\epsilon}$ or $d(c, b_c) < \frac{\ell_c}{\epsilon}$. If we have $d(c, b_c) \geq \frac{\ell_c}{\epsilon}$, then $\text{COST}(c, \mathcal{B}) \geq n_c \cdot d(c, b_c) \geq n_c \cdot \frac{\ell_c}{\epsilon}$. On the other, since we delete up to $\frac{\epsilon^2}{\sqrt{d}} \cdot n_c$ points of cell c, the cost that we lose is at most

$$\frac{\epsilon^2}{\sqrt{d}} \cdot n_c(\sqrt{d}\ell_c + d(c, b_c)) \leq \frac{\epsilon^2}{\sqrt{d}} \cdot n_c \cdot (\sqrt{d}\epsilon + 1) \cdot d(c, b_c) \leq \epsilon \cdot \text{COST}(c, \mathcal{B}) \ .$$

Now suppose we have $d(c, b_c) < \frac{\ell_c}{\epsilon}$. We have two cases, either $j = 0$ or $j > 0$. We first prove the lemma when $j = 0$. Since cell c is in $G_{i,0}$, we have $\ell_c = \frac{\epsilon}{10\sqrt{d}\alpha} \cdot \frac{\text{COST}(P,C)}{n}$. Therefore, $d(c, b_c) < \frac{\ell_c}{\epsilon} \leq \frac{1}{10\sqrt{d}\alpha} \cdot \frac{\text{COST}(P,C)}{n}$. From every cell in $Ball_{i,0} \cap G_{i,0}$ we delete up to $\frac{\epsilon^2}{\sqrt{d}} \cdot n_c$ points. Therefore, the cost that we lose is at most

$$\sum_{c \in Ball_{i,0} \cap G_{i,0}} \frac{\epsilon^2}{\sqrt{d}} \cdot n_c \cdot \frac{1}{10\sqrt{d}\alpha} \cdot \frac{\text{COST}(P, C)}{n} \leq \frac{\epsilon^2}{10d\alpha} \cdot \text{COST}(P, C) \leq \frac{\epsilon^2}{10d} \cdot \text{COST}(P, \mathcal{B}) \ .$$

The second case is when $j > 0$. Recall that $d(c, b_c) < \frac{\ell_c}{\epsilon}$. Observe that since $j > 0$, $d(c, c_i) \geq 2^{j-1} \cdot \frac{\text{COST}(P,C)}{n}$ and $\ell_c = \frac{\epsilon}{10\sqrt{d}\alpha} \cdot 2^j \cdot \frac{\text{COST}(P,C)}{n} \leq \frac{\epsilon}{5\sqrt{d}\alpha} \cdot d(c, c_i)$ which means $d(c, b_c) < \frac{\ell_c}{\epsilon} \leq \frac{1}{5\sqrt{d}\alpha} \cdot d(c, c_i)$. For each such cell c we delete up to $\frac{\epsilon^2}{\sqrt{d}} \cdot n_c$ points. Thus, taking the summation over all i and $j > 0$ yields

$$\begin{aligned}\sum_{i=1}^{k} \sum_{j>0} \sum_{c \in G_{i,j} \cap Ball_{i,j}} \frac{\epsilon^2}{\sqrt{d}} \cdot n_c \cdot d(c, b_c) &\leq \sum_{i=1}^{k} \sum_{j>0} \sum_{c \in G_{i,j} \cap Ball_{i,j}} \frac{\epsilon^2}{\sqrt{d}} \cdot n_c \cdot \frac{1}{5\sqrt{d}\alpha} \cdot d(c, c_i) \\ &\leq \frac{\epsilon^2}{5d\alpha} \cdot \text{COST}(P, C) \leq \frac{\epsilon^2}{5d} \cdot \text{COST}(P, \mathcal{B}) \ .\end{aligned}$$

□

B.2 Coreset Technique of [13] due to Chen

We explain his coreset technique for the k-median problem. The same coreset technique works for the k-means problem. We first invoke a α-approximation algorithm on a point set $P \in \mathbb{R}^d$ that returns a set $C = \{c_1, \cdots, c_k\} \subset \mathbb{R}^d$ of k centers. Let C_i be the set of points that are in the cluster of center c_i, i.e. $C_i = \{p \in P : d(p, c_i) \leq \min_{c_j \in C} d(p, c_j)\}$. We consider $\log n + 1$ balls $Ball_{i,j}(c_i, 2^j \cdot \frac{\text{COST}(P,C)}{n})$ centered at center c_i of radii $2^j \cdot \frac{\text{COST}(P,C)}{n}$ for $j \in \{0, 1, 2, \cdots, \log n\}$. In ring $Ball_{i,j} \backslash Ball_{i,j-1}$ having $n_{i,j}$ points, we take a sample set $S_{i,j}$ of size $s = \min(n_{i,j}, \epsilon^{-2} dk \log(\frac{k \log n}{\epsilon\delta})$ points uniformly at random and we assign a weight of $n_{i,j}/s$ to each sampled point. We replace all points in $Ball_{i,j}$ with weighted set $S_{i,j}$. Let $\mathcal{K}_P$ be the union set of weighted sampled sets returned by this coreset technique, that is $\mathcal{K}_P = \cup_{i,j} S_{i,j}$. We also let partition Λ_P to be the set of rings $\Lambda_P = \{Ball_{i,j} \backslash Ball_{i,j-1} : i \in [k], j \in [\log n + 1]\}$.

In Step 3 of Algorithm 3 for each ring $Ball_{i,j} \backslash Ball_{i,j-1}$ having $n_{i,j}$ points we let $s_{CC} = f(n, d, \epsilon, \delta)$ be $\min(n_{i,j}, \epsilon^{-2} dk \log(\frac{k \log n}{\epsilon\delta})$. Moreover, we let parameter s in Lemma 20 (which is the size of this coreset maintained by merge-and-reduce approach) be $O(k^2 d \epsilon^{-2} \log^8 n)$. We now prove a variant of Lemma 17 for this coreset. Observe that to adjust the parameter ϵ in Lemma 17 for Lemma 27, we replace ϵ by $\frac{\epsilon^2}{5\sqrt{d}}$.

Lemma 28 *Let* $P \subset \mathbb{R}^d$ *be a point set of size* n *and* $k \in \mathbb{N}$ *be a parameter. Let* $\Lambda_P = \{Ball_{i,j} : i \in [k], j \in [\log n + 1]\}$ *be the partition set of balls returned by the coreset technique of Chen [13]. Let*

$\mathcal{B} = \{b_1, \cdots, b_k\} \subset \mathbb{R}^d$ be an arbitrary set of k centers. Suppose for every ring $Ball_{i,j} \backslash Ball_{i,j-1} \in \Lambda_P$ having $n_{i,j}$ points, we delete up to $\frac{\epsilon^2}{\sqrt{d}} \cdot n_{i,j}$ points and let $(Ball_{i,j} \backslash Ball_{i,j-1})'$ be this ball after deletion of these points. We then have

$$\sum_{Ball_{i,j} \backslash Ball_{i,j-1} \in \Lambda_P} |\mathit{COST}((Ball_{i,j} \backslash Ball_{i,j-1})', \mathcal{B}) - \mathit{COST}(Ball_{i,j} \backslash Ball_{i,j-1}, \mathcal{B})| \leq \epsilon \cdot \mathit{COST}(P, \mathcal{B}) \ ,$$

where $\mathit{COST}(Ball_{i,j} \backslash Ball_{i,j-1}, \mathcal{B}) = \sum_{p \in Ball_{i,j} \backslash Ball_{i,j-1}} d(p, \mathcal{B})$ and $\mathit{COST}((Ball_{i,j} \backslash Ball_{i,j-1})', \mathcal{B}) = \sum_{p \in (Ball_{i,j} \backslash Ball_{i,j-1})'} d(p, \mathcal{B})$.

Proof : The proof is in the same spirit of the proof of Lemma 27. Let us fix a ring $\mathrm{Ball}_{i,j} \backslash \mathrm{Ball}_{i,j-1} \in \Lambda_P$ for $i \in [k]$ and $j \in \{0, 1, \cdots, \log n\}$. Observe that the radius of $\mathrm{Ball}_{i,j}$ is $2^j \cdot \frac{\mathrm{COST}(P,C)}{n}$ and the radius of $\mathrm{Ball}_{i,j-1}$ is $2^{j-1} \cdot \frac{\mathrm{COST}(P,C)}{n}$. For the simplicity let us denote $\mathrm{Ball}_{i,j} \backslash \mathrm{Ball}_{i,j-1}$ by $R_{i,j}$ and we let $\ell_{R_{i,j}} = 2^j \cdot \frac{\mathrm{COST}(P,C)}{n}$ and let $n_{R_{i,j}}$ be the number of points in the ring $R_{i,j}$. Assume the nearest center to ring $R_{i,j}$ is $b_{R_{i,j}} \in \mathcal{B}$. We either have $d(R_{i,j}, b_{R_{i,j}}) \geq \frac{\ell_{R_{i,j}}}{\epsilon}$ or $d(R_{i,j}, b_{R_{i,j}}) < \frac{\ell_{R_{i,j}}}{\epsilon}$. If we have $d(R_{i,j}, b_{R_{i,j}}) \geq \frac{\ell_{R_{i,j}}}{\epsilon}$, then $\mathrm{COST}(R_{i,j}, \mathcal{B}) \geq n_{R_{i,j}} \cdot d(R_{i,j}, b_{R_{i,j}}) \geq n_{R_{i,j}} \cdot \frac{\ell_{R_{i,j}}}{\epsilon}$. On the other, since we delete up to $\frac{\epsilon^2}{2} \cdot n_{R_{i,j}}$ points of cell $R_{i,j}$, the cost that we lose is at most

$$\frac{\epsilon^2}{2} \cdot n_{R_{i,j}} (2\ell_{R_{i,j}} + d(R_{i,j}, b_{R_{i,j}})) \leq \frac{\epsilon^2}{2} \cdot n_{R_{i,j}} \cdot (2\epsilon + 1) \cdot d(R_{i,j}, b_{R_{i,j}}) \leq \epsilon \cdot \mathrm{COST}(R_{i,j}, \mathcal{B}) \ .$$

Now suppose we have $d(R_{i,j}, b_{R_{i,j}}) < \frac{\ell_{R_{i,j}}}{\epsilon}$. We have two cases, either $j = 0$ or $j > 0$. We first prove the lemma when $j = 0$. For $j = 0$, ring $R_{i,0}$ is in fact ball $\mathrm{Ball}_{i,0}$ of radius $\ell_{R_{i,0}} = \frac{\mathrm{COST}(P,C)}{n}$. Since cell c is in $G_{i,0}$, we have $\ell_c = \frac{\epsilon}{10\sqrt{d}\alpha} \cdot \frac{\mathrm{COST}(P,C)}{n}$. Therefore, $d(R_{i,0}, b_{R_{i,0}}) < \frac{\ell_{R_{i,0}}}{\epsilon} \leq \frac{1}{\epsilon} \cdot \frac{\mathrm{COST}(P,C)}{n}$. We delete up to $\frac{\epsilon^2}{2} \cdot n_{R_{i,0}}$ points from $R_{i,0}$. Therefore, the cost that we lose is at most

$$\frac{\epsilon^2}{2} \cdot n_{R_{i,0}} \cdot d(R_{i,0}, b_{R_{i,0}}) \leq \frac{\epsilon^2}{2} \cdot n_{R_{i,0}} \cdot \frac{1}{\epsilon} \cdot \frac{\mathrm{COST}(P,C)}{n} \leq \frac{\epsilon}{2} \cdot n_{R_{i,0}} \cdot \frac{\mathrm{COST}(P,C)}{n} \ .$$

We have $i \in [k]$. Hence a summation over i will find the overall cost that we lose as follows.

$$\sum_{i \in [k]} \frac{\epsilon}{2} \cdot n_{R_{i,0}} \cdot \frac{\mathrm{COST}(P,C)}{n} \leq \frac{\epsilon}{2} \cdot \mathrm{COST}(P,C) \ .$$

The second case is when $j > 0$. Recall that $d(R_{i,j}, b_{R_{i,j}}) < \frac{\ell_{R_{i,j}}}{\epsilon}$. Observe that since $j > 0$, $d(R_{i,j}, c_i) \geq 2^{j-1} \cdot \frac{\mathrm{COST}(P,C)}{n}$ and $\ell_{R_{i,j}} = 2^j \cdot \frac{\mathrm{COST}(P,C)}{n} \leq 2d(R_{i,j}, c_i)$ which means $d(R_{i,j}, b_{R_{i,j}}) < \frac{\ell_{R_{i,j}}}{\epsilon} \leq \frac{2}{\epsilon} \cdot d(R_{i,j}, c_i)$. For each such ring $R_{i,j}$ we delete up to $\frac{\epsilon^2}{2} \cdot n_{R_{i,j}}$ points. Thus, taking the summation over all i and $j > 0$ yields

$$\begin{aligned}\sum_{i=1}^{k} \sum_{j>0} \sum_{R_{i,j} \in \Lambda_P} \frac{\epsilon^2}{2} \cdot n_{R_{i,j}} \cdot d(R_{i,j}, b_{R_{i,j}}) &\leq \sum_{i=1}^{k} \sum_{j>0} \sum_{R_{i,j} \in \Lambda_P} \frac{\epsilon^2}{2} \cdot n_{R_{i,j}} \cdot \frac{2}{\epsilon} \cdot d(R_{i,j}, c_i) \\ &\leq \epsilon \cdot \mathrm{COST}(P,C) \ .\end{aligned}$$

□

B.3 Proof of Lemma 18

Proof of Lemma 18: Recall that we take (k,ϵ)-coreset B_i from its children which are buckets B_{i-1} and B'_{i-1}. Observe that B_{i-1} and B'_{i-1} are also (k,ϵ)-coresets of subtrees rooted at nodes B_{i-1} and B'_{i-1} with partitions $\Lambda_{B_{i-1}}$ and $\Lambda_{B'_{i-1}}$, respectively. Similarly, we take (k,ϵ)-coreset B_{i-1} from its children which are buckets B_{i-2} and B'_{i-2} that are, in turn, (k,ϵ)-coresets of subtrees rooted at nodes B_{i-2} and B'_{i-2}. We let $O_{i-1} = B_{i-2} \cup B'_{i-2}$. Let us fix arbitrary regions $R_{i-1} \in \Lambda_{B_{i-1}}$ and $R_i \in \Lambda_{B_i}$ such that $R_{i-1} \cap R_i \neq \emptyset$.

Recall that we take a sample set of size $r_{i-1} = \min\big(|O_{i-1} \cap R_{i-1}|, \max\big(s_{\mathcal{CC}}, O(d_{VC}\epsilon_{i-1}^{-2}\log(n)\cdot\log(\frac{d_{VC}\log(n)}{\epsilon_{i-1}\delta}))\big)\big)$ points uniformly at random from $O_{i-1} \cap R_{i-1}$, assign a weight of $(|O_{i-1} \cap R_{i-1}|)/r_{i-1}$ to every sampled point, and we add the weighted sampled points to B_{i-1}. Here, $s_{\mathcal{CC}}$ is from Algorithm UNIFIED. This essentially means, $B_{i-1} \cap R_{i-1}$ is an ϵ_{i-1}-sample for $O_{i-1} \cap R_{i-1}$. Since we treat a weighted point p having weight w_p as w_p points at coordinates of p, we have $|B_{i-1} \cap R_{i-1}| = |O_{i-1} \cap R_{i-1}|$. Thus,

$$\big||(O_{i-1} \cap R_{i-1}) \cap R_i| - |(B_{i-1} \cap R_{i-1}) \cap R_i|\big| \leq \epsilon_{i-1} \cdot |O_{i-1} \cap R_{i-1}| \ .$$

Observe that for regions R_i and R_{i-1}, using Lemma 26, $(P, R_i \cap R_{i-1})$ is a range space of dimension $O(2d_{VC}\log(2d_{VC}))$. So, we can write

$$\big||O_{i-1} \cap (R_{i-1} \cap R_i)| - |B_{i-1} \cap (R_{i-1} \cap R_i)|\big| \leq \epsilon_{i-1} \cdot |O_{i-1} \cap R_{i-1}| \ .$$

Now let us expand $O_{i-1} \cap (R_{i-1} \cap R_i)$ where we use $O_{i-1} = B_{i-2} \cup B'_{i-2}$. Since B_{i-2} and B'_{i-2} are disjoint, we then have $(B_{i-2} \cup B'_{i-2}) \cap (R_{i-1} \cap R_i) = (B_{i-2} \cap (R_{i-1} \cap R_i)) \cup (B'_{i-2} \cap (R_{i-1} \cap R_i))$.

Similarly, let us consider (k,ϵ)-coreset B_{i-2} with its partition $\Lambda_{B_{i-2}}$ which is a (k,ϵ)-coreset of its children, i.e., buckets B_{i-3} and B'_{i-3}. Again, let $O_{i-2} = B_{i-3} \cup B'_{i-3}$. Let us fix an arbitrary region $R_{i-2} \in \Lambda_{B_{i-2}}$ such that $R_{i-2} \cap R_{i-1} \cap R_i \neq \emptyset$. Again, we take a sample set of size $r_{i-2} = \min\big(|O_{i-2} \cap R_{i-2}|, \max\big(s_{\mathcal{CC}}, O(d_{VC}\epsilon_{i-2}^{-2}\log(n)\cdot\log(\frac{d_{VC}\log(n)}{\epsilon_{i-2}\delta}))\big)\big)$ points uniformly at random from $O_{i-2} \cap R_{i-2}$, assign a weight of $(|O_{i-2} \cap R_{i-2}|)/r_{i-2}$ to every sampled point and we add the weighted sampled points to B_{i-2}. Since $B_{i-2} \cap R_{i-2}$ is an ϵ_{i-2}-sample for $O_{i-2} \cap R_{i-2}$ and $|B_{i-2} \cap R_{i-2}| = |O_{i-2} \cap R_{i-2}|$, we then have

$$\big||(O_{i-2} \cap R_{i-2}) \cap (R_{i-1} \cap R_i)| - |(B_{i-2} \cap R_{i-2}) \cap (R_{i-2} \cap R_i)|\big| \leq \epsilon_{i-2} \cdot (|O_{i-2} \cap R_{i-2}|) \ .$$

Observe that for regions R_i, R_{i-1} and R_{i-2}, using Lemma 26, $(P, R_i \cap R_{i-1} \cap R_{i-2})$ is a range space of dimension $O(3d_{VC}\log(3d_{VC}))$. So, we can write

$$\big||O_{i-2} \cap (R_{i-2} \cap R_{i-1} \cap R_i)| - |B_{i-2} \cap (R_{i-2} \cap R_{i-2} \cap R_i)|\big| \leq \epsilon_{i-2} \cdot (|O_{i-2} \cap R_{i-2}|) \ .$$

We do the same for B'_{i-2}. We define O'_{i-2} for B'_{i-2} similar to O_{i-2}. Using triangle inequality we have

$$\Big| \sum_{R_{i-2} \in \Lambda_{B_{i-2}}} |O_{i-2} \cap (R_{i-2} \cap R_{i-1} \cap R_i)| + \sum_{R'_{i-2} \in \Lambda_{B'_{i-2}}} |O'_{i-2} \cap (R'_{i-2} \cap R_{i-1} \cap R_i)| - |B_{i-1} \cap (R_{i-1} \cap R_i)|\Big|$$
$$\leq \epsilon_{i-2} \cdot \Big(\sum_{R_{i-2} \in \Lambda_{B_{i-2}}} (|O_{i-2} \cap R_{i-2}|) + \sum_{R'_{i-2} \in \Lambda_{B'_{i-2}}} (|O'_{i-2} \cap R'_{i-2}|)\Big) + \epsilon_{i-1} \cdot |O_{i-1} \cap R_{i-1}| \ .$$

Now we recurse from level $i-2$ down to level 2 in which we have (k,ϵ)-coreset B_2 with partition Λ_{B_2} which is a (k,ϵ)-coreset of its children, i.e., buckets B_1 and B'_1. Again, let $O_2 = B_1 \cup B'_1$. Let us fix an arbitrary region $R_2 \in \Lambda_{B_2}$ such that $R_2 \cap \cdots \cap R_i \neq \emptyset$. Once again, we take a sample set of size $r_2 = \min\big(|O_2 \cap R_2|, \max\big(s_{\mathcal{CC}}, O(d_{VC}\epsilon_2^{-2}\log(n)\cdot\log(\frac{d_{VC}\log(n)}{\epsilon_2\delta}))\big)\big)$ points uniformly at random from

$O_2 \cap R_2$, assign a weight of $\frac{|O_2 \cap R_2|}{r_2}$ to every sampled point, and we add the weighted sampled points to B_2. Since $B_2 \cap R_2$ is an ϵ_2-sample for $O_2 \cap R_2$ and $|B_2 \cap R_2| = |O_2 \cap R_2|$, we then have

$$\left| |(O_2 \cap R_2) \cap (R_3 \cap \cdots \cap R_i)| - |(B_2 \cap R_2) \cap (R_3 \cap \cdots \cap R_i)| \right| \leq \epsilon_2 \cdot (|O_2 \cap R_2|) \ .$$

Observe that for regions $R_i, R_{i-1}, \cdots R_2$ using Lemma 26, $(P, R_i \cap \cdots \cap R_2)$ is a range space of dimension $O((i-1)d_{VC} \log((i-1)d_{VC}))$. So, we can write

$$\left| |O_2 \cap (R_2 \cap \cdots \cap R_i)| - |B_2 \cap (R_2 \cap \cdots \cap R_i)| \right| \leq \epsilon_2 \cdot (|O_2 \cap R_2|) \ .$$

By repeated applications of the triangle inequality for levels $i-2$ down to 2 we obtain

$$\begin{aligned}
&\Big| \sum_{\text{node } x_2 \text{ in level } 2} |O_{x_2} \cap (R_{i-1} \cap R_i)| - |B_{i-1} \cap (R_{i-1} \cap R_i)| \Big| \\
&= \Big| \sum_{x_2 \text{ in level } 2} \sum_{R_2^{x_2} \in \Lambda_{x_2}} \sum_{R_3^{x_3} \in \Lambda_{x_3}} \cdots \sum_{R_{i-2}^{x_{i-2}} \in \Lambda_{x_{i-2}}} |O_{x_2} \cap (R_2^{x_2} \cap R_3^{x_3} \cap \cdots \cap R_{i-2}^{x_{i-2}}) \cap R_{i-1} \cap R_i| - |B_{i-1} \cap (R_{i-1} \cap R_i)| \Big| \\
&\leq \sum_{\text{level } j=2}^{i-2} \sum_{\text{node } x_j \text{ in level } j} \epsilon_j \cdot \Big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \Big) + \epsilon_{i-1} \cdot |O_{i-1} \cap R_{i-1}| \ ,
\end{aligned}$$

where x_2 is a child of node x_3, x_3 is a child of node x_4, and so on, and O_{x_j} is the point set at node x_j. Observe that in level 1 (i.e., leaf level) we do not merge buckets and merging buckets starts at level 2, because of that the index of the first sum start with $j = 2$. We take sums $\sum_{R_{i-1} \in \Lambda_{B_{i-1}}}$ and $\sum_{R'_{i-1} \in \Lambda_{B'_{i-1}}}$ to conclude

$$\left| |P_i \cap R_i| - |B_i \cap R_i| \right| \leq \sum_{\text{level } j=2}^{i-1} \sum_{\text{node } x_j \text{ in level } j} \epsilon_j \Big(\sum_{R \in \Lambda_{x_j}} (|O_{x_j} \cap R|) \Big) \ .$$

In Algorithm 3 we simply replace VC-dimension d_{VC} by $O(d_{VC} \log n)$ for all levels $i \in \log n$. □

B.4 Proof of Lemma 20

Proof of Theorem 20: According to Algorithm SWCoreset, the next index that we keep in sequence X occurs when ϵ-fraction of a region $R \in \Lambda_{\mathcal{K}_{x_j}}$ changes. Since s is the size of (k, ϵ)-coreset that Algorithm MergeReduce maintains for P, the upper bound on the number of regions in partition $\Lambda_{\mathcal{K}_{x_j}}$ is also s. By Lemma 17, as long as at most ϵ-fraction of the weight of a region in partition $\Lambda_{\mathcal{K}_{x_j}}$ drops, we still have a (k, ϵ)-coreset. Thus, after at most s/ϵ indices, the optimal clustering cost drops by at least ϵ-fraction of its cost. Therefore, after $O(\log_{1+\epsilon} n) = O(\epsilon^{-1} \log n)$ of this sequence of ϵ-fraction drops in the optimal clustering cost, the cost converges to zero. Overall, the number of indices that we maintain is $O(s\epsilon^{-2} \log n)$. Moreover, for each index we maintain a (k, ϵ)-coreset of size s using Algorithm MergeReduce; therefore, the space usage of our algorithm is $O(s^2 \epsilon^{-2} \log n)$. □