View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

# Improved Algorithms for Time Decay Streams

**Vladimir Braverman**
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
vova@cs.jhu.edu

**Harry Lang**
MIT CSAIL, Cambridge, MA, USA
harry1@mit.edu

**Enayat Ullah**
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
enayat@jhu.edu

**Samson Zhou**
School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA
samsonzhou@gmail.com

## Abstract

In the time-decay model for data streams, elements of an underlying data set arrive sequentially with the recently arrived elements being more important. A common approach for handling large data sets is to maintain a *coreset*, a succinct summary of the processed data that allows approximate recovery of a predetermined query. We provide a general framework that takes any offline-coreset and gives a time-decay coreset for polynomial time decay functions.

We also consider the exponential time decay model for $k$-median clustering, where we provide a constant factor approximation algorithm that utilizes the online facility location algorithm. Our algorithm stores $O(k \log(h\Delta) + h)$ points where $h$ is the half-life of the decay function and $\Delta$ is the aspect ratio of the dataset. Our techniques extend to $k$-means clustering and $M$-estimators as well.

**2012 ACM Subject Classification** Theory of computation → Facility location and clustering; Theory of computation → Streaming, sublinear and near linear time algorithms

**Keywords and phrases** Streaming algorithms, approximation algorithms, facility location and clustering

**Digital Object Identifier** 10.4230/LIPIcs.APPROX-RANDOM.2019.27

**Category** APPROX

**Acknowledgements** 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.

## 1 Introduction

The *streaming model* of computation has become an increasingly popular model for processing massive datasets. In this model, the data is presented sequentially, and the objective is to answer some pre-defined query. The overwhelmingly large size of the dataset imposes a number of restrictions on any algorithm designed to answer the pre-defined query. For example, a streaming algorithm is permitted only a few passes, or in many cases, only a single pass over the data. Moreover, the algorithm should also use space sublinear in, or even logarithmic in, the size of the data. For more details on the background and applications of the streaming model, [4, 45, 1] provide excellent surveys.

© Vladimir Braverman, Harry Lang, Enayat Ullah, and Samson Zhou;
licensed under Creative Commons License CC-BY
Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019).
Editors: Dimitris Achlioptas and László A. Végh; Article No. 27; pp. 27:1–27:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Informally, a coreset for a given problem is a small *summary* of the dataset such that the *cost* of any candidate solution on the coreset is approximately the same as the cost in the original set. Coresets have been used in a variety of problems, including generalized facility locations [29], $k$-means clustering [31, 9], principal component analysis [33], and $\ell_p$-regression [24]. Coresets also have a number of applications in distributed models (see [39, 44, 6, 3], for example). To maintain the coresets throughout the data stream, one possible approach is the so called merge-and-reduce method, in which the multiple sets may be adjusted and combined. Several well-known coreset constructions [37, 17] for the $k$-median and $k$-means problems are based on the merge-and-reduce paradigm.

## 1.1 Motivation

Many applications discard obsolete data, choosing to favor relatively recent data to base their queries. This motivates the *time decay* model, in which there exists a function $w$ so that the weight of the $t^{\text{th}}$ most recent item is $w(t)$. Note that this is a generalization of both the *insertion-only* streaming model, where $w(t) = 1$ for all $t$, and the *sliding-window* model, where $w(t) = 1$ for the most recent $W$ items, and $w(t) = 0$ for $t > W$. In this paper, we study the problem of maintaining coresets over a polynomial decay model, where $w(t) = \frac{1}{t^s}$ for some parameter $s > 0$, and an exponential decay model, where $w(t) = 2^{\frac{T-t+1}{h}}$ at time $T$ for some *half-life* parameter $h > 0$.

Although exponential decay model is well-motivated by natural phenomena that exhibit half-life behavior, [20] notices that exponential decay and the sliding window model is often insufficient for many applications because the decay occurs too quickly and suggests that polynomial decay may be a reasonable alternative for some applications, such as availability of network links. For example, consider a network link that fails at every time between 10 and 60 and a second network link that fails once at time 75. Intuitively, it seems like the second link should be better, but under many parameters, the exponential decay model and sliding window model will both agree that the first link is better. Fortunately, under the polynomial decay model, events that occur near the same time have approximately the same weight, and we will obtain some view in which the first link is preferred [40]. In practice, time decay functions have been used in natural language understanding to give more importance to recent utterances than the past ones [47].

**Organization.** The rest of the paper is organized as follows. In Section 2, we summarize the main results of the paper and the algorithmic approaches. In Section 3, we discuss the related work, and in Section 4, we formalize the problem and discuss the preliminaries required. In Sections 5 and 6, we handle the polynomial and exponential decay, respectively, in detail, wherein we present the algorithmic details as well as the complete analysis.

# 2 Our Contributions

We summarize our results and give a high-level idea of our approach for problems in the polynomial and exponential decay models in the following subsections respectively. The reader is encouraged to go through Sections 5 and 6 for details.

## 2.1 Polynomial decay

In the polynomial decay model, a stream of points $P$ arrives sequentially and the weight of the $t^{\text{th}}$ most recent point, denoted as $w(t)$, is $w(t) = \frac{1}{t^s}$ where $s > 0$ is a given constant parameter of the decay function. We first state a theorem that shows that we can use an offline coreset construction mechanism to give a coreset for the polynomial decay model.

▶ **Theorem 1.** *Given an algorithm that takes a set of $n$ points as input and constructs an $\epsilon$-coreset of $F(n,\epsilon)$ points in $\mathcal{O}(nT(\epsilon))$ time, there exists a polynomial decay algorithm that maintains an $\epsilon$-coreset while storing $\mathcal{O}\left(\epsilon^{-1}\log n\, F\left(n, \frac{\epsilon}{\log n}\right)\right)$ points and with update time $\mathcal{O}\left(\epsilon^{-1}\ \log n\ F(n,\epsilon)\, T(\epsilon/\log n)\right)$.*

Theorem 1 applies to any time-decay problem on data streams that admits an approximation algorithm using coresets. Among its applications are the problems of $k$-median and $k$-means clustering, $M$-estimator clustering, projective clustering, and subspace approximation. We list a few of these results in Table 1. Our result is a generalization of the *vanilla* merge-and-reduce approach used to convert offline coresets to streaming counterparts. In particular, plugging in $s = 0$, we get the vanilla streaming model, and the theorem recovers the corresponding guarantees.

**Table 1** Coresets for some problems in polynomial decay streams.

| Problem | Coreset size | Offline algorithm |
|---|---|---|
| Metric $k$-median clustering | $\mathcal{O}\left(\frac{s}{\epsilon^3} k \log k \log^4 n\right)$ | [30] |
| Metric $k$-means clustering | $\mathcal{O}\left(\frac{s}{\epsilon^3} k \log k \log^4 n\right)$ | [9] |
| Metric $M$-estimator | $\mathcal{O}\left(\frac{s}{\epsilon^3} k \log k \log^4 n\right)$ | [9] |
| $j^{\text{th}}$ subspace approximation | $\mathcal{O}\left(\frac{j^2 s}{\epsilon^4} \log^8 n \log\left(\frac{\log n}{\epsilon}\right)\right)$ | [30] |
| Low rank approximation | $\mathcal{O}\left(\frac{s}{\epsilon^2} kd \log n\right)$ | [34] |

**Approach.** A natural starting point would be to attempt to generalize existing sliding window algorithms to time decay models. These algorithms typically use a histogram data structure [14], in which multiple instances of streaming algorithms are started at various points in time, one of which well-approximates the objective evaluated on the data set represented by the sliding window. However, generalizing these histogram data structures to time-decay models does not seem to work since the weights of all data points changes upon each new update in time-decay model, whereas streaming algorithms typically assume static weights for each data point.

Instead, our algorithm partitions the stream into blocks, where each block represents a disjoint collection of data point between certain time points. Each arriving element initially begins as its own block, containing one element. The algorithm maintains an unweighted coreset for each block, and merges blocks (i.e corresponding coresets) as they become older. However, at the end, each block is to be weighted according to some function, and so the algorithm chooses to merge blocks when the weights of the blocks become "close". Thus, a coreset for each block will represent the set of points well, as the weights of the points in each block do not differ by too much.

## 2.2 Exponential decay

We also provide an algorithm that achieves a constant approximation for $k$-median clustering in the exponential decay model. Our guarantees also extend to $k$-means clustering and $M$-estimators.

Given a set $P$ of points in a metric space, let $\Delta$ denote its aspect ratio i.e the ratio between the largest and (non-zero) smallest distance between any two points in $P$. The weight of the $t^{\text{th}}$ most recent point at time $T$ is $w(t) = 2^{\frac{T-t+1}{h}}$ where $h > 0$ is the half-life parameter of the exponential decay function.

▶ **Theorem 2.** *There exists a streaming algorithm that given a stream $P$ of points with exponentially decaying weights, with aspect ratio $\Delta$ and half-life $h$, produces an $\mathcal{O}(1)$-approximate solution to $k$-median clustering. The algorithm runs in $\mathcal{O}(nk\log(h\Delta))$ time and uses $\mathcal{O}(k\log(h\Delta)+h)$ space.*

**Approach.** Although our previous framework will work for other decay models, the algorithm may use prohibitively large space. The intuition behind the polynomial decay approach is that a separate coreset is maintained for each set of points that roughly have the same weight. In other words, the previous framework maintains a separate coreset each time the weight of the points decrease by some constant amount, so that if $R$ is the ratio between the largest weight and the smallest weight, then the total number of coresets stored by the algorithm is roughly $\log R$. In the polynomial decay model, the number of stored coresets is $\mathcal{O}(\log n)$, but in the exponential decay model, the number of stored coresets would be $\mathcal{O}(n)$, which would no longer be sublinear in the size of the input. Hence, we require a new approach for the exponential decay model.

Instead, we use the online facility location (OFL) algorithm of Meyerson [43] as a subroutine to solve $k$-median clustering in the exponential decay model. In the online facility location problem, we are given a metric space along with a facility cost for each point/location that appears in the data stream. The objective is to choose a (small) number of facility locations to minimize the total facility cost plus the service cost, where the service cost of a point is its distance to the closest facility. For more details, please see Section 6.

Our algorithm for the exponential time decay model proceeds on the data stream, working in phases. Each phase corresponds to an increasing "guess" for the value of the *cost* of the optimal clustering. Using this guess, each phase queries the corresponding instance of OFL. If the guess is *correct*, then the subroutine selects a bounded number of facilities. On the other hand, if either the cost or the number of selected facilities surpasses a certain quantity, then the guess for the optimal cost must be incorrect, and the algorithm triggers a phase change. Upon a phase change, our algorithm uses an offline $k$-median clustering algorithm to cluster the facility set and produces exactly $k$ points. It then runs a new instance of OFL with a larger guess, and continues processing the data stream.

However, there is a slight subtlety in this analysis. The number of points stored by OFL is dependent on the weights of the point. In an exponential decay function, the ratio between the largest weight and smallest weight of points in the data set may be exponentially large. Thus to avoid OFL from keeping more than a logarithmic number of points, we force OFL to terminate after seeing $\log(h\Delta)$ points during a phase. Furthermore, we store points verbatim until we see $k+h$ *distinct* points, upon whence we will trigger a phase change. We show that forcing this phase change does indeed correspond with an increase in the guess of the value for the optimal cost.

## 3 Related Work

The first insertion-only streaming algorithm for the $k$-median clustering problem was presented in 2000 by Guha, Mishra, Motwani, and O'Callaghan [36]. Their algorithm uses $\mathcal{O}(n^{\epsilon})$ space for a $2^{\mathcal{O}(1/\epsilon)}$ approximation, for some $0<\epsilon<1$. Subsequently, Charikar *et al* [16] present an $\mathcal{O}(1)$-approximation algorithm for $k$-means clustering that uses $\mathcal{O}\left(k\log^2 n\right)$ space. Their algorithm uses a number of phases, each corresponding to a different guess for the value of the cost of optimal solution. The guesses are then used in the online facility location (OFL) algorithm of [43], which provides a set of centers whose number and cost allows the algorithm

to reject or accept the guess. This technique is now one of the standard approaches for handling $k$-service problems. Braverman *et al* [13] improve the space usage of this technique to $\mathcal{O}(k \log n)$. [11] and [12] develop algorithms for $k$-means clustering on sliding windows, in which expired data should not be included in determining the cost of a solution.

Another line of approach for $k$-service problems is the construction of coresets, in particular when the data points lie in the Euclidean space. Har-Peled and Mazumdar [37] give an insertion-only streaming algorithm for $k$-medians and $k$-means that provides a $(1+\epsilon)$-approximation, using space $\mathcal{O}\left(k\epsilon^{-d}\log^{2d+2} n\right)$, where $d$ is the dimension of the space. Similarly, Chen [17] introduced an algorithm using $\mathcal{O}\left(k^2 d\epsilon^{-2}\log^8 n\right)$ space, with the same approximation guarantees.

Cohen and Strauss [20] study problems in time-decaying data streams in 2003. There are a number of results [40, 22, 21, 23] in this line of work, but the most prominent time-decay model is the sliding window model. Datar *et al* [25] introduced the exponential histogram as a framework in the sliding window for estimating statistics such as count, sum of positive integers, average, and $\ell_p$ norms. This initiated an active line of research, including improvements to count and sum [35], frequent itemsets [18, 10], frequency counts and quantiles [2, 42], rarity and similarity [26], variance and $k$-medians [5] and other geometric and numerical linear algebra problems [28, 15, 8].

## 4 Preliminaries

Let $\mathcal{X}$ be the set of possible points in a space with metric $d$. A weighted set is a pair $(P, w)$ with a set $P \subset \mathcal{X}$ and a weight function $w : P \to [0, \infty)$. A query space is a tuple $(P, w, f, Q)$ that combines a weighted set with a set $Q$ of possible queries and a function $f : \mathcal{X} \times Q \to [0, \infty)$. A query space induces a function

$$\bar{f}(P, w, q) = \sum_{p \in P} w(p) f(p, q).$$

We now instantiate the above with some simple examples.

▶ **Example 3** ($k$-means)**.** Let $Q$ be all sets of $k$ points in $\mathbb{R}^d$, and for $C \in Q$ define $f(p, C) = \min_{c \in C} d^2(p, c)$. The $k$-means cost of $(P, w)$ to $C$ is

$$\sum_{p \in P} w(p) \min_{c \in C} d^2(p, c).$$

▶ **Example 4** ($k$-median)**.** Let $Q$ be all sets of $k$ points in $\mathbb{R}^d$, and for $C \in Q$ define $f(p, C) = \min_{c \in C} d(p, c)$. The $k$-median cost of $(P, w)$ to $C$ is

$$\sum_{p \in P} w(p) \min_{c \in C} d(p, c).$$

Note that both $k$-median and $k$-means are captured in $\bar{f}(P, w, C)$. We now define an $\epsilon$-coreset.

▶ **Definition 5** ($\epsilon$-coreset)**.** *A $\epsilon$-coreset for the query space $(P, w, f, Q)$ is a tuple $(Z, u)$, where $Z \subseteq \mathcal{X}$ is a set of points and $u : Z \to [0, \infty)$ are their corresponding weights, such that for every $q \in Q$*

$$(1-\epsilon)\bar{f}(P, w, q) \le \bar{f}(Z, u, q) \le (1+\epsilon)\bar{f}(P, w, q).$$

An important property of coresets is that they are *closed* under operations like union and composition. We formalize this below.

▶ **Proposition 6** (Merge-and-reduce, [17]). *Coresets satisfy the following two properties.*

1. *If $\mathcal{S}_1$ and $\mathcal{S}_2$ are $\epsilon$-coresets of disjoint sets $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, then $\mathcal{S}_1 \cup \mathcal{S}_2$ is an $\epsilon$-coreset of $\mathcal{P}_1 \cup \mathcal{P}_2$.*
2. *If $\mathcal{S}_1$ is an $\epsilon$-coreset of $\mathcal{S}_2$ and $\mathcal{S}_2$ is a $\delta$-coreset of $\mathcal{S}_3$, then $\mathcal{S}_1$ is a $((1+\epsilon)(1+\delta)-1)$-coreset of $\mathcal{S}_3$.*

We now define approximate triangle inequality, a property that allows us to extend our results obtained in metric spaces to ones with *semi-distance* functions. In particular, this allows us to extend results for $k$-median clustering to $k$-means and $M$-estimators in exponential decay streams.

▶ **Definition 7** ($\lambda$-approximate triangle inequality). *A function $d(\cdot,\cdot)$ on a space $\mathcal{X}$ satisfies the $\lambda$-approximate triangle inequality if for all $x, y, z \in \mathcal{X}$,*

$$d(x,z) \leq \lambda(d(x,y) + d(y,z)).$$

## 5 Polynomial Decay

We consider a time decay, wherein a point $p$ in the stream, which arrived at time $t$, has weight $w(p) = (T-t+1)^{-s}$ at time $T > t$, for some parameter $s > 0$. Equivalently, the $t^{\text{th}}$ most recent element has weight $t^{-s}$ for some $s > 0$.

We present a general framework which, for given problem, takes an offline coreset construction algorithm and adapts it to polynomial decay streams. Our technique can be viewed as a generalization of merge-and-reduce technique of Bentley and Saxe [7]. We also briefly discuss some applications towards that end. We start with stating our main theorem for polynomial decay streams.

▶ **Theorem 8.** *Given an offline algorithm that takes a set of $n$ points as input and constructs an $\epsilon$-coreset of $F(n,\epsilon)$ points in $\mathcal{O}(n\,T(\epsilon))$ time, there exists a polynomial decay algorithm that maintains an $\epsilon$-coreset while storing $\mathcal{O}\left(\epsilon^{-1} s \log n\, F(n, \epsilon/\log n)\right)$ points and with update time*

$$\mathcal{O}\left(\epsilon^{-1} s\ \log n\ F(n,\epsilon)\, T(\epsilon/\log n)\right).$$

**Notation.** We use $\mathbb{N}$ to denote the set of natural numbers. We use CS-RAM to denote an offline coreset construction algorithm, which given $n$ points, constructs an $\epsilon$-coreset in time $\mathcal{O}(n\,T(\epsilon))$ and takes space $F(n,\epsilon)$. We abuse notation by using $F(n,\epsilon)$ to also refer to the corresponding coreset.

### 5.1 Algorithm

We start with giving a high-level intuition of the algorithm. Given a stream of points, the algorithm implicitly maintains a partition of the streams into disjoint *blocks*. A block is a collection of consecutive points in the stream, and is represented by two positive integers $a$ and $b$ as $[a,b]$, where $a$ represents the position of the first point in the block and $b$ the last point, relative to the start of the stream. Let the set of blocks be denoted by $\mathcal{B}$. Our algorithm stores points of a given block by maintaining a coreset for the points in that block. As the stream progresses, we merge older blocks i.e. the corresponding coresets. Informally, the merge happens when the weights of the blocks become *close*.

We first define a set of integer *markers* $x_i$, which for a given $i \in \mathbb{N}$, depends on the decay parameter $s$ and target $\epsilon$. These markers dictate when to merge blocks as the stream progresses. For a given $i \in \mathbb{N}$, we define $x_i$ to be the minimum integer greater than or equal to $2^i$ such that

$$\frac{1-\epsilon}{(x_i - 2^i + 1)^s} \leq \frac{1+\epsilon}{x_i^s}.$$

Equivalently, we can write $\left(\frac{x_i}{x_i-2^i+1}\right)^s \leq \frac{1+\epsilon}{1-\epsilon}$. Note that each of the $2^i$ points following $x_i$ in the stream, has weight within $\frac{1+\epsilon}{1-\epsilon}$ times the weight of $x_i$. Moreover, $x_i$'s can be exactly pre-computed from the equation and we therefore assume that these are implicitly stored by the algorithm. Each new element in the stream starts as a new block. As mentioned before, the blocks are represented by two integers $[a, b]$ and the points are stored as a coreset. When a block $[a, b]$ *reaches* $x_i$, then algorithm merges all of $[x_i - 2^i + 1, x_i]$ points into a single coreset. In the end, the algorithm outputs the *weighted* union of the coresets of the blocks.

To visualize this, consider the integer line, and suppose that we have $x_i$'s marked on the positive side of the line, for example $x_1 = 2, x_2 = 4 \ldots$. The tuple indices of the blocks represent the relative position of the point in the stream, with the start being 1 and the end point being $n$. At the start, the stream is on the non-positive end with the first point at 0. As the time progresses, the stream moves to the right side. Therefore, when we observe the first element, it moves to the point 1. We then store it as a new block, represented by $[1, 1]$; we also simultaneously store a coreset corresponding to it. As time progresses, a block reaches $x_i$ for some $i$ which can be formally expressed as $a + x_i \leq n$. We then merge all blocks in the range $[a, a + 2^i - 1]$. Note that by definition of $x_i$, we would have observed all these elements and also we will not merge partial blocks. We present this idea in full in Algorithm 1 and intuition in Figure 1. We remark that when we construct coresets, we use an offline algorithm CS-RAM which given a set of $n$ points $P$ and a query space $(P, w, f, q)$ produces an $\epsilon$-coreset.

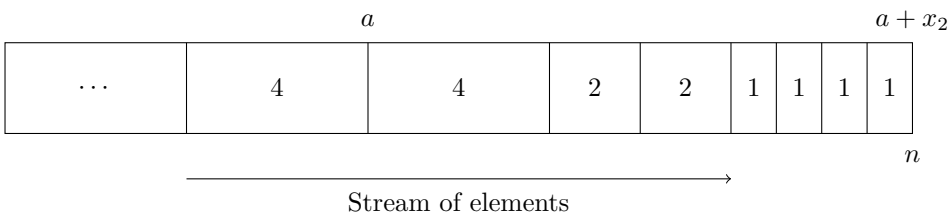


**Figure 1** The algorithm merges blocks in each interval $[a, a + 2^i - i]$ for $a \leq n - x_i$.

**Algorithm 1** $\epsilon$-coreset for polynomial decaying streams.

**Input:** Stream $P$, polynomial decay function $w(t) = \frac{1}{t^s}$, for some $s > 0$, an offline coreset construction algorithm CS-RAM
**Output:** $(1 + \epsilon)$ coreset.

```
Initialize B = ∅
for each element p_n of the stream do
    Insert [n, n] into B as a new block and construct a coreset
    for each block [a, b] ∈ B do
        if a + x_i < n for some i then
            Merge the blocks in [a, a + 2^i − 1] and reduce to get an ε/(3 log n)-coreset
        end if
    end for
end for
for each block [a, b] ∈ B do
    Give the block weight u(a, b) = 1/2 (1−ε/a^s + 1+ε/b^s)
end for
```

## 5.2 Analysis

We first show that a weighted combination of blocks gives us an $\epsilon$-coreset. For a block $[a, b]$, let the weight of the block be denoted as $u(a, b)$. We set $u(a, b) = \bar{u}$ where $\bar{u}$ satisfies

$$\frac{1-\epsilon}{a^s} \leq \bar{u} \leq \frac{1+\epsilon}{b^s}.$$

The following lemma shows that any such $\bar{u}$ produces a $3\epsilon$-coreset.

▶ **Lemma 9.** *Let $(Z, u)$ be an $\epsilon$-coreset for $(P, w, f, Q)$. Let $\bar{u} : Z \to [0, \infty)$ be such that $(1-\epsilon)u(z) \leq \bar{u}(z) \leq (1+\epsilon)u(z)$ for every $z \in Z$, then $(Z, \bar{u})$ is a $3\epsilon$-coreset for $(P, w, f, Q)$.*

**Proof.** Since $(Z, u)$ is an $\epsilon$-coreset for $(P, w, f, Q)$, therefore for every $q \in Q$,

$$\begin{aligned}
&(1-\epsilon)\bar{f}(P, w, q) \leq \bar{f}(Z, u, q) \leq (1+\epsilon)\bar{f}(P, w, q) \\
\iff &(1-\epsilon)\sum_{p\in P} w(p)f(p, q) \leq \sum_{z\in Z} u(z)f(z, q) \leq (1+\epsilon)\sum_{p\in P} w(p)f(p, q) \\
\iff &(1-\epsilon)^2\sum_{p\in P} w(p)f(p, q) \leq \sum_{z\in Z} \bar{u}(z)f(z, q) \leq (1+\epsilon)^2\sum_{p\in P} w(p)f(p, q).
\end{aligned}$$

Note that for $\epsilon < 1$, we have $(1-2\epsilon)\bar{f}(P, w, q) \leq (1-\epsilon)^2\bar{f}(P, w, q) \leq \bar{f}(Z, \bar{u}, q) \leq (1+\epsilon)^2\bar{f}(P, w, q) \leq (1+3\epsilon)f(P, w, q)$. Therefore $(Z, \bar{u})$ is a $3\epsilon$-coreset for $(P, w, f, Q)$. ◀

Having assigned weights to the blocks, we can take the union to get the coreset of $\mathcal{B}$. For simplicity, we choose $u(a, b) = \frac{1}{2}\left(\frac{1-\epsilon}{a^s} + \frac{1+\epsilon}{b^s}\right)$ in Algorithm 1. We now present a lemma that bounds the number of blocks maintained by the algorithm.

▶ **Lemma 10.** *Given a polynomial decay stream of $n$ points as input to Algorithm 1, the number of blocks produced is $\mathcal{O}\left(\epsilon^{-1}s\log n\right)$.*

**Proof.** Consider any two adjacent blocks. By the definition of the $x_i$'s, the ratio between the weights of the oldest and youngest elements is at least $(1+\epsilon)/(1-\epsilon)$. In the full stream, the oldest element has weight $1/n^s$ and the youngest element has weight 1. Let $B$ be the number of blocks so that $\left(\frac{1+\epsilon}{1-\epsilon}\right)^{\lfloor B\rfloor} \leq n^s$. Solving for $B$, we get $B \leq \frac{s\log n}{\log((1+\epsilon)/(1-\epsilon))}$. We will now lower bound the denominator using the numerical inequality $\ln(1+x) \geq \frac{2x}{2+x}$ for $x > 0$; equivalently $\log(1+x) \geq c \cdot \frac{2x}{2+x}$ for $x > 0$ and $c = \Theta(1)$. We get $\log\left(\frac{1+\epsilon}{1-\epsilon}\right) = \log\left(1 + \frac{2\epsilon}{1-\epsilon}\right) \geq 2c\epsilon$, and therefore we have $B = \mathcal{O}\left(\epsilon^{-1}s\log n\right)$. ◀

We now give the proof of the main theorem for the polynomial decay model.

**Proof of Theorem 8.** From Proposition 6, we get that when we merge disjoint blocks, we do not sacrifice the coreset approximation parameter $\epsilon$. However, when we reduce, for instance two $\epsilon$-corsets, we get a $2\epsilon$-coreset. For $n$ points observed in the stream, note that there would be at most $\log n$ reduces. This follows from the fact that the size of successive blocks increase exponentially. Therefore using an offline $\epsilon'$-coreset construction algorithm CS-RAM with $\epsilon' = \epsilon/3\log n$, we get that merging and reducing the blocks produces an $\epsilon/3$-coreset (by Proposition 6). Finally, from Lemma 9, we get that taking a union of these blocks weighted by $u(a, b) = \frac{1}{2}\left(\frac{1-\epsilon}{a^s} + \frac{1+\epsilon}{b^s}\right)$ gives us an $\epsilon$-coreset.

For the space bound, we have from Lemma 10 that the number of blocks is $\mathcal{O}\left(\epsilon^{-1}s\log n\right)$. Since we maintain an $\epsilon/\log n$ coreset for each block, we get that the offline coreset construction algorithm takes space $F(n, \epsilon/\log n)$. Therefore, we get that the space complexity is

$\mathcal{O}\left(\epsilon^{-1} s \log n\, F(n, \epsilon/\log n)\right)$. For update time, note that for $n$ points, we have $\mathcal{O}\left(\epsilon^{-1} s \log n\right)$ blocks and we use an $(\epsilon/\log n)$-coreset algorithm which takes time $\mathcal{O}\left(F(n,\epsilon)\ T(\epsilon/\log n)\right)$ per block. We therefore get a total time of $\mathcal{O}\left(\epsilon^{-1} s\ \log n\ F(n,\epsilon)\, T(\epsilon/\log n)\right)$ ◀

**Applications.** Coresets have been designed for a wide variety of geometric, numerical linear algebra and learning problems. Some examples include $k$-median and $k$-means clustering [17], low rank approximation [46], $\ell_p$ regression [19], projective clustering [27], subspace approximation [32], kernel methods [48], Bayesian inference [38] etc. We instantiate our framework with a few of these problems, and present the results in Table 1.

# 6 Exponential Decay

We now discuss another model of time decay in which the weights of previous points decay exponentially with time. Analogous to our polynomial decay model, a point that first appeared in the stream at time $t \leq T$ has weight $2^{\frac{T-t+1}{h}}$ at time $T$, where the parameter $h > 0$ is the half-life of the decay function. We however consider a different viewpoint to simplify the analysis; we maintain that the weight of a point observed at time $t$ is *fixed* to be $2^{t/h}$ where $h > 0$ is the half-life parameter. These are equivalent since the ratio of weights between successive points is the same in both the models.

**Online Facility Location.** We first discuss the problem of Online Facility Location (OFL) as our algorithm uses it as a sub-routine. The problem of facility location, given a set of points $P \subseteq \mathcal{X}$, called *demands*, a distance function $d(\cdot,\cdot)$ and fixed cost $f > 0$, conventionally called the facility cost, asks to find a set of points $\mathcal{C}$ that minimizes the following objective.

$$\min_{\mathcal{C}\subseteq\mathcal{X}} \sum_{p\in P} \min_{c\in\mathcal{C}} d(p,c) + |\mathcal{C}|\, f$$

Informally, it seeks a set of points such that the *cumulative* cost of serving the demands (known as *service cost*), which is $d(p,c)$ and opening new facilities $f$, is minimized. Online Facility Location is the variant of the above problem in the streaming setting, wherein the facility assignments and service costs incurred are irrevocable. That is to say, once a point is assigned to a facility, it cannot be reassigned to a different facility at a later point in time, even if the newer facility is closer. A simple and popular algorithm to this problem is by Meyerson [43], wherein upon receiving a point, it calculates its distance to the nearest facility and flips a coin with bias equal to the distance divided by facility cost. If the outcome is heads (or 1), it opens a new facility, otherwise the nearest point serves this demand and it incurs a service cost, equal to the distance. From here on, we abuse notation and use OFLto refer to the algorithm of Meyerson [43].

## 6.1 Algorithm

Our algorithm for exponential decaying streams is a variant of the popular $k$-median clustering algorithm [13, 16], which uses OFL as a sub-routine. We first briefly discuss the algorithm of [13] and then elucidate on how we adapt this to exponential decay streams. The algorithm operates in *phases*, where in each phase it maintains a *guess*, denoted by $L$, to the lower bound on optimal cost. It then uses this guess to instantiate the OFL algorithm of [43] on a set of points in the stream. If the service cost of OFL grows high or the number of facilities grows large, it infers that the guess is too low and triggers a *phase change*. It then increases the guess by a factor of $\beta$ (to be set appropriately) and the facilities are put back at the start of the stream and another round of OFL is run.

**Notation.** We first define and explain some key quantities. The *aspect ratio* of a set is defined as the ratio between the largest distance and the smallest non-zero distance between any two points in the set. We use $\Delta$ to denote the aspect ratio of the stream $P$. For simplicity of presentation, we assume that the minimum non-zero distance between two points is at least 1. We define $W$ as the total weight of the first $h \log \Delta$ points in the stream divided by the minimum weight. Suppose the stream starts at $t = z$, then for any $h = \Omega(1)$,

$$W = \frac{1}{2^{z/h}} \sum_{t=z}^{h \log(\Delta+1)} 2^{t/h} = \frac{\Delta}{2^{1/h} - 1} = \Theta(h\Delta).$$

For a set $P \subseteq (\mathcal{X}, d)$, we use $\mathsf{OPT}_k(P)$ to denote the optimal $k$-median clustering cost for the set. For two sets $P$ and $S$, we use $\mathsf{COST}(P, S)$ to denote the cost of clustering $P$ with $S$ as medians. Whenever we use $\mathsf{OPT}$, it corresponds to the optimal cost of $k$-median clustering of the stream seen till the point in context. We use KM-RAM to denote an offline constant $c_r$-approximate $k$-median clustering algorithm in the random access model (RAM). Given a set of points $P$ and a positive integer $k$, KM-RAM outputs $(\mathcal{C}, \lambda)$, where $\mathcal{C}$ is a set of $k$ points and $\lambda = \mathsf{COST}(P, \mathcal{C}) \leq c_r \cdot \mathsf{OPT}_k(P)$.

**Our Algorithm.** Our algorithm, inspired from [16, 13], works in phases. We however have important differences. Each of our phases are again sub-divided into two *sub-phases*. In the first sub-phase we execute OFL same as [16, 13] and after each point we check if the cost or the number of facilities is too large. If this is indeed the case, we trigger a phase change. However, if we read $h \log \Delta$ points in a phase, then we move on to the second sub-phase of the algorithm. Here we simply count points and store them verbatim. Upon reading $k + h$ points, we trigger a phase change. The intuition for this sub-phase is that a phase change is triggered when $\mathsf{OPT}$ increases by a factor of $\beta$. After $h \log \Delta$ points, subsequent points are so heavy relative to points of the previous phase that any service cost will be large enough to ensure $\mathsf{OPT}$ has increased. Therefore, we restrict the algorithm to read at most $h \log \Delta + k + h$ points in a single phase. When we start a new phase, we cluster the existing facility set to extract exactly $k$ points using an off-the-shelf constant approximate KM-RAM algorithm and continue processing the stream. We present the above idea in full in Algorithm 2. We now state our main theorem for exponential decay streams.

▶ **Theorem 11.** *There exists a streaming algorithm that given a stream $P$ of exponential decaying points with aspect ratio $\Delta$ and half-life $h$, produces an $\mathcal{O}(1)$-approximate solution to $k$-median clustering. The algorithm runs in $\mathcal{O}(nk \log(h\Delta))$ time and uses $\mathcal{O}(k \log(h\Delta) + h)$ space.*

## 6.2 Analysis

We first analyze the service cost and space complexity of OFL. For the $t^{\text{th}}$ point in the stream $p_t$, the weight of $p_t$, denoted $w(p_t)$, is $w(p_t) = 2^{t/h}$. The following two lemmas will establish bounds on the service cost and number of facilities of OFL.

▶ **Lemma 12.** *When OFL is run on a stream of $n$ points with exponentially decaying weights, with facility cost $f = \frac{L}{k(1+\log W)}$ where $L > 0$, it produces a service cost of at most $6\mathsf{OPT}_k(P) + 2L$ with probability at least $1/2$.*

**Proof.** The proof follows the standard analysis of Online Facility Location. Let $P$ is the set of points read in a phase. Instead of looking at $|P|$ distinct points with varying weights, we view it as *repeated* points of unit or minimum weight. The total number of points is therefore at most $W = \Theta(h\Delta)$.

**Algorithm 2** k-median clustering in exponential decay streams.

**Input:** $k$, stream $P$, an offline constant approximate $k$-median clustering algorithm KM-RAM.

$L \leftarrow 1$, $\mathcal{C} \leftarrow \emptyset$
**while** solution not found **do**
  $i \leftarrow 0$, $\mathsf{COST} \leftarrow 0$, $f \leftarrow \dfrac{L}{k(1+h\log\Delta)}$.
  **while** stream not ended **do**
    $p \leftarrow$ next point from stream
    $q \leftarrow$ closest point to $p$ in $\mathcal{C}$
    $\sigma \leftarrow \left(\min\left(\dfrac{w(p)\cdot d(p,q)}{f}, 1\right)\right)$
    **if** probability $\sigma$ **then** ▷*do with probability* $\sigma$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
    **else**
      $\mathsf{COST} \leftarrow \mathsf{COST} + w(p)\cdot d(p,q)$
      $w(q) \leftarrow w(q) + w(p)$
    **end if**
    $i \leftarrow i+1$
    **if** $\mathsf{COST} > \gamma L$ or $|\mathcal{C}| > (\gamma-1)k(1+\log W)$ **then** ▷*cost or number of facilities too large*
      break and raise flag ▷*trigger phase change*
    **else if** $i \geq h\log\Delta$ **then** ▷*second sub-phase*
      **for** $l = 1$ to $h+k$ **do** ▷*count points and store them verbatim*
        $p \leftarrow$ next point from stream
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
      **end for**
      break and raise flag
    **end if**
  **end while**
  **if** flag raised **then** ▷*phase change*
    $(\mathcal{C}, \lambda) \leftarrow$ KM-RAM $(\mathcal{C}, k)$ ▷*cluster existing facilities*
    $L \leftarrow \max\left(\beta L, \frac{\lambda}{c_r\gamma}\right)$
  **else**
    Declare solution found
  **end if**
  $(\mathcal{C}, \lambda) \leftarrow$ KM-RAM $(\mathcal{C}, k)$
**end while**
**Output:** $\mathcal{C}, \mathsf{COST}$

We remind the reader that $\mathsf{OPT}_k(P) = \min_{K\subseteq P, |K|=k} \sum_{p\in P} \min_{y\in K} d(p,y)$ is the optimal cost and $\mathsf{COST}(\mathrm{P})$ is the total service cost incurred by OFL. Let $\mathcal{C}^*$ be the set of corresponding facilities allocated by OPT, and $c_i^*$'s denote the optimum $k$ facilities where $i \in [k]$ and $C_i^*$ the set of points from $P$ served by the facility $c_i^*$. Let $A_i = \sum_{x\in C_i^*} d(x, c_i^*)$ be the service cost of $C_i^*$. We now further partition each region into *rings*. Let $S_i^1$ be the first ring around $c_i^*$ that contains half the nearest points in $C_i^*$. Formally, $S_i^1 = \underset{K, |K| = |C_i^*|/2}{\arg\min} \sum_{x\in K} d(x, c_i^*)$.

Furthermore, $S_i^2$ is the second ring around $c_i^*$ containing one-quarter of the points in $C_i^*$ and so on. Therefore, we can inductively define $S_i^j = \underset{K,|K|=|C_i^*|/2^j}{\arg\min} \sum_{x\in K\setminus\cup_{l=1}^{j-1} S_i^l} d(x, c_i^*)$. Note that $S_i^j$ may be not be uniquely identifiable, but their existence suffices for the sake of analysis. Let $A_i^j = \sum_{x\in S_i^j} d(x, c_i)$ be the cost of set $S_i^j$. For a point $p$, use $d_p^*$ and $d_p$ for its optimal cost and cost incurred in the algorithm respectively.

We look at two cases. In the first case, suppose each region has a facility open; let the facility of $S_i^j$ be $s_i^j$. We look at the cost incurred by subsequent points arriving in this region. Consider the set $S_i^j$ and let $q$ be a facility in $S_i^j$. A subsequent point $p$ incurs a cost $d_p = d(p, q)$. By triangle inequality, we have $d_p \leq d_p^* + d_q^*$. By definition of $S_i^j$, we have $d_q^* \leq d_z^*$ for any point $z \in S_i^{j+1}$. We sum over all $z$ in $S_i^{j+1}$ and get $d_q^* \leq \frac{A_i^{j+1}}{|S_i^{j+1}|}$. We therefore get $d_p \leq d_p^* + \frac{A_i^{j+1}}{|S_i^{j+1}|}$. Summing over all points is $S_i^j$, we get $\mathsf{COST}(S_i^j, s_i^j) \leq A_i^j + \frac{|S_i^j| A_i^{j+1}}{|S_i^{j+1}|} = A_i^j + 2\cdot A_i^{j+1}$. Summing over all $j$'s, we get $\mathsf{COST}(C_i^*, c_i^*) \leq 3A_i$. Finally, summing over $i$'s, we get that in the first case $\mathsf{COST}(P, \mathcal{C}^*) \leq 3\mathsf{OPT}_k(P)$. We now look at the second case wherein each region has a facility open. The number of points is at most $W$, therefore, the number of regions is at most $k(1 + \log(W))$. The expected service cost incurred by a region before opening a facility is at most $f$ (See Fact 1, [41]). Therefore, the total service cost $\leq f\, k(1 + \log(W)) = L$. Combining the two cases, we get that $\mathsf{COST}(P, \mathcal{C}^*) \leq 3\mathsf{OPT}_k(P) + L$. Note that when we store points verbatim, we do not incur any service cost. With a simple application of Markov inequality, we get that with probability at least 1/2, $\mathsf{COST}(P, \mathcal{C}^*) \leq 6\mathsf{OPT}_k(P) + 2L$. ◀

▶ **Lemma 13.** *When OFL is run on a stream of $n$ points with exponentially decaying weights, with facility cost $f = \frac{L}{k(1+\log W)}$ where $L > 0$, the number of facilities produced is at most $(2 + \frac{6}{L}\mathsf{OPT}_k(P))k(1 + \log W)$, with probability at least 1/2.*

**Proof.** Considering the points as repeated points of minimum weight, the total number of points is at most $W$ and the total number of regions is at most $k(1 + \log W)$. One facility in each region gives us $k(1 + \log W)$ facilities. After opening a facility in a region, each subsequent point has probability $\frac{d_p}{f}$ to open a facility. Therefore, the expected number of facilities is $\sum_p \frac{d_p}{f}$. We showed in Lemma 12 that $\sum_p d_p \leq 3\ \mathsf{OPT}_k(P)$. Hence, the expected number of facilities is at most $\frac{3\mathsf{OPT}_k(P)}{f} = \frac{3\mathsf{OPT}_k(P)k(1+\log W)}{L}$. A simple application of Markov's inequality completes the proof. ◀

***k*-median clustering.** We now state some key lemmas that will help us establish that the algorithm produces a $\mathcal{O}(1)$ approximation to the $k$-median clustering cost. We then show how these come together and present the detailed guarantees in Theorem 17.

▶ **Lemma 14.** *At every phase change, with probability at least 1/2, $\mathsf{OPT}_k(P) > L$ if $\beta \leq 2$ and $\gamma \geq 9$.*

**Proof.** The phase change is triggered in two ways, either the cost or the number of facilities grows large (more precisely, cost more that $\gamma L$ or the number of facilities greater than $(\gamma - 1)k(1 + \log W)$), or we read too many points. Let us look at the first case. Assume that $L \geq \mathsf{OPT}_k(P)$, then from Lemma 12 and 13, we get that with probability at least 1/2, $\mathsf{COST} \leq 8L$ and the number of facilities is $\leq 8k(1 + \log W)$ respectively. However with $\gamma \geq 9$, neither of the two conditions are met and therefore the premise that a phase change was triggered gives us a contradiction. Hence, in the first case, we get $L < \mathsf{OPT}_k(P)$ with probability at least 1/2.

In the other case, we store points exactly (incurring no additional cost). The only danger in this case is performing a phase change too early (before $\mathsf{OPT}$ has doubled). Let $\underline{\mathsf{OPT}}$ be the value of $\mathsf{OPT}$ at the beginning of the phase, which we assume starts at time $t = z$. Since points cannot be at distance greater than $\Delta$, then

$$\begin{aligned}\underline{\mathsf{OPT}} &\leq \Delta(1 + 2^{1/h} + \ldots + 2^{z/h}) \\ &\leq \Delta \frac{2^{(z+1)/h} - 1}{2^{1/h} - 1}\end{aligned}$$

Now let $\overline{\mathsf{OPT}}$ be the value of $\mathsf{OPT}$ after terminating the phase (which occurs after reading $k + h$ distinct points after the initial $h \log \Delta$ points of the phase). We must prove that $\overline{\mathsf{OPT}} \geq 2\underline{\mathsf{OPT}}$. Observe that after reading $k + h$ distinct points, we must cluster at least $h$ points across a distance of at least 1 (since we can have at most $k$ centers). The weights of these points begin at $2^{(z+h\log\Delta+1)/h}$. Therefore,

$$\begin{aligned}\overline{\mathsf{OPT}} &\geq \underline{\mathsf{OPT}} + \sum_{i=z+h\log\Delta}^{z+h\log\Delta+h} 2^{i/h} \\ &= \underline{\mathsf{OPT}} + \frac{2^{(z+h\log_2(\Delta)+h)/h} - 2^{(z+h\log_2(\Delta))/h}}{2^{1/h} - 1} \\ &\geq \underline{\mathsf{OPT}} + \Delta\left(\frac{2^{(z+1)/h} - 1}{2^{1/h} - 1}\right) \\ &\geq 2\underline{\mathsf{OPT}},\end{aligned}$$

where the second inequality follows from straightforward arithmetic. Let $L'$ be the value of $L$ in the previous phase. Thus,

$$\overline{\mathsf{OPT}} \geq 2\underline{\mathsf{OPT}} > 2L' = \frac{2}{\beta}L$$

where the second inequality holds with probability at least 1/2, as justified above. Setting $\beta \leq 2$ completes the proof. ◀

▶ **Lemma 15.** *At any part in the algorithm, we have* $\mathsf{COST}(P, \mathcal{C}) \leq \left(\gamma + \frac{1+c_r\beta}{\beta-1}\right) L$.

**Proof.** We know that the increase of $\mathsf{COST}(P, \mathcal{C})$ in the current phase is upper bounded by the variable $\mathsf{COST}$ (see Algorithm 2). In a single phase, we have $\mathsf{COST} \leq \gamma L$. Therefore, outside the phase loop, we just need to show that it is at most $\frac{1+c_r\beta}{\beta-1}L$. Note that it changes only by the KM-RAM algorithm, which incurs cost of $\lambda \leq c_r\gamma L$. Suppose that it holds in the previous phase and let $L'$ be the value of $L$ in the previous phase. Then the cost outside the loop is $\gamma L' + \frac{1+c_r\beta}{\beta-1}L' + \lambda \leq \frac{1+c_r\beta}{\beta-1}L$, which finishes the proof. ◀

▶ **Lemma 16.** *With probability at least* 1/2, $L \leq \left(1 + \frac{1}{\gamma} + \frac{1+c_r\beta}{\gamma(\beta-1)}\right) \mathsf{OPT}_k(P)$.

Let $L'$ and $\mathcal{C}'$ denote the values of $L$ and $\mathcal{C}$ in the previous phase. We condition on the event that $L' < \mathsf{OPT}_k(P)$, which we know from Lemma 14 occurs with probability at least 1/2. From the update equation of $L$, we either have $L = \beta L'$ or $L = \frac{\lambda}{c_r\gamma}$. In the first case, we directly get $L \leq \beta\mathsf{OPT}_k(P)$. With $\beta \leq 2$, we get the claim of the lemma. We now look at the second case, where we have $\gamma c_r L \leq \lambda \leq c_r\mathsf{OPT}_k(\mathcal{C}')$ from the guarantee of the KM-RAM algorithm. It is easy to see that $\mathsf{OPT}_k(\mathcal{C}') \leq \mathsf{OPT}_k(P) + \mathsf{COST}(P, \mathcal{C}')$ by a simple application of triangle inequality on all the points. Moreover, from Lemma 15, we have $\mathsf{COST}(P, \mathcal{C}') \leq \left(\gamma + \frac{1+c_r\beta}{\beta-1}\right) L' \leq \left(\gamma + \frac{1+c_r\beta}{\beta-1}\right) \mathsf{OPT}_k(P)$. Combining these, we get $L \leq \left(\frac{1}{\gamma} + 1 + \frac{1+c_r\beta}{\gamma(\beta-1)}\right) \mathsf{OPT}_k(P)$.

We now restate the theorem for the exponential decay model but tailored to Algorithm 2 with all the algorithmic details precisely stated.

▶ **Theorem 17.** *Let $P$ be a stream of $n$ points with exponential decaying weights parametrized by the half-life parameter $h$ and let $k$ be some positive integer. Algorithm 2 run with $\beta \leq 2, \gamma \geq 9, W = \mathcal{O}(h\Delta)$ on the stream $P$ outputs $k$ points, which produce an $\mathcal{O}(1)$ approximation to the optimal cost of $k$-median clustering on $P$ with high probability. The algorithm runs in time $\mathcal{O}(nk \log W)$ and uses space $\mathcal{O}(k \log W + h)$.*

**Proof.** Combining Lemma 15 and 16, we get that

$$\mathsf{COST}(P, \mathcal{C}) \leq \left(\gamma + \frac{1 + c_r\beta}{\beta - 1}\right)\left(\frac{1}{\gamma} + 1 + \frac{1 + c_r\beta}{\gamma(\beta - 1)}\right)\mathsf{OPT}_k(P).$$

Setting $\beta = 2, \gamma = 10$ and $c_r = 3$ gives us that $\mathsf{COST}(P, \mathcal{C}) \leq 40\mathsf{OPT}_k(P)$.

We emphasize that we give a *streaming* guarantee, that is, given a fixed point in the stream, it will hold for all the points seen till then. Note that in the proofs of Lemma 14 and 16, we only need that the random event hold with probability at least $1/2$ *only* in the previous phase. We can therefore amplify the probability of success by running $\log(1/\delta)$ parallel instances to get the bounds to hold with probability at least $1 - \delta$. The space bound of the algorithm is $\mathcal{O}(k \log W + h) = \mathcal{O}(k \log(h\Delta) + h)$, which simply follows from the condition in the algorithm that we don't allow the number of facilities to grow beyond $\mathcal{O}(k(1 + \log(W))$ combined with the fact that we store $k + h$ points verbatim in the second sub-phase. ◀

**Extensions.** As in [41], our algorithm can easily be extended to other distance functions that satisfy the approximate triangle inequality (see Definition 7). In particular, we get constant approximate algorithms for $k$-means clustering and $M$-estimators in the exponential decay model.

## References


1. Charu C. Aggarwal, editor. *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer, 2007.
2. Arvind Arasu and Gurmeet Singh Manku. Approximate Counts and Quantiles over Sliding Windows. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 286–296, 2004.
3. Sepehr Assadi and Sanjeev Khanna. Randomized Composable Coresets for Matching and Vertex Cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 3–12, 2017.
4. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.
5. Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 234–243, 2003.
6. Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A New Framework for Distributed Submodular Maximization. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 645–654, 2016.
7. Jon Louis Bentley and James B Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

8 Vladimir Braverman, Petros Drineas, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Numerical Linear Algebra in the Sliding Window Model. *arXiv preprint*, 2018. `arXiv:1805.03765`.

9 Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint*, 2016. `arXiv:1612.00889`.

10 Vladimir Braverman, Elena Grigorescu, Harry Lang, David P. Woodruff, and Samson Zhou. Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 7:1–7:22, 2018.

11 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering on Sliding Windows in Polylogarithmic Space. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 350–364, 2015.

12 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering Problems on Sliding Windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1374–1390, 2016.

13 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*, pages 26–40. Society for Industrial and Applied Mathematics, 2011.

14 Vladimir Braverman and Rafail Ostrovsky. Smooth Histograms for Sliding Windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 283–293, 2007.

15 Timothy M. Chan and Bashir S. Sadjad. Geometric Optimization Problems over Sliding Windows. *Int. J. Comput. Geometry Appl.*, 16(2-3):145–158, 2006. A preliminary version appeared in the Proceedings of Algorithms and Computation, 15th International Symposium (ISAAC), 2004.

16 Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39. ACM, 2003.

17 Ke 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.

18 Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006. A preliminary version appeared in the Proceedings of the 4th IEEE International Conference on Data Mining (ICDM), 2004.

19 Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 205–214. ACM, 2009.

20 Edith Cohen and Martin Strauss. Maintaining time-decaying stream aggregates. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 223–233, 2003.

21 Graham Cormode, Flip Korn, and Srikanta Tirthapura. Time-decaying aggregates in out-of-order streams. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 89–98, 2008.

22 Graham Cormode, Srikanta Tirthapura, and Bojian Xu. Time-decaying sketches for sensor data aggregation. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC*, pages 215–224, 2007.

23 Graham Cormode, Srikanta Tirthapura, and Bojian Xu. Time-Decayed Correlated Aggregates over Data Streams. In *Proceedings of the SIAM International Conference on Data Mining, SDM*, pages 271–282, 2009.

24 Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W. Mahoney. Sampling Algorithms and Coresets for $\ell_p$ Regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009. A preliminary version appeared in the Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA) 2008.

25 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. A preliminary version appeared in the Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002.

26 Mayur Datar and S. Muthukrishnan. Estimating Rarity and Similarity over Data Stream Windows. In *Algorithms - ESA 2002, 10th Annual European Symposium, Proceedings*, pages 323–334, 2002.

27 Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. Society for Industrial and Applied Mathematics, 2006.

28 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing Diameter in the Streaming and Sliding-Window Models. *Algorithmica*, 41(1):25–41, 2005.

29 Dan Feldman, Amos Fiat, and Micha Sharir. Coresets forWeighted Facilities and Their Applications. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 315–324, 2006.

30 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 569–578, 2011.

31 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd ACM Symposium on Computational Geometry (SoCG)*, pages 11–18, 2007.

32 Dan Feldman, Morteza Monemizadeh, Christian Sohler, and David P Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 630–649. Society for Industrial and Applied Mathematics, 2010.

33 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for $k$-means, PCA and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA), 2013*, pages 1434–1453, 2013.

34 Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

35 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.

36 Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Foundations of computer science, 2000. proceedings. 41st annual symposium on*, pages 359–366. IEEE, 2000.

37 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.

38 Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.

39 Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 100–108, 2014.

40 Tsvi Kopelowitz and Ely Porat. Improved Algorithms for Polynomial-Time Decay and Time-Decay with Additive Error. In *Theoretical Computer Science, 9th Italian Conference, ICTCS Proceedings*, pages 309–322, 2005.

41 Harry Lang. Online Facility Location on Semi-Random Streams. *arXiv preprint*, 2017. arXiv:1711.09384.

42 Lap-Kei Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 290–297, 2006.

43 Adam Meyerson. Online facility location. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 426–431. IEEE, 2001.

44 Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized Composable Core-sets for Distributed Submodular Maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 153–162, 2015.

45 S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

46 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

47 Shang-Yu Su, Pei-Chieh Yuan, and Yun-Nung Chen. How time matters: Learning time-decay attention for contextual spoken language understanding in dialogues. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2133–2142, 2018.

48 Yan Zheng and Jeff M Phillips. Coresets for Kernel Regression. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–654. ACM, 2017.