Improved Sliding Window Algorithms for Clustering and Coverage via Bucketing-Based Sketches

Alessandro Epasto∗ Mohammad Mahdian† Vahab Mirrokni‡ Peilin Zhong§

Abstract

Streaming computation plays an important role in large-scale data analysis. The sliding window model is a model of streaming computation which also captures the recency of the data. In this model, data arrives one item at a time, but only the latest W data items are considered for a particular problem. The goal is to output a good solution at the end of the stream by maintaining a small summary during the stream.

In this work, we propose a new algorithmic framework for designing efficient sliding window algorithms via bucketing-based sketches. Based on this new framework, we develop space-efficient sliding window algorithms for k-cover, k-clustering and diversity maximization problems. For each of the above problems, our algorithm achieves (1 ± ϵ)-approximation. Compared with the previous work, it improves both the approximation ratio and the space.

1 Introduction

The success of large-scale computational systems together with the need of solving problems on massive data motivate the development of efficient algorithms on these systems. The streaming model, where data items arrive one-by-one and can only be accessed by a single pass, was introduced by a seminal work [AMS99]. The goal is to (approximately) solve a problem at the end of the stream while using as little space as possible during the stream. Due to its theoretical elegance and accurate modeling of real-world streaming computing systems (such as Spark Streaming [ZDL†12]), the streaming model has attracted a lot of attention in the past decades. Space-efficient streaming algorithms were developed for a series of fundamental problems including e.g., frequency moments estimation [AMS99, IW05, BO10, JW19], ℓp sampling [MW10, AKO11, JW21], clustering [FL11, GM16, BFL†17], coverage [BMKK14, CW16, ER16, SG09], diversity maximization [Ind04, CPPU17], sparse recovery [NSW19, NS19], low rank matrix approximation [GP14, Lib13, BWZ16, SWZ17], graph problems [FKM†05, AGM13, AGK14, SW15, LS20, CKP†21]. We refer readers to a survey [Mut05] for more streaming algorithms and applications.

However, the classic streaming model does not completely capture the important aspect of the recency of the data. While this model treats all data items equally over the data stream, recent data is much more important in many applications. For example, the recommendation system may not want to generate recommendations based on very old user history. Moreover, in some scenarios, data may need to be removed after a certain time for privacy or legal reasons, e.g., data privacy laws such as the General Data Protection Regulation (GDPR), requires to not retain data beyond a specified period [Upa19].

To study these scenarios, the sliding window model was proposed by [DGIM02]. This model is similar to the streaming model, except that only the latest W data items in the stream are considered for analysis. Although sliding window algorithms are more useful in many applications, designing an efficient sliding window algorithm is usually more difficult than designing an efficient streaming algorithm. The reason is that when W is an upper bound of the size of the stream, the streaming model can be regarded as a special case of the sliding window model. To tackle problems in the sliding window model, several frameworks [DGIM02, BO07, BGL†18] were proposed to reduce the sliding window problems to the streaming problems. While such reductions exist for some problems with certain structural properties, many well-studied problems such as clustering, coverage,
diversity maximization, low rank matrix approximation, graph sparsification and submodular maximization are not captured by such general frameworks and are studied separately in the sliding window literature [CASS16, BLM18, BLM16, BEL+19, BDM+20, CMS13, CNZ16, ELVZ17].

In this work, we develop a new framework for sliding window algorithms. In contrast to requiring certain properties of the underlying objectives, our framework asks for an efficient algorithmic primitive called bucketing-based sketch for the problem. Based on this new framework, we develop near-optimal sliding window algorithms for $k$-cover and $k$-clustering problems. We also develop an efficient sliding window algorithm for diversity maximization which almost matches the space and the approximation ratio of the state-of-the-art streaming algorithm.

**Notation and Preliminaries** We start by providing notation and preliminaries. Let $[n]$ denote the set $\{1, 2, \cdots, n\}$. For $A, B, \alpha \geq 0$, we say $B$ is an $\alpha$-approximation of $A$ if $A \leq B \leq \alpha \cdot A$ ($\alpha \geq 1$) or $\alpha \cdot A \leq B \leq A$ ($\alpha \leq 1$). We use $\mathbf{1}(\cdot)$ to denote the indicator function, i.e., $\mathbf{1}(E) = 1$ if the event $E$ happens and $\mathbf{1}(E) = 0$ otherwise. For any set $S$, we use $2^S$ to denote the family of all subsets of $S$. We use $O(f(n))$ to denote $O(f(n) \log(f(n)))$.

For a vector $x \in \mathbb{R}^d$, we use $x(i)$ to denote the $i$-th entry of $x$ for $i \in [d]$. We use $e_i \in \mathbb{R}^d$ for $i \in [d]$ to denote the standard unit vector where the $i$-th entry of $e_i$ is 1 and all other entries of $e_i$ are 0. For $x \in \mathbb{R}^d$, we use $\|x\|_2$ to denote the $\ell_2$ norm of $x$, i.e., $\|x\|_2 = \sqrt{\sum_{i \in [d]} x(i)^2}$.

In the sliding window model, there is a stream of data items and the $i$-th data item in the stream is $x_i$. At an arbitrary timestamp $N$, the input data set $X$ is implicitly defined by the data stream and a given window size parameter $W > 0$ such that $X = \{x_{N-W+1}, x_{N-W+2}, \cdots, x_N\}$. The goal is to output a good (approximate) solution for some specific computational problem with respect to $X$ while requiring as small space as possible at any time during the stream.

**1.1 Our Results and Comparison to Prior Work** There exist a long line of research on the sliding window model. As the first algorithmic framework for analyzing sliding window model, [DGIM02] proposed exponential histogram as a technique for approximating the class of “weakly additive” functions in this model. Later, a more general framework smooth histogram was developed by [BO07] in which the authors handle the class of “smooth” functions where all weakly additive functions are shown to be smooth. This framework is further generalized by [BGL+18]. Based on these frameworks, space-efficient $(1 \pm \epsilon)$-approximate sliding window algorithms were developed for many problems including count, sum of integers, $\ell_p$ norm, frequency moments, length of longest subsequence, geometric mean, distinct elements, and heavy hitters. However, many fundamental problems such as coverage, clustering and diversity maximization are not smooth or not smooth enough to be $(1 \pm \epsilon)$-approximated by smooth histogram (see Appendix A for a discussion). These problems are studied in the sliding window model separately [BLM18, BLM16, BEL+19, BDM+20]. None of these papers, however, achieve optimal or near-optimal approximation factors or space bounds for these problems. In contrast, we develop a new general framework for the sliding window model, and show efficient sliding window algorithms for $k$-cover, $k$-clustering and diversity maximization as natural applications of this unified framework. Compared to previous work [BEL+19, BDM+20], all our results can achieve an improved $(1 \pm \epsilon)$-approximation with improved space requirements: the space of our algorithm for $k$-cover and $k$-clustering is near optimal and the space for diversity maximization almost matches the previous best algorithm. Previous work either achieves suboptimal approximation (e.g., 2-approximation [BEL+19] or $O(1)$ [BEL+20]) or sub-optimal space requirements (e.g., quadratic space requirement for $k$-clustering in [BDM+20]).

**Algorithmic framework via Bucketing-based sketches.** We define the notion of bucketing-based sketch as a well-structured summary of the data such that it can be easily updated in the sliding window model. Roughly speaking, a bucketing-based sketch of the data contains a number of buckets where each bucket has a threshold. For each data item, we process some information and put the item with processed information into some buckets. If a bucket is full, i.e., the size of the information stored in the bucket is larger than the threshold, then only arbitrary maximal number of items are kept in the bucket. A good (approximate) solution can be recovered from the sketch. The formal definition of the bucketing-based sketch is given by Definition 2.1 and the sliding window algorithmic framework via bucketing-based sketch is described in Algorithm 1. We refer readers to Section 1.2 for high level ideas and concrete simple examples of developing sliding window algorithms via bucketing-based sketches.

In contrast to the smooth histogram framework [BO07] which utilizes the property of the function to make
it suitable for sliding window computations, we ask for an algorithmic primitive, i.e., an efficient bucketing-based sketch, which in turn, yields an efficient sliding window algorithm.

**k-Clustering.** We consider the point set $X$ in the discrete Euclidean space $\{1, 2, \ldots, \Delta\}^d$ for $\Delta, d \in \mathbb{Z}_{\geq 1}$. This is without loss of generality, because if the clustering cost is not zero, we can discretize the space by changing the cost by an arbitrary small multiplicative factor (see e.g., [Ind04, FS05, BFL17, HSYZ18]). For $k \geq 1$ and $p > 0$, $\ell_p$ $k$-clustering asks for a set of $k$ centers $Z \subset \mathbb{R}^d$ such that the $\ell_p$ clustering cost is minimized, i.e.,

$$\text{OPT}(X) := \min_{Z \subset \mathbb{R}^d : |Z| = k} \sum_{x \in X} \min_{z \in Z} ||x - z||_p^p.$$

The above clustering objective is $k$-median if $p = 1$ and is $k$-means if $p = 2$. We extend the sensitivity sampling based sketching technique [FL11, BFL+17] and construct an efficient bucketing-based sketch for $\ell_p$ $k$-clustering. Based on such sketch, we show an efficient sliding window algorithm for $\ell_p$ $k$-clustering in the sliding window model.

**Theorem 1.1. (A simple version of Theorem 5.3)** Consider a point set in $[\Delta]^d$ given by a stream with window size $W \geq 1$. For $\epsilon \in (0, 0.5)$, there is a sliding window algorithm which outputs a $(1 + \epsilon)$-approximation for $\ell_p$ $k$-clustering with probability at least 0.99. The algorithm uses space $(kd + dO(p))/\epsilon^3 \cdot \text{poly} \log(kd\Delta W)$. The update time for each point during the stream is at most $(kd + dO(p))/\epsilon^3 \cdot \text{poly} \log(kd\Delta W)$.

If dimension $d$ is much larger than $\log(k)$, it can be reduced to $O(\log(k)/\epsilon^2)$ by using $O(kd \log(k)/\epsilon^2)$ space based on the dimension reduction technique of [MMR10] (see Appendix D for more details). Since $\Omega(k)$ space is necessary for any multiplicative approximation of $k$-clustering (see Appendix C), our space is optimal up to a poly($\log(kd\Delta W)/\epsilon$) factor.

Our algorithm can actually output a coreset (see Definition 5.1). Thus, we output a $(1 + \epsilon)$-approximation by finding the optimal clustering of the coreset. If we desire a polynomial time approximation algorithm at the end of the stream, we can apply any polynomial time $\alpha$-approximation algorithm over the coreset and we can obtain a $(1 + \epsilon)/\alpha$-approximate solution. For example, polynomial time constant approximation algorithms for $k$-median and $k$-means are known [ANFSW19].

$k$-Clustering has been studied in the sliding window model by a line of work [BLLM15, BFL+17, BEL+20]. To obtain $(1 + \epsilon)$-approximation, [BFL+17] proposed an algorithm that, given a coreset which is maintainable in the streaming model with space $s$, maintains the coreset in the sliding window model using $O(s^2 \epsilon^{-2} \log W)$ space. Since any coreset has size $\Omega(k)$ (by similar argument of Appendix C), the space needed by their algorithm is $\Omega(k^2)$. [BEL+20] proposed a sliding window algorithm, for arbitrary metric spaces, with space linear in $k$ while they can only achieve a constant approximation with a constant $\geq 10$. In contrast to their algorithms, our algorithm achieves both optimal space and $(1 + \epsilon)$-approximation ratio simultaneously.

**k-Cover.** In the $k$-cover problem, given a ground set $\mathcal{E}$ of $m$ elements and a family $\mathcal{S} \subseteq 2^\mathcal{E}$ of $n$ sets, the goal is to choose $k$ sets $\mathcal{P} \subseteq \mathcal{S}$ such that $|\bigcup_{S \in \mathcal{P}} S|$ is maximized. We consider $k$-cover in the edge-arrival model [BEM17], i.e., $\mathcal{S}$ is given by a set of pairs $(S_1, e_1), (S_2, e_2), \cdots \in \mathcal{S} \times \mathcal{E}$ where $(S_i, e_i)$ indicates that element $e_i$ is in the set $S_i$. We show how to use a bucketing-based sketch to implement the sketch proposed by [BEM17]. As a result, we develop an efficient sliding window algorithm for $k$-cover in the edge-arrival sliding window model.

**Theorem 1.2. (A simple version of Theorem 3.3)** Consider a $k$-cover instance over $n$ sets given by an edge-arrival streaming model with window size $W \geq 1$. For $\epsilon \in (0, 0.5)$, there is a sliding window algorithm which outputs a $(1 - \epsilon)$-approximation for $k$-cover with probability at least 0.99. The algorithm uses space $n/\epsilon^3 \cdot \text{poly} \log(nW/\epsilon)$. The update time for each edge during the stream is at most $k \cdot \text{poly} \log(nW)$.

In addition to the above theorem, if polynomial running time is desired to recover an approximation at the end of the stream, we can use near linear time to obtain a $(1 - 1/e - \epsilon)$-approximation (see Theorem 3.2).

To the best of our knowledge, sliding window algorithm for $k$-cover has not been studied previously. The most relevant work is [BEM17] which studies $k$-cover in the edge-arrival streaming model. They achieve $(1 - 1/e - \epsilon)$-approximation if the running time at the end of the stream is required to be polynomial and $(1 - \epsilon)$-approximation otherwise. The space of their algorithm is proven to be near optimal. Note that the streaming model is a special case of the sliding window model when we set the window size $W$ to be an upper bound of the length of the stream. Since the space of our algorithm matches theirs up to a poly-logarithmic factor, the space of our sliding
window algorithm is also near optimal. Coverage problems are also heavily studied in the set-arrival model. Set-arrival model is a special case of edge-arrival model, where each set arrives at a time and brings with it a list of its elements. Even in the set-arrival model, the best known streaming algorithms for k-cover need \( n \) space \([BMKK14, SG09]\).

**Diversity maximization.** We consider the point set \( X \) in the discrete Euclidean space \( \{1, 2, \cdots, \Delta\}^d \) for \( \Delta, d \in \mathbb{Z}_{\geq 1} \). The goal of diversity maximization is to find a subset \( Q \subseteq X \) with \( |Q| = k \) such that \( \text{div}(Q) \) is maximized, where \( \text{div}(\cdot) \) can be any diversity function listed in Table 1, which is originally studied by [IMMM14].

Inspired by the coreset ideas of [CPPU17], we develop an efficient bucketing-based sketch for diversity maximization. Therefore, we obtain efficient sliding window algorithms for diversity maximization problems.

<table>
<thead>
<tr>
<th>Name</th>
<th>Diversity Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote-edge</td>
<td>( \text{div}(Q) = \min_{u \neq v \in Q} |u - v|_2 )</td>
</tr>
<tr>
<td>Remote-clique</td>
<td>( \text{div}(Q) = \sum_{u \neq v \in Q} |u - v|_2 )</td>
</tr>
<tr>
<td>Remote-tree</td>
<td>( \text{div}(Q) ) is the cost of minimum spanning tree of ( Q )</td>
</tr>
<tr>
<td>Remote-cycle</td>
<td>( \text{div}(Q) ) is the cost of minimum TSP tour of ( Q )</td>
</tr>
<tr>
<td>Remote t-trees</td>
<td>( \text{div}(Q) ) is the minimum cost of ( t ) trees spanning ( Q )</td>
</tr>
<tr>
<td>Remote t-cycles</td>
<td>( \text{div}(Q) ) is the minimum cost of ( t ) cycles spanning ( Q )</td>
</tr>
<tr>
<td>Remote-star</td>
<td>( \text{div}(Q) = \min_{u \in Q} \sum_{v \in Q, v \neq u} |u - v|_2 )</td>
</tr>
<tr>
<td>Remote-bipartition</td>
<td>( \text{div}(Q) = \min_{Q_1, Q_2 \subseteq Q, Q_1 \cap Q_2 = \emptyset}</td>
</tr>
<tr>
<td>Remote-forest</td>
<td>( \text{div}(Q) ) is the minimum cost of a perfect matching of ( Q ) (for even ( k ))</td>
</tr>
</tbody>
</table>

Table 1: Diversity Functions

**Theorem 1.3.** (A simple version of Theorem 4.2) Consider a point set in \( [\Delta]^d \) given by a stream with window size \( W \geq 1 \). For \( \epsilon \in (0, 0.5) \), there is a deterministic sliding window algorithm which outputs a \((1 - \epsilon)\)-approximation for the diversity maximization problem. The algorithm uses space \( kd \log(d \Delta) \cdot O(\sqrt{d}/\epsilon)^d \) for remote-edge, remote-tree, remote-cycle, remote t-trees and remote t-cycles, and uses space \( k^2 d \log(d \Delta) \cdot O(\sqrt{d}/\epsilon)^d \) for remote-clique, remote-star, remote-bipartition, remote-forest and remote-matching. The update time for each point during the stream is at most \( O(d \log(d \Delta)) \).

We actually output a subset \( S \subseteq X \) at the end of the stream such that the maximized diversity of \( S \) is a \((1 - \epsilon)\)-approximation of the maximized diversity of \( X \) (see Section 3). Therefore, if we apply any polynomial time \( \alpha \)-approximate diversity maximization algorithm on \( S \), we can get a \((1 - \epsilon)\alpha\)-approximation of the maximized diversity of \( X \) in polynomial time. We refer readers to [CH01] [HIKT99] [HRT97] [Tam91] for polynomial time constant approximation algorithms for specific diversity functions.

Diversity maximization was previously studied in the sliding window model by [BEL+19]. In comparison with their algorithm, we achieve \((1 - \epsilon)\)-approximation while they only achieve \((1/5 - \epsilon)\)-approximation. On the other hand, our space matches their space while ignoring the factors of \( d \) and \( \epsilon \). In the case where the window size \( W \) is an upper bound of the length of the stream, that is, in the streaming model, diversity maximization was studied by [IMMM14] [CPPU17]. The space needed by [IMMM14] is \( \Omega(\sqrt{W}) \). The space used by [CPPU17] is almost the same as ours.

### 1.2 High-Level Ideas of Sliding Window Algorithms via Bucketing-based Sketches

To deliver the high level ideas of our framework, in this section, we consider two simple problems fitting our framework.

Consider a (multi-)set \( X \subseteq \{0, 1\} \). Let \( X_0 = \{x \in X \mid x = 0\} \), \( X_1 = \{x \in X \mid x = 1\} \). Suppose \( |X_1| \in [o, 2 \cdot o] \) for some \( o > 0 \). There is a simple way to construct a bucketing-based sketch \( Z(X) \) for estimating \( |X_1| \) up to \((1 \pm \epsilon)\)-approximation. Let \( Z(X) \) only contain one bucket, and we add each \( x \) with \( x = 1 \) into the bucket with probability \( p = \min\left(\Theta(\log(1/\delta))/(\epsilon^2 \cdot o), 1\right) \). By Chernoff bound, \(|Z(X)|/p \) is a \((1 \pm \epsilon)\)-approximation to \(|X_1| \) with probability at least \( 1 - \delta \). Now consider that \( X \) is given by a stream with window size \( W \). Then it corresponds to the following sliding window algorithm for estimating \( |X_1| \). Let threshold \( k = 10 \cdot \log(1/\delta)/\epsilon^2 \).

We maintain a set \( H \) during the stream. For each data item \( x \) in the stream, if \( x = 1 \), we add \( x \) into \( H \) with probability \( p \). If \(|H| > k\), we remove the data item from \( H \) with the earliest timestamp. The observation is that
if \(|X_1| \in [o, 2 \cdot o]\) at the end of the stream, \(Z(X) \subseteq H\) with probability at least \(1 - \delta\). Thus, we can recover \(Z(X)\) via \(H\), i.e., \(Z(X) = \{x \in H \mid \text{the timestamp of } x \text{ is at least } N - W + 1\}\), where \(N\) is the final timestamp. Thus, a \((1 \pm \epsilon)\)-approximation to \(|X_1|\) can be obtained via \(H\) and the space needed during the stream is \(O(k)\). Notice that we are able to approximately verify whether \(|X_1| \in [o, 2 \cdot o]\) via \(H\) for any \(X\): If we cannot use \(H\) to recover \(Z(X)\), i.e., the earliest timestamp of an item in \(H\) is later than \(N - W + 1\), then \(|Z(X)| > |H|\) and \(|Z(X)|/|p| > |H|/|p| = k/|p| \geq 2 \cdot o\) which implies that \(|X_1| > 2 \cdot o\). Otherwise, if we can recover \(Z(X)\) from \(H\) and \(|Z(X)|/|p| = |H|/|p| < o\), then \(|X_1| < o\). Thus, by considering \(o = 1, 2, 4, 8, \cdots, W\) in parallel during the stream, we are able to estimate \(|X_1|\) at the end of the stream. The total space needed is \(O(k \log W)\).

The above algorithm can be extended to the following toy 1-median problem. We still consider the (multi-)set \(X \subseteq \{0, 1\}\), the goal is to estimate \(\text{OPT}(X) = \min_{x \in \{0, 1\}} \sum_{x \in X} |x - z|\). It is easy to verify that \(\text{OPT}(X) = \min(|X_0|, |X_1|)\). Suppose \(\text{OPT}(X) \in [o, 2 \cdot o]\). Similar to the problem described in the previous paragraph, we can construct a bucketing-based sketch \(Z'(X)\). \(Z'(X)\) has two buckets, and each has a threshold \(k = 10 \log(1/\delta)/\epsilon^2\). We add each \(x\) with \(x = 1\) into the first bucket with probability \(p = \min(\Theta(\log(1/\delta))/(\epsilon^2 \cdot o), 1)\), and add each \(x\) with \(x = 0\) into the second bucket with probability \(p\). Since \(\min(|X_0|, |X_1|) = \text{OPT}(X) \in [o, 2 \cdot o]\), we know that \(\min(|\{x \in Z'(X) \mid x = 0\}|, |\{x \in Z'(X) \mid x = 1\}|)/p = (1 \pm \epsilon)\)-approximation of \(\text{OPT}(X)\) with probability at least \(1 - \delta\). Now suppose \(X\) is given by a stream with window size \(W\). Based on the same argument as the previous paragraph, we can either obtain the sketch \(Z'(X)\) and get a \((1 \pm \epsilon)\)-approximation of \(\text{OPT}(X)\) via \(Z'(X)\) at the end of the stream or we can approximately confirm that \(\text{OPT}(X) \notin [o, 2 \cdot o]\). By considering \(o = 1, 2, 4, 8, \cdots, W\) in parallel during the stream, we are able to estimate \(\text{OPT}(X)\) at the end of the stream. The total space needed is \(O(k \log W)\). In Appendix 5 we show that toy 1-median problem is not smooth enough to be \((1 \pm \epsilon)\)-approximated by smooth histogram.

By formalizing above ideas in Section 2, we obtain our algorithmic framework in Algorithm 1.

2 A New Algorithmic Framework for Sliding Window Model

Given a set of data items \(X\), a natural class of problems is to estimate the value \(f(X)\) for some non-negative function \(f(\cdot)\). In many situations, \(X\) is very large, and we cannot afford to store entire \(X\) in the space. A common approach is to compute a small (randomized) summary/sketch \(Z(X)\) such that we can estimate \(f(X)\) by \(z(Z(X))\) for some function \(z(\cdot)\). If \(z(Z(X))\) is an \(\alpha\)-approximation to \(f(X)\) (with a good probability), then we say \(Z(\cdot)\) is an \(\alpha\)-approximate (randomized) sketch function for \(f(\cdot)\), and \(z(\cdot)\) is the corresponding recover function. Similarly, if (with a good probability) \(z(Z(X))\) either outputs \text{FAIL} or outputs an \(\alpha\)-approximation to \(f(X)\), then we say \(Z(\cdot)\) is a weak \(\alpha\)-approximate (randomized) sketch function for \(f(\cdot)\). If it is clear in the context, we will call a sketch function a sketch for short. We say \(Z(\cdot)\) is a weak \(\alpha\)-restricted \(\alpha\)-approximate sketch for \(f(\cdot)\) if \(Z(\cdot)\) is a weak \(\alpha\)-approximate sketch for the restriction of \(f(\cdot)\) to \(\{X \mid f(X) \geq 0\}\).

In the following, we give a definition of a particular type of sketches. We call it bucketing-based sketch. We will show that this sketch plays an important role in designing our sliding window algorithms. Figure 1 shows an explanation of the bucketing-based sketch.

**Definition 2.1.** (Bucketing-based sketch) Let \(\mathcal{X}\) be the universe of data items. We say \(z(\cdot)\) is a (randomized) bucketing-based sketch function, if for any \(X \subseteq \mathcal{X}\), \(Z(X)\) satisfies following properties:

1. \(Z(X)\) is a tuple of sub-sketches \((Z_1(X), Z_2(X), \cdots, Z_t(X))\).
2. For \(i \in [t]\), there is a (random) filter function \(h_i : \mathcal{X} \rightarrow \{0, 1\}\), a (random) bucketing function \(g_i : \mathcal{X} \rightarrow B\), a (random) processing function \(\zeta_i : \mathcal{X} \rightarrow \mathcal{I}\) and a threshold \(T_i \geq 0\) such that \(Z_i(X) \subseteq \mathcal{X} \times \mathcal{I}\) is constructed as the following:
   
   (a) For \(x \in X\) with \(h_i(x) = 0\), \((x, \zeta_i(x))\) must not be added into \(Z_i(X)\).
   
   (b) Consider each bucket \(B \in \mathcal{B}\). If \(|\sum_{x \in X \mid h_i(x) = 1, g_i(x) = B} |\zeta_i(x)| \leq T_i\), where \(|\zeta_i(x)|\) denotes the space needed to store the information \(\zeta_i(x)\), then for every \(x \in X\) with \(h_i(x) = 1\) and \(g_i(x) = B\), \((x, \zeta_i(x))\) is added into \(Z_i(X)\). Otherwise, \((x_1, \zeta_i(x_1)), (x_2, \zeta_i(x_2)), \cdots, (x_s, \zeta_i(x_s))\) are added into \(Z_i(X)\) for some arbitrary points \(x_1, x_2, \cdots, x_s\) satisfying \(h_i(x_1) = h_i(x_2) = \cdots = h_i(x_s) = 1, g_i(x_1) = g_i(x_2) = \cdots = 1\).

An arbitrary tiebreaker is allowed for selecting \(x_1, x_2, \cdots, x_s\). In our paper, we keep the data items with the latest timestamps during the stream.
The space budget of $Z(X)$ is defined as $\sum_{i=1}^{t} \min(T_i, \sum_{x \in X: h_i(x) = 1, g_i(x) = B} |\zeta_i(x)|)$.

It is easy to see that the space budget of $Z(X)$ defined in Definition 2.1 is always an upper bound of the actual space used by $Z(X)$. The notion of the space budget is more convenient in the analysis since the space budget of $Z(X)$ is always an upper bound of the space budget of $Z(X')$ if $X' \subseteq X$, while the actual space used by $Z(X)$ can be less than the actual space used by $Z(X')$. This is because when a bucket $B \in B$ is full, data items $x_1, x_2, \cdots, x_s$ in condition $2 \beta$ of Definition 2.1 can be any choice. Different choices of $x_1, x_2, \cdots, x_s$ will result in different actual space used by $Z(X)$. In addition, it is easy to verify that if the size of $\zeta_i(x)$ is the same for all data items $x$, the space budget of $Z(X)$ is always the same as the actual space used by $Z(X)$.

We show that if $f(\cdot)$ admits a weak $\alpha$-approximate bucketing-based sketch, there is a sliding window algorithm for estimating $f(\cdot)$. The algorithm is shown in Algorithm 1.

**Theorem 2.1.** Let $X$ be the universe of data items. Let $D$ be an arbitrary family of subsets of $X$. Consider a function $f : D \rightarrow \mathbb{R}_{\geq 0}$ satisfying that $\forall X \in D, f(X) \in [m, M]$ for $0 < m \leq M$. Suppose $\forall o \in [m, M]$, there always exists a weak $\alpha$-restricted $\alpha$-approximate bucketing-based sketch $Z^o(\cdot) = (Z^o_1(\cdot), Z^o_2(\cdot), \cdots, Z^o_t(\cdot))$ for $f(\cdot)$ such that

1. the recover function over $Z^o(X)$ either outputs $FAIL$ or outputs an $\alpha$-approximation to $f(X)$ with probability at least $1 - \delta$ conditioned on $f(X) \geq o$;
2. if $f(X) \leq 2 \cdot o$, with probability at least $1 - \delta$, the space budget of $Z^o(X)$ is upper bounded by $S$ and the recover function over $Z^o(X)$ does not output $FAIL$.

Let $X \in D$ be a set of data items given by a sliding window over a stream. Then there is a sliding window algorithm (Algorithm 2) which outputs an $\alpha$-approximation to $f(X)$ and uses space at most $O(S \cdot \log(M/m))$. The success probability of the algorithm is at least $1 - O(\delta \cdot \log(M/m))$. Furthermore, if the time needed to compute a filter function, a bucketing function and a processing function on any data item $x \in X$ is always upper bounded by $T$, the update time of the algorithm for each data item is at most $O(t \log(M/m) \cdot T + S \cdot \log(M/m))$. 

![Figure 1: Bucketing-based sketch.](image-url)
Algorithm 1 Sliding Window Algorithmic Framework via Bucketing-based Sketches

1: Pre-determined: \(f(\cdot), S, m \) and \( M \).

2: Input: A stream of data items \( x_1, x_2, \cdots \in X \) and the window size \( W \).

3: \( O := \{m - 2, m - 4, m - 8, \cdots, M\} \).

4: For \( o \in O \), let \( Z^o(\cdot) = (Z^o_1(\cdot), Z^o_2(\cdot), \cdots, Z^o_m(\cdot)) \) be a weak \( \alpha \)-restricted \( \alpha \)-approximate bucketing-based sketch function for \( f(\cdot) \) described by Theorem 2.1

5: For \( o \in O, i \in [t], \) let \( h^o_i(\cdot), g^o_i(\cdot), \zeta^o_i(\cdot) \) and \( T^o_i \) be the filter function, bucketing function, processing function and the threshold function corresponding to the sub-sketch function \( Z^o_i(\cdot) \) respectively.

6: For \( o \in O \), initialize \( l_o := 1, H^o := (H^o_1, H^o_2, \cdots, H^o_m) \) and \( \forall i \in [t], H^o_i := 0 \).

7: Initialize current timestamp \( N := 0 \).

8: for the next data item \( x_{N+1} \) in the stream do

9: \( N := N + 1 \).

10: Set the timestamp \( \tau(x_N) := N \).

11: for \( o \in O \) do

12: for \( i \in [t] \) do

13: If \( h^o_i(x_N) = 1 \), add \( (x_N, \zeta^o_i(x_N)) \) into \( H^o_i \).

14: while \( \sum_{(x, \zeta^o_i(x)) \in H^o_i} (g^o_i(x) - g^o_i(x_N)) |\zeta^o_i(x)| > T^o_i \) do

15: \( \) Remove \( y, \zeta^o_i(y) \in \{ (x, \zeta^o_i(x)) \in H^o_i \mid g^o_i(x) = g^o_i(x_N) \} \) with the smallest \( \tau(y) \) from \( H^o_i \).

16: end while

17: end for

18: while the size of \( H^o \) is larger than \( S \) do

19: For \( i \in [t], \) if \( \exists (x, \zeta^o_i(x)) \in H^o_i \) such that the timestamp \( \tau(x) \le l_o \), remove \( (x, \zeta^o_i(x)) \) from \( H^o_i \).

20: \( l_o := l_o + 1 \).

21: end while

22: end for

23: end for

24: For each \( o \in O \) with \( l_o \le N - W + 1 \), let \( \tilde{H}^o = (\tilde{H}^o_1, \cdots, \tilde{H}^o_t) \) be \( H^o = (H^o_1, \cdots, H^o_m) \) but for each \( i \in [t] \) remove all \( (x, \zeta^o_i(x)) \) with timestamp \( \tau(x) \le N - W \) from \( H^o_i \).

25: Find the smallest \( o^\ast \) such that the recover function over \( \tilde{H}^o^\ast \) does not output \( \text{FAIL} \) and return the output of the recover function over \( \tilde{H}^o^\ast \) as the \( \alpha \)-approximation to \( f^\ast(\{x_{N-W+1}, x_{N-W+2}, \cdots, x_N\}) \).

Proof. Firstly, let us analyze the total space used by Algorithm 1. The total space needed is the space to store \( H^o \) for every \( o \in O \). Due to the loop in line 18-21 of Algorithm 1, the size of \( H^o \) is always upper bounded by \( S \). Thus, the total space of the correctness is at most \( O(|O| \cdot S) = O(S \cdot \log(M/m)) \).

Next, let us analyze the correctness. Let \( X^o = \{x_{l_o}, x_{l_o+1}, \cdots, x_N\} \) during the stream.

Claim 2.1. At any time at the end (or before the start) of the iteration of the loop in the line 8-23 of Algorithm 1, \( \forall o \in O, H^o \) is always \( Z^o(X^o) \). Furthermore, \( \forall o \in O, i \in [t], H^o_i \) only contains maximal number of data items with the lastest timestamps, i.e., \( \forall x \in X^o \) with \( h^o_i(x) = 1 \), if \( (x, \zeta^o_i(x)) \notin H^o_i \) then \( \sum_{y \in X^o : \tau(y) \ge \tau(x), h^o_i(y) = 1} g^o_i(y) - g^o_i(x_N) |\zeta^o_i(y)| > T^o_i \).

Proof. The proof is by induction. Let \( N \) be the value as at the end (or before the start) of the iteration of the loop in the line 8-23 of Algorithm 1. Consider an arbitrary \( o \in O \). Consider the case when \( N = 0 \). \( X^o \) is empty and thus the claim holds. In the following we consider the case when the claim holds for \( N - 1 \).

Let us first consider the behavior of the loop in line 12-17. Due to line 13, \( x_N \) can be added into \( H^o_i \) for \( i \in [t] \) only when \( h^o_i(x_N) = 1 \). Thus, the condition 2b of Definition 2.1 is satisfied. Fix \( i \in [t], \) and let the bucket \( B = g^o_i(x_N) \). Since we may add \((x_N, \zeta^o_i(x_N))\) into \( H^o_i \), the bucket \( B \) may violate the condition 2b of Definition 2.1. By our induction hypothesis, the condition 2b holds before processing \( x_N \). There are two cases. In the first case \( \sum_{x \in X^o \setminus \{x_N\} : h^o_i(x) = 1} g^o_i(x) = B |\zeta^o_i(x)| \le T^o_i \). In this case, \( H^o_i = \{x, \zeta^o_i(x) \mid x \in X^o, h^o_i(x) = 1, g^o_i(x) = B \} \) before line 14. In the second case \( \sum_{x \in X^o \setminus \{x_N\} : h^o_i(x) = 1} g^o_i(x) = B |\zeta^o_i(x)| \le T^o_i \). In this case, \( H^o_i = \{x, \zeta^o_i(x) \mid x \in X^o, h^o_i(x) = 1, g^o_i(x) = B \} \) before line 14. In either case, due to the loop in line 14-16, it is easy to verify that the condition 2b of Definition 2.1 is always preserved at the end of the loop. Furthermore, due to the induction hypothesis and the condition of the loop in line 14, we have that \( \forall x \in X^o \) with \( h^o_i(x) = 1 \) if \( (x, \zeta^o_i(x)) \notin H^o_i \), then \( \sum_{y \in X^o : \tau(y) \ge \tau(x), h^o_i(y) = 1} g^o_i(y) - g^o_i(x_N) |\zeta^o_i(y)| > T^o_i \).
Next, let us analyze the behavior of the loop in line 18. Let us focus on a particular \( o \in O \) and \( i \in [t] \). Since line 19 only removes elements from \( H^o_i \), the condition 2a will not be violated. Furthermore, since line 19 only removes elements with the earliest timestamps, we still have that \( \forall x \in X^o \) with \( h^o_i(x) = 1 \), if \( (x, \zeta^o_i(x)) \notin H^o_i \) then \( \sum_{y \in X^o: y \geq \tau(x), h^o_i(y) = 1} g^o_i(y) = g^o_i(x) |\zeta^o_i(y)| > T_i^o \). Now we are going to prove that condition 2b of Definition 2.1 holds after line 20. Suppose that the condition 2b does not hold after line 20. We can find \( x' \in X^o \) such that \( h^o_i(x') = 1 \), \( (x', \zeta^o_i(x')) \notin H^o_i \) and \( \sum_{x \in X^o: x \geq \tau(x), h^o_i(x) = 1} g^o_i(x) \leq T_i^o \) which contradicts to the condition \( \forall x \in X^o \) with \( h^o_i(x) = 1 \) if \( (x, \zeta^o_i(x)) \notin H^o_i \), then \( \sum_{y \in X^o: y \geq \tau(x), h^o_i(y) = 1} g^o_i(y) = g^o_i(x) |\zeta^o_i(y)| > T_i^o \). Thus the condition 2b of Definition 2.1 must hold after line 20.

Thus, the claimed statement always holds.

Due to Claim 2.1, we are able to verify that \( \tilde{H}^o \) for each valid \( o \) in line 24 is equivalent to \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \): the condition 2a of Definition 2.1 holds since line 24 only removes elements, and the condition 2b holds since otherwise we can find \( x' \in \{x_N-W+1, x_N-W-2, \cdots, x_N\}, i \in [t] \) such that \( h^o_i(x') = 1 \), \( (x', \zeta^o_i(x')) \notin H^o_i \), \( \sum_{x \in X^o: x \geq \tau(x), h^o_i(x) = 1} g^o_i(x) \leq T_i^o \) which violates Claim 2.1. Since for \( o \in O \), \( Z^o(\cdot) \) is a weak \( \alpha \)-restricted \( \alpha \)-approximate sketch for \( f(\cdot) \). In the remaining of the proof, we will show that \( o^* \leq f(x_N-W+1, x_N-W-2, \cdots, x_N) \) with a good probability.

By the construction of \( O \), we can find \( o' \in O \) such that \( f(x) \in [o', 2 \cdot o'] \). With probability at least \( 1 - \delta \), \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) has space budget at most \( S \), and the recover function over \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) does not output \( \text{FAIL} \). We condition on that \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) has space budget at most \( S \), and the recover function over \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) does not output \( \text{FAIL} \). If \( l_o > N - W + 1 \), by the condition of line 18 and the proof of Claim 2.1, it implies that the space budget of \( Z^o(N-W+1, N-W+2, \cdots, N') \) for some \( N' \leq N \) is greater than \( S \) which contradicts to that \( Z^o(x_N-W+1, \cdots, x_N) \) has space budget at most \( S \). Thus, we must have \( l_o \leq N - W + 1 \). Thus, by line 25 we know that \( o^* \leq o' \leq f(x) \).

For any \( o \in O \) with \( o \leq f(x_N-W+1, x_N-W-2, \cdots, x_N) \), with probability at least \( 1 - \delta \), we are able to obtain an \( \alpha \)-approximation from \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) conditioned on that the recover function over \( Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \) does not output \( \text{FAIL} \). By taking union bound over all \( o \in O \), with probability at least \( 1 - O(\delta \cdot \log(M/m)) \), we are able to obtain an \( \alpha \)-approximation to \( f(x_N-W+1, x_N-W-2, \cdots, x_N) \) from \( \tilde{H}^o = Z^o(x_N-W+1, x_N-W-2, \cdots, x_N) \).

Finally, let us analyze the update time of the algorithm. When the algorithm processes \( x_N \), the time needed to compute \( h^o_i(x), g^o_i(x), \zeta^o_i(x) \) for each \( i \in [t], o \in O \) is always upper bounded by \( T \). Thus the total time needed to compute the filter function, bucketing function and processing function on \( x_N \) is at most \( O(t \cdot |O| \cdot T) = O(t \cdot \log(M/m) \cdot T) \). In the remaining of the updating process, the algorithm can only remove items from \( H^o \) for \( o \in O \). Since the size of \( H^o \) for \( o \in O \) is at most \( S \), the overall time of removal is at most \( O(S \cdot |O|) = O(S \cdot \log(M/m)) \). Thus, the overall update time is \( O(t \log(M/m) \cdot T + S \cdot \log(M/m)) \).

Remark 2.1. Although Theorem 2.1 does not handle the instance \( X \) with \( f(X) = 0 \), it is usually easy to verify whether \( f(X) = 0 \) in the sliding window model for many problems (such as all problems studied in this paper). Therefore, we can use Algorithm 1 to handle the case \( f(X) > 0 \) and run a sliding window procedure to verify whether \( f(X) = 0 \) in parallel.

3 k-Cover in the Sliding Window Model

In the k-cover problem, there is a ground set \( E \) of \( m \) elements and a family \( S \subseteq 2^E \) of \( n \) subsets of the elements. Let \( P \subseteq S \) be a subfamily of subsets. The coverage of \( P \) is denoted as \( \mathcal{C}(P) = \bigcup_{S \in P} S \), i.e., the total number of elements in the union of all subsets in \( P \). The goal of k-coverage is to find a subfamily \( P \subseteq S \) with size \( |P| = k \) such that \( \mathcal{C}(P) \) is maximized. The k-cover problem can also be described by a bipartite graph \( G \), where \( S \) corresponds to the vertices on the one part, and \( E \) corresponds to the vertices on the other part. There is an edge between a subset \( S \in P \) and an element \( i \in E \) if and only if the element \( i \) is in the subset \( S \). For a set \( P \) of vertices in \( G \), let \( \Gamma(G, P) \) denote the neighbors of \( P \) in \( G \). In particular, if \( P \) corresponds to a subfamily of subsets in \( S \), then we have \( |\Gamma(G, P)| = \mathcal{C}(P) \). We use \( \text{OPT}_k(G) \) to denote the optimal k-cover value, i.e., \( \text{OPT}_k(G) := \max_{P \subseteq S: |P| = k} |\Gamma(G, P)| \). If \( G \) is edge-clear in the context, we will use \( \text{OPT}_k \) to denote \( \text{OPT}_k(G) \) for short.

In this paper, we consider k-cover in the edge-arrival model [BEM17]. In particular, there is a stream of
set-element pairs \((S_1, e_1), (S_2, e_2), \ldots\), where \(S_i \in S\) corresponds to a subset, \(e_i \in E\) corresponds to an element. Given a window size \(W\), at timestamp \(N\), the goal is to approximate \(\text{OPT}_k(G)\), where \(G\) is represented by the edges \((S_{N-W+1}, e_{N-W+1}), (S_{N-W+2}, e_{N-W+2}), \ldots, (S_N, e_N)\).

### 3.1 Offline Sketch via Subsampling

Let us review a sketch proposed by [BEM17]. In particular, a sketch graph \(H'_p\) for \(G\) is created. Let \(\delta \in (0, 0.5)\) be a probability parameter. Let \(h\) be an \(O(k \log(1/\delta) \log n)\)-wise independent hash function which maps each element in \(E\) to \([0, 1]\) uniformly at random. Let \(\varepsilon \in (0, 0.5)\) be a given accuracy parameter. For \(p \in [0, 1]\), the sketch \(H'_p\) is constructed as follows. \(H'_p\) is a bipartite graph where the vertices on the one part correspond to \(S\) and the vertices on the other part correspond to the elements \(\{e \in E \mid h(e) \leq p\}\). For an element \(e \in E\) with \(h(e) \leq p\), if the degree of \(e\) in \(G\) is at most \(n \log(1/e)/(ek)\), we add all edges connected to \(e\) in \(G\) to \(H'_p\) otherwise we add arbitrary \(n \log(1/e)/(ek)\) edges connected to \(e\) in \(G\) to \(H'_p\).

The properties of \(H'_p\) is stated in the following lemma. The randomness is over the choice of the hash function \(h\).

**Lemma 3.1.** ([BEM17]) For \(\min \left(\frac{k \log(1/\delta) \log(n)}{\varepsilon^2 \text{OPT}_k(G)}, 1\right) \leq p \leq 1\), with probability at least \(1 - O(\delta)\), for any \(\alpha \in [0, 1]\) and any \(P \subseteq S\) with \(|P| = k\) and \(|\Gamma(H'_p, P)| \geq \alpha \cdot \text{OPT}_k(H'_p)\), we have \(|\Gamma(G, P)| \geq (\alpha - 3\varepsilon) \cdot \text{OPT}_k(G)\), and \((1 + \varepsilon) \cdot \text{OPT}_k(G) \geq \frac{1}{2} \cdot |\Gamma(H'_p, P)| \geq (\alpha - 2\varepsilon) \cdot \text{OPT}_k(G)\). Furthermore, for any \(C \geq 1\), if \(p \leq C \cdot \frac{k \log(1/\delta) \log(n)}{\varepsilon^2 \text{OPT}_k(G)}\), then with probability at least \(1 - O(\delta)\), the number of edges in \(H'_p\) is at most \(C \cdot 100n \log(1/\delta) \log(1/\varepsilon) \log(n)/\varepsilon^2\).

### 3.2 Bucketing-based Sketch for \(k\)-Cover Problem

To design an efficient sliding window algorithm for \(k\)-cover, we only need to develop an efficient bucketing-based sketch according to Theorem 2.1. Fortunately, the sketch graph \(H'_p\) described in Section 3.1 yields an efficient bucketing-based sketch. In this section, we will show how to construct the bucketing-based sketch. Notice that \(\text{OPT}_k(G)\) is always in \([1, W]\). Let \(\alpha \in [1, W]\) and let \(\delta, \varepsilon\) be the same as described in Section 3.1. We construct the sketch function \(Z(\cdot)\) as following. \(Z(\cdot)\) only contains one sub-sketch \(Z_1(\cdot)\), i.e., \(Z(\cdot) = (Z_1(\cdot))\). For convenience, we abuse the notation and denote \(Z_1(\cdot)\) as \(Z(\cdot)\). We use \(Z(G)\) to denote \(Z(X)\) for \(X \subseteq S \times E\) where \(G\) is the bipartite graph corresponding to the edge set \(X\). According to Definition 2.1 to describe \(Z(\cdot)\), we only need to specify the filter function \(h(\cdot)\), the bucketing function \(g(\cdot)\), the processing function \(\zeta(\cdot)\) and the threshold \(T\). Let \(h\) be the same as described in Section 3.1, an \(O(k \log(1/\delta) \log n)\)-wise independent hash function which maps each element in \(E\) to \([0, 1]\) uniformly at random. Let \(p = \min \left(\frac{k \log(1/\delta) \log(n)}{\varepsilon^2 \alpha}, 1\right)\). We construct \(h(\cdot), g(\cdot), \zeta(\cdot)\) and \(T\) as follows:

1. \(\forall (S, e) \in S \times E, h(S, e) = \begin{cases} 1, & h(e) \leq p, \\ 0, & \text{o.w.} \end{cases}\)
2. \(\forall S, e \in S \times E, g(S, e) = e\)
3. \(\forall (S, e) \in S \times E, \zeta(S, e) = (S, e)\)
4. \(T = n \log(1/e)/(ek)\).

**Lemma 3.2.** The space budget of \(Z(G)\) is the same as the number of edges of \(H'_p\), and \(H'_p\) can be constructed via \(Z(G)\) in \(O(|Z(G)|)\) time.

**Proof.** Since each item in \(Z(G)\) is a tuple \(((S, e), \zeta(S, e))\) and \(\zeta(S, e) = (S, e)\), we abuse the notation and regard each item in \(Z(G)\) as \((S, e) \in S \times E\), and it is clear that the space budget of \(Z(G)\) is the same as the size of \(Z(G)\). It is easy to verify that there is a one-to-one correspondence between the items of \(Z(G)\) and the edges of \(H'_p\). Consider an element \(e \in E\). If \(h(e) > p\), any edge \((S, e)\) will neither be added to \(Z(G)\) \(((S, e)\) is filtered by the filter function \(h(\cdot))\) nor \(H'_p\). If \(h(e) \leq p\) and the degree of \(e\) in \(G\) is at most \(T = n \log(1/e)/(ek)\), every edge \((S, e)\) will be added into \(Z(G)\) since the bucket \(e\) will not be full, and \((S, e)\) will also be added into \(H'_p\) due to the construction of \(H'_p\). If \(h(e) \leq p\) and the degree of \(e\) in \(G\) is larger than \(T = n \log(1/e)/(ek)\), arbitrary \(T\) edges \((S, e)\) will be added into \(Z(G)\) since the bucket \(e\) is full, and these edges are also added into \(H'_p\) according to the construction of \(H'_p\). \(\square\)
Theorem 3.1. (Bucketing-based sketch for k-cover under edge-arrival model) Consider the k-cover problem over a set of edges in $S \times E$ where $E$ is a ground set of $m$ elements and $S$ is a family of $n$ sets of elements. For any $\epsilon, \delta \in (0, 0.5), \alpha \geq 1, \alpha \in [0, 1]$, there always exists an $\alpha$-restricted ($\alpha - \epsilon$)-approximate bucketing-based sketch $Z^\alpha(\cdot)$ such that for any edge set $X \subseteq S \times E$,

1. the recover function over $Z^\alpha(X)$ outputs an ($\alpha - \epsilon$)-approximation to the optimal k-cover value with probability at least $1 - \delta$ conditioned on that the optimal value is at least $\alpha$;

2. if the optimal value is at most $2 \cdot o$, with probability at least $1 - \delta$, the space budget of $Z^\alpha(X)$ is upper bounded by $O(n \log(1/\delta) \log(1/e) \log(n)/\epsilon^3)$.

The time needed to compute a filter function, a bucketing function and a processing function on any $(S, e) \in S \times E$ is always upper bounded by $O(k \log(1/\delta) \log(n))$. Furthermore, if $\alpha \leq 1 - 1/e$, the time to compute recover function over $Z^\alpha(X)$ is at most $O(|Z^\alpha(X)|)$.

Proof. Let $G$ be the bipartite graph corresponding to the edge set $X$. We will prove that the sketch $Z(G)$ described in this section is the desired sketch $Z^\alpha(X)$.

According to Lemma 3.2, we can recover $H'_p$ for $p = \min \left( \frac{k \log(1/\delta) \log(n)}{\epsilon^3 o}, 1 \right)$ via $Z(G)$ in $O(|Z(G)|)$ time. If $\text{OPT}_k(G) \geq o$, we have $p \geq \min \left( \frac{k \log(1/\delta) \log(n)}{\epsilon^3 o \cdot \text{OPT}_k(G)}, 1 \right)$. Let $P' \subseteq S$ with $|P'| = k$ such that $|\Gamma(H'_p, P')| = \alpha \cdot \text{OPT}_k(H'_p)$. According to Lemma 3.1, we have $\exists (1 + \epsilon) \cdot \text{OPT}_k(G) \geq \frac{1}{p} \cdot |\Gamma(H'_p, P')| \geq (\alpha - O(\epsilon)) \cdot \text{OPT}_k(G)$ with probability at least $1 - \delta$. Thus, $Z(G)$ is an $\alpha$-restricted ($\alpha - O(\epsilon)$)-approximate bucketing-based sketch. If $\alpha = 1$, we can find $P'$ by brute-force search which takes $O(n^k |Z^\alpha(X)|)$ time. If $\alpha = 1 - 1/e$, we can find $P'$ by running a well-known greedy algorithm on $H'_p$ which takes $O(|Z(G)|)$ time. If $\text{OPT}_k(G) \leq 2 \cdot o$, according to Lemma 3.1 with probability at least $1 - \delta$, the number of edges of $H'_p$ is at most $200 n \cdot \log(1/\delta) \log(1/e) \log(n)/\epsilon^3$.

According to Lemma 3.2 the space budget of $Z(G)$ is the same as the number of edges of $H'_p$, and thus $|Z(G)| \leq O(n \log(1/\delta) \log(1/e) \log(n)/\epsilon^3)$. The time to perform the filter function $h$ is the same as the time to evaluate the $O(k \log(1/\delta) \log(n))$-wise independent hash function $h$, which is $O(k \log(1/\delta) \log(n))$. The time to perform the bucketing function $g(\cdot)$ and the processing function $\zeta(\cdot)$ is $O(1)$.

After scaling $\delta$ and $\epsilon$ by a constant factor, we conclude the proof. \hfill \Box

3.3 $k$-Cover in Edge-Arrival Sliding Window Model By plugging the bucketing-based sketch for k-cover into our algorithmic framework (Algorithm 1), we are able to obtain an efficient sliding window algorithm for k-cover problem.

Theorem 3.2. (Sliding window k-cover with fast running time) For any $\epsilon, \delta' \in (0, 0.5)$, there is a $(1 - 1/e - \epsilon)$-approximation algorithm for k-cover in the edge-arrival sliding window model with window size $W \geq 1$ using space $O(n \log(1/\delta') \log(W)/\epsilon^3)$. The update time is $O(k \log(1/\delta') \log(W))$. The running time to get the approximation at the end of the stream is at most $O(n \log(1/\delta') \log(W)/\epsilon^3)$. The success probability is at least $1 - \delta'$.

Proof. By plugging the bucketing-based sketch in Theorem 3.1 with approximation parameter $\alpha = 1 - 1/e$ and probability parameter $\delta = \Theta(\delta'/\log(W))$ into Algorithm 1, we get the desired algorithm.

Since the window size is $W$, the optimal k-cover value is between 1 and $W$. According to Theorem 3.1, there always exists an $\alpha$-restricted ($1 - 1/e - \epsilon$)-approximate bucketing-based sketch $Z^\alpha(\cdot)$ for k-cover such that a $(1 - 1/e - \epsilon)$-approximation to the optimal k-cover value can be recovered with probability at least $1 - \delta$ conditioned on that the optimal k-cover value is at least $\alpha$. Furthermore, if the optimal k-cover value is at most $2 \cdot o$, the space budget of the sketch is at most $O(n \log(1/\delta) \log(1/e) \log(n)/\epsilon^3)$ with probability at least $1 - \delta$. According to Theorem 2.1 after plugging the bucketing-based sketches into Algorithm 1, we obtain the sliding window algorithm which outputs a $(1 - 1/e - \epsilon)$-approximation to the optimal k-cover value. The space needed is at most $O(n \log(1/\delta) \log(1/e) \log(n)/\epsilon^3 \cdot \log(W)) = \tilde{O}(n \log(1/\delta') \log(W)/\epsilon^3)$. The success probability is at least $1 - O(\delta' \cdot \log(W)) = 1 - \delta'$. According to Theorem 3.1 the time needed to compute a filter function, a bucketing function and a processing function on any data item is always at most $O(k \log(1/\delta) \log(n))$. By taking a more
careful analysis, we can get a update time better than that claimed in Theorem 2.1. Consider an iteration of the loop in line 8 of Algorithm 1. We only need to compute filter function, bucketing function and processing function $O(\log W)$ times since $|O| = O(\log W)$ and $t = 1$. It takes $O(k \log(1/\delta) \log(n) \log(W))$ time. Since each tuple in the sketch has size 1, the loop in line 14 of Algorithm 1 has at most one iteration each time. Similarly, the loop in line 21 of Algorithm 1 has at most one iteration each time. Thus, the overall update time is at most $O(k \log(1/\delta) \log(n) \log(W)) = O(k \log(1/\delta) \log(n) \log(W) \log(W))$ time. At the end of the stream, to get the final approximation, we need to evaluate the recover function $O(\log W)$ times according to Algorithm 1. Since the size of each sketch is at most $O(n \log(1/\delta)/\epsilon^3)$, the time needed to compute the recover function for each sketch is at most $O(n \log(1/\delta)/\epsilon^3)$ according to Theorem 3.1. Thus, the overall running time to get the approximation at the end of the stream is at most $O(n \log(1/\delta) \log(W)/\epsilon^3)$.

**Theorem 3.3.** (Sliding window $k$-cover with $(1 - \epsilon)$-approximation) For any $\epsilon, \delta' \in (0, 0.5)$, there is a $(1 - \epsilon)$-approximation algorithm for $k$-cover in the edge-arrival sliding window model with window size $W \geq 1$ using space $O(n \log(1/\delta') \log(W)/\epsilon^3)$. The update time is at most $O(k \log(1/\delta') \log(n) \log(W))$. The success probability is at least $1 - \delta'$.

**Proof.** The proof is almost the same as the proof of Theorem 3.2 except that we choose the approximation parameter $\alpha = 1$ for the bucketing-based sketch in Theorem 3.1. □

4 Diversity Maximization in the Sliding Window Model

We consider diversity maximization problems in $d$-dimensional Euclidean space. In these problems, we are give a set of points $P \subset \mathbb{R}^d$, and the goal is to find a subset $Q$ of $k$ points with the maximum diversity. The diversity of a set $Q$, div$(Q)$ depends on the distances between points in $Q$. In this section, we study diversity maximization problems with various well-known diversity functions div$(\cdot)$ [IMMM14, BEL+19]. In particular, the descriptions of these diversity functions are listed in Table 1.

When $k, d$ and div$(\cdot)$ are specified, we use OPT($P$) to denote the optimal cost of $P$, i.e., OPT($P$) = $\max_{Q \subseteq P : |Q| = k}$ div$(Q)$. Let $\overline{\text{OPT}}(P)$ denote $\text{OPT}(P)$ divided by the number of pairwise distances that are summed in the diversity function div$(\cdot)$. In particular, for remote-edge, $\overline{\text{OPT}}(P) = \text{OPT}(P)/(k(k-1)/2)$; for remote-tree and remote-star, $\overline{\text{OPT}}(P) = \text{OPT}(P)/(k-1)$; for remote-cycle, remote t-cycles and remote-pseudoforest, $\overline{\text{OPT}}(P) = \text{OPT}(P)/k$; for remote t-trees, $\overline{\text{OPT}}(P) = \text{OPT}(P)/(k-t)$; for remote-bipartition, $\overline{\text{OPT}}(P) = \text{OPT}(P)/((k/2) \cdot (k/2))$; and for remote-matching, $\overline{\text{OPT}}(P) = \text{OPT}(P)/(k/2)$. The goal of estimating OPT$(P)$ becomes to estimate $\overline{\text{OPT}}(P)$.

4.1 Offline Sketch via Discretization of the Space

In this section, we show how to get a subset of points (sketch) of $P$ such that a good approximate solution of the obtained subset of points is a good approximate solution for $P$. Given a parameter $\mu \in \mathbb{R}_{>0}$, let $\mathcal{G}_\mu \subset \mathbb{R}^d$ denote the set of regular grid points with side length $\mu$, i.e., $\mathcal{G}_\mu = \{x_1, x_2, \cdots, x_d\} \subset \mathbb{R}^d \mid \forall i \in [d], x_i = \mu \cdot y_i, y_i \in \mathbb{Z}\}$. Let $h_\mu : \mathbb{R}^d \rightarrow \mathcal{G}_\mu$ be a mapping constructed as the following:

$$\forall x = (x(1), x(2), \cdots, x(d)) \in \mathbb{R}^d, h_\mu(x) = ([x(1)/\mu], [x(2)/\mu], \cdots, [x(d)/\mu]) \cdot \mu.$$  

**Fact 4.1.** $\forall p \neq q \in \mathcal{G}_\mu, \|p-q\|_2 \geq \mu. \forall p \in \mathbb{R}^d, \|p-h_\mu(p)\|_2 \leq \sqrt{d} \cdot \mu.$

Let $T_{div} \in \mathbb{Z}_{\geq 1}$ be a parameter which only depends on the type of the diversity function. In particular $T_{div} = 1$ for remote-edge, remote-tree, remote-cycle, remote t-trees and remote t-cycles, and $T_{div} = k$ for remote-clique, remote-star, remote-bipartition, remote-pseudoforest and remote-matching. The sketch $S_\mu$ is constructed as follows. $S_\mu$ is a subset of points of $P$. Initialize $S_\mu$ to be arbitrary $k$ points from $P$ to ensure $|S_\mu| \geq k$. For each point $q \in \mathcal{G}_\mu$ in the image of $P$ under $h_\mu$, if $|\{p \in P \mid h_\mu(p) = q\}| \leq T_{div}$, add all points $p \in P$ with $h_\mu(p) = q$ into $S_\mu$, otherwise, add arbitrary $T_{div}$ points $p \in P$ with $h_\mu(p) = q$ into $S_\mu$. The properties of $S_\mu$ are stated in the following lemma.

**Lemma 4.1.** Let $\epsilon \in (0, 0.5)$. For $0 < \mu \leq \epsilon \cdot \overline{\text{OPT}}(P)/(10\sqrt{d})$, $\overline{\text{OPT}}(P) \geq \overline{\text{OPT}}(S_\mu) \geq (1 - \epsilon) \cdot \overline{\text{OPT}}(P)$. Furthermore, for any $C > 0$, if $\mu \geq C \cdot \epsilon \cdot \overline{\text{OPT}}(P)/(10\sqrt{d})$, $|S_\mu| \leq k \cdot (2\sqrt{d} + 20\sqrt{d}/(C \cdot \epsilon) + 1)^d \cdot T_{div} + k$. 

3015 Copyright © 2022 by SIAM

Unauthorized reproduction of this article is prohibited
Proof. Let us consider the size of $S_\mu$ when $\mu \geq C \cdot \epsilon \cdot \OPT(P)/(10 \sqrt{d})$ for some $C > 0$. We claim that there exists a subset $Q \subseteq P$ with $|Q| < k$ such that $\forall p \in P, \min_{q \in Q} \|p - q\|_2 \leq \OPT(P)$. We use the following procedure to find $Q$:

1. Mark every point in $P$ as uncovered. Let $Q \leftarrow \emptyset$.
2. Choose an arbitrary uncovered point $q \in P$ and add $q$ into $Q$.
3. Mark every point $p \in P$ with $\|p - q\|_2 \leq \OPT(P)$ as covered.
4. Repeat above two steps until $P$ is empty.

It is easy to verify that $\forall p \in P$, there exists $q \in Q$ such that $\|p - q\|_2 \leq \OPT(P)$. If $|Q| \geq k$, we can choose a subset $Q' \subseteq Q$ such that $|Q'| = k$. Notice that the pairwise distances between points in $Q'$ are always greater than $\OPT(P)$ which implies $\text{div}(Q') > \OPT(P)$ and thus leads to a contradiction. Let us consider the size of image set $I = \{h_\mu(p) \mid p \in P\}$. By Fact 4.1, we have $\forall x \in I, \exists p \in P, \|x - p\|_2 \leq \sqrt{d} \cdot \mu$ which implies that $\exists q \in Q, \|x - q\|_2 \leq \sqrt{d} \cdot \mu + \OPT(P)$. By Fact 4.1, we have $\forall x, y \in I, \|x - y\|_2 \geq \mu$. Let $B(x, r)$ denote the ball centered at $x$ with radius $r$, i.e., $B(x, r) = \{y \in \mathbb{R}^d \mid \|x - y\|_2 \leq r\}$. Then, we know that $\bigcup_{x \in I} B(x, \mu/2) \subseteq \bigcup_{q \in Q} B(q, \sqrt{d} \cdot \mu + \OPT(P) + \mu/2)$. By analyzing the volume of the balls, we have $|I| \leq |Q| \cdot (2 \cdot \sqrt{d} + 2 \cdot \OPT(P) + \mu/1)^d \leq k \cdot (2 \sqrt{d} + 20 \sqrt{d} / (C \cdot \epsilon + 1))^d$. Since for each $x \in I$, we keep at most $\text{div}(P) \in P$ with $h_\mu(p) = x$, the size of $S_\mu$ is at most $k \cdot (2 \sqrt{d} + 20 \sqrt{d} / (C \cdot \epsilon + 1))^d \cdot \text{div}(P) + k$.

Next, let us prove the approximation. Firstly, consider the cases for remote-clique, remote-star, remote-bipartition, remote-pseudoforest and remote-matching. In these cases, $\text{div}(P)$ is the optimal solution, i.e., $\text{div}(Q') = \OPT(P)$. We can find a set $Q' = \{q_1', q_2', \ldots, q_k'\}$ be the optimal solution such that $\forall i \in [k], h_\mu(q_i') = h_\mu(q_i)$. By Fact 4.1, we have $\forall i \in [k], \|q_i' - q_i\|_2 \leq 2 \sqrt{d} \cdot \mu$. Thus, $\forall i, j \in [k], \|q_i' - q_j\|_2 \leq 2 \sqrt{d} \cdot \mu$. Since $\mu \leq \epsilon \cdot \OPT(P)/(10 \sqrt{d})$, we have

$$\OPT(P) \geq \OPT(S_\mu) \geq \text{div}(Q') \geq \OPT(P) - 4 \sqrt{d} \cdot \mu \geq (1 - \epsilon) \cdot \OPT(P)$$

which implies that $\OPT(P) \geq \OPT(S_\mu) \geq (1 - \epsilon) \cdot \OPT(P)$.

Then consider the cases for remote-edge, remote-tree, remote-cycle, remote-$t$-trees and remote $t$-cycles. Let $Q^* = \{q_1^*, q_2^*, \ldots, q_k^*\}$ be the optimal solution, i.e., $\text{div}(Q^*) = \OPT(P)$. Let $Q' = \{q_1', q_2', \ldots, q_k'\}$ be a multi-set such that $\forall i \in [k], h_\mu(q_i') = h_\mu(q_i)$. By the similar argument, we can show that $\text{div}(Q') \geq \OPT(P) - 4 \sqrt{d} \cdot \mu \geq (1 - \epsilon) \cdot \OPT(P)$. Notice that if we replace the duplicated points with arbitrary points, the diversity does not decrease. So we can find a set $Q'' \subseteq S_\mu$ such that $\text{div}(Q'') \geq \text{div}(Q')$. Thus, $\OPT(P) \geq \OPT(S_\mu) \geq \text{div}(Q'') \geq (1 - \epsilon) \cdot \OPT(P)$.

4.2 Bucketing-based Sketch for Diversity Maximization To design an efficient sliding window algorithm for diversity maximization, we need to develop an efficient bucketing-based sketch according to Theorem 4.1. In this section, we show how to construct $S_\mu$ described in Section 4.1 via bucketing-based sketch. We suppose that the point set $P \subseteq [\Delta]^d$ if $\OPT(P) \neq 0$, we have $\OPT(P) \in [1, \sqrt{d} \cdot \Delta]$. In this section, we consider the case when $\OPT(P) > 0$. We will handle the case when $\OPT(P) = 0$ in our final sliding window diversity maximization algorithm (see the proof of Theorem 4.2). Let $o \in [1, \sqrt{d} \cdot \Delta]$. Let $\epsilon \in (0, 0.5)$. Let $\mu = \epsilon \cdot o / (10 \sqrt{d})$. Let $h_\mu(\cdot), G_\mu$ and $\text{div}(\cdot)$ be the same as described in Section 4.1. We construct our bucketing-based sketch $Z(\cdot)$ as following. $Z(\cdot)$ is composed by two sub-sketches $(Z_1(\cdot), Z_2(\cdot))$. The goal of $Z_1(\cdot)$ is to maintain arbitrary $k$ points of $P$. The goal of $Z_2(\cdot)$ is to maintain other points in $S_\mu$. According to Definition 2.1, we need to specify the filter functions $h_1(\cdot), h_2(\cdot)$, the bucketing functions $g_1(\cdot), g_2(\cdot)$, the processing functions $\zeta_1(\cdot), \zeta_2(\cdot)$ and the thresholds $T_1, T_2$. For $Z_1(\cdot)$, we construct $h_1(\cdot), g_1(\cdot), \zeta_1(\cdot)$ and $T_1$ as follows:

1. $\forall x \in \mathbb{R}^d, h_1(x) = 1$, i.e., no points are filtered.
2. $Z_1(\cdot)$ only has one bucket, i.e., $\forall x \in \mathbb{R}^d, g_1(x)$ always maps $x$ to the same bucket in $Z_1(\cdot)$.
3. $\forall x \in \mathbb{R}^d, \zeta_1(x) = x$.
4. $T_1 = k$. 

3016 Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/29/22 to 54.218.70.166 Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy
For $Z_2(\cdot)$, we construct $h_2(\cdot), g_2(\cdot), \zeta_2(\cdot)$ and $T_2$ as follows:

1. $\forall x \in \mathbb{R}^d, h_2(x) = 1$, i.e., no points are filtered.

2. The buckets $\mathcal{B} = \mathcal{G}_\mu$, i.e., each grid point in $\mathcal{G}_\mu$ corresponds to a bucket. $\forall x \in \mathbb{R}^d, g_2(x) = h_\mu(x)$.

3. $\forall x \in \mathbb{R}^d, \zeta_2(x) = x$.

4. $T_2 = T_{\text{div}}$.

**Lemma 4.2.** $|Z(P)| = |S_\mu|$, the space budget of $Z(P)$ is at most $O(|S_\mu| \cdot d)$, and $S_\mu$ can be constructed via $Z(P)$ in the time linear in the size of $Z(P)$.

**Proof.** Since $\zeta_1(x) = \zeta_2(x) = x$, we abuse the notation and regard each item in $Z_1(P)$ and $Z_2(P)$ as a point $x$ itself instead of the pair $(x, \zeta_1(x))$ or $(x, \zeta_2(x))$. The construction of $S_\mu$ described in Section 4.1 is equivalent to the following process.

1. Initialize $S_\mu = \emptyset$.

2. Add all points stored in $Z_1(P)$ into $S_\mu$. Since the threshold $T_1 = k$ and $g_1(\cdot)$ maps points into the same bucket, this step corresponds to adding arbitrary $k$ points from $P$ into $S_\mu$.

3. For each bucket $q \in \mathcal{G}_\mu$, add all points $x \in Z_2(P)$ with $g_2(x) = q$ into $S_\mu$. Since the bucketing function $g_2(x) = h_\mu(x)$ and the threshold $T_2 = T_{\text{div}}$, this step corresponds to adding all points $p \in P$ with $h_\mu(p) = q$ into $S_\mu$ if $|\{p \in P \mid h(p) = q\}| \leq T_{\text{div}}$ and adding $T_{\text{div}}$ points $p \in P$ with $h_\mu(p) = q$ into $S_\mu$ otherwise.

Therefore, $|Z(P)| = |S_\mu|$. Since each point has dimension $d$, the space budget of $Z(P)$ is at most $O(|S_\mu| \cdot d)$ The construction time is linear. □

**Theorem 4.1.** (Bucketing-based sketch for diversity maximization) Consider any diversity maximization problem over a set of points $P \subseteq [\Delta]^d$ with parameter $k \geq 1$. Let $T_{\text{div}} = 1$ if the diversity function is remote-edge, remote-tree, remote-cycle, remote-trees or remote-t-cycles, and let $T_{\text{div}} = k$ if the diversity function is remote-clique, remote-star, remote-bipartition, remote-pseudoforest or remote-matching. For any $\epsilon \in (0, 0.5), \alpha \geq 1, \alpha \in [0, 1]$, there always exists an $\alpha$-restricted $O(\alpha - \epsilon)$-approximate bucketing-based sketch $Z^\alpha(\cdot)$ such that for any $P \subseteq [\Delta]^d$,

1. the recover function over $Z^\alpha(P)$ outputs an $O(\alpha - \epsilon)$-approximation to $\text{OPT}(P)$ if $\text{OPT}(P) \geq \alpha$;

2. if $\text{OPT}(P) \leq 2 \cdot \alpha$, the space budget of $Z^\alpha(P)$ is at most $k \cdot T_{\text{div}} \cdot O(\sqrt{d}/\epsilon)^d \cdot d$.

The time needed to compute a filter function, a bucketing function and a processing function on any $x \in \mathbb{R}^d$ is always upper bounded by $O(d)$. The time to compute recover function over $Z^\alpha(P)$ is at most $T(|Z^\alpha(P)|)$ where $T(n')$ denotes the running time needed to compute an $\alpha$-approximation for the diversity maximization for a point set with $n'$ points in $[\Delta]^d$.

**Proof.** We will prove that the sketch $Z(P)$ described in this section is the desired sketch $Z^\alpha(P)$.

According to Lemma 4.2, we can recover $S_\mu$ for $\mu = \epsilon \cdot O(10\sqrt{d})$ via $Z(P)$ in time linear in the size of $Z(P)$. If $\text{OPT}(P) \geq \alpha$, we have $\mu \leq \epsilon \cdot \text{OPT}(P)/(10\sqrt{d})$. According to Lemma 4.1, we have $\text{OPT}(P) \geq \text{OPT}(S_\mu) \geq (1 - \epsilon) \cdot \text{OPT}(P)$. According to Lemma 4.2, we have $|S_\mu| = |Z(P)|$. We can use $T(|Z(P)|)$ time to find a subset $S \subseteq S_\mu \subseteq P$ with $|S| = k$ such that $\text{OPT}(P) \geq \text{div}(S) \geq \alpha \cdot \text{OPT}(S_\mu) \geq \alpha (1 - \epsilon) \cdot \text{OPT}(P) \geq (\alpha - \epsilon) \cdot \text{OPT}(P)$. If $\text{OPT}(P) \leq 2 \cdot \alpha$, according to Lemma 4.1, we have $|S_\mu| \leq k \cdot T_{\text{div}} \cdot O(\sqrt{d}/\epsilon)^d$. Thus the time needed to compute recover function over $Z(P)$ is at most $T(|Z(P)|)$. According to Lemma 4.2, the space budget of $Z(P)$ is at most $O(|S_\mu| \cdot d)$, and thus the space budget of $Z(P)$ is at most $k \cdot T_{\text{div}} \cdot O(\sqrt{d}/\epsilon)^d \cdot d$.

Since each point has dimension $d$, the time to perform a filter function, a bucketing function and a processing function on $x$ is $O(d)$. □
4.3 Sliding Window Algorithm for Diversity Maximization

By plugging the bucketing-based sketch for diversity maximization into our algorithmic framework (Algorithm 1), we are able to obtain an efficient sliding window algorithm for diversity maximization.

**Theorem 4.2.** For any \( \epsilon \in (0,0.5), \alpha \in [0,1] \) and any diversity function listed in Table 7, there is a \((\alpha - \epsilon)\)-approximate algorithm for the diversity maximization problem with \( k \geq 1 \) for a point set from \( |\Delta|^d \) in the sliding window model with window size \( W \geq 1 \) using space \( kdt_{div} \log(d\Delta) \cdot O(\sqrt{d}/\epsilon)^d \) where \( t_{div} = 1 \) if the diversity function is remote-edge, remote-tree, remote-cycle, remote t-trees or remote t-cycles, and \( t_{div} = k \) if the diversity function is remote-clique, remote-star, remote-bipartition, remote-pseudoforest or remote-matching. The update time is at most \( O(d\log(d\Delta)) \). The running time to get the approximation at the end of the stream is at most \( T(k \cdot t_{div} \cdot O(\sqrt{d}/\epsilon)^d \cdot O(\log(d\Delta))) \) where \( T(n') \) denotes the running time needed to compute an \( \alpha \)-approximation for the diversity maximization for a point set with \( n' \) points in \( |\Delta|^d \).

**Proof.** Suppose the point set of interest at the end of the stream is \( P \subseteq |\Delta|^d \). Firstly, we need to handle the case if \( \text{OPT}(P) = 0 \). According to the definition of the diversity functions, \( \text{OPT}(P) = 0 \) can only happen when the number of distinct points in \( P \) is at most \( k - 1 \). We can use the following sliding window procedure to check whether \( P \) has at least \( k \) points, and if the number of distinct points is at most \( k - 1 \), we can retrieve either all points or at least \( T_{div} \) points at each distinct point.

1. Initialize a list of points \( L = \emptyset \).
2. For the latest \( x \) in the stream:
   (a) Add \( x \) into \( L \).
   (b) If there are \( T_{div} + 1 \) points in \( L \) that are equal to \( x \), remove the point which is equal to \( x \) with the earliest timestamp from \( L \).
   (c) Otherwise, if \( L \) contains \( kt_{div} + 1 \) points, remove the point with the earliest timestamp from \( L \).
3. At timestamp \( N \), let \( L' = \{ x \in L \mid \text{the timestamp is at least } N - W + 1, \text{ i.e., } x \in P \} \).
4. If \( L' \) has at most \( k - 1 \) distinct points, return \( \text{OPT}(L') = \text{OPT}(P) \).

It is easy to verify that \( L \) has the following two properties: 1) if the timestamp of \( x \) is earlier than the timestamp of \( y \) and \( x \) is in \( L \), then either \( y \) is in \( L \) or there are \( T_{div} \) points \( y' \) in \( L \) such that \( y' = y \) and the timestamp of every \( y' \) is later than the timestamp of \( y \); 2) For each \( x \in L \), the number of points \( x' \in L \) (including \( x \) itself) that is equal to \( x \) is at most \( T_{div} \). Since \( L \) can contain at most \( kT_{div} \) points and at most \( T_{div} \) duplicates can be stored in \( L \) for each distinct point, if \( P \) has at least \( k \) distinct points, then \( L \) must contain at least \( k \) distinct points. Next, we are going to prove that \( L' \) also contains at least \( k \) distinct points. Suppose \( x_{i_1}, x_{i_2}, \ldots, x_{i_k} \) with \( i_1 < i_2 < \cdots < i_k \) are \( k \) distinct points in the stream such that \( \forall j \geq i_j, x_j \in L \) in the stream is equal to \( x_{i_j} \) for some \( q \in [k] \), and \( \forall q \in [k], j > i_q, x_j \) is distinct from \( x_{i_q} \). Since \( P \) contains at least \( k \) distinct points, all \( x_{i_1}, x_{i_2}, \ldots, x_{i_k} \) must in \( P \), i.e., \( i_1 \geq N - W + 1 \). If \( \exists q \in [k] \) such that \( x_{i_q} \) is not in \( L \), then since \( L \) contains at least \( k \) distinct points, there must be \( x_j \in L \) with \( j < i_q \) which contradicts to the first property of \( L \). Thus, all \( x_{i_1}, x_{i_2}, \ldots, x_{i_k} \) must in \( L \) which implies that \( L' \) contains at least \( k \) distinct points \( x_{i_1}, x_{i_2}, \ldots, x_{i_k} \) since \( i_1 \geq N - W + 1 \). Next, we discuss the case when \( P \) contains at most \( k - 1 \) distinct points. For \( j \geq N - W + 1 \), if \( x_j \) is not in \( L' \), then \( x_j \) is not in \( L \). It implies that there are at least \( T_{div} j' \) with \( j' > j \) such that \( x_{j'} \) is equal to \( x_j \) and \( x_{j'} \) is in \( L' \) according to the first property of \( L \). Thus, for each distinct point in \( P, L' \) contains either all duplicates or at least \( T_{div} \) duplicates. According to the definition of diversity functions in Table 7, we can verify that \( \text{OPT}(L') = \text{OPT}(P) \).

The space needed is \( O(kT_{div}d) \), and the update time is \( O(d) \).

In the remaining of the proof, we only need to discuss the case when \( \text{OPT}(P) > 0 \). In this case, we have \( \text{OPT}(P) \in [1, \sqrt{d\Delta}] \). According to Theorem 4.1, \( \forall \alpha \in [1, \sqrt{d\Delta}], \alpha \in [0,1], \epsilon \in (0,0.5) \), there always exists an \( \alpha \)-restricted \((\alpha - \epsilon)\)-approximate bucketing-based sketch \( Z^\alpha() \) for diversity maximization such that an \((\alpha - \epsilon)\)-approximation to \( \text{OPT}(P) \) can be recovered if \( \text{OPT}(P) \geq \alpha \). Furthermore, if \( \text{OPT}(P) \leq 2 - \alpha \), the space budget of the sketch is at most \( k \cdot t_{div} \cdot O(\sqrt{d}/\epsilon)^d \cdot d \). According to Theorem 2.1, after plugging the bucketing-based sketch into Algorithm 1, we obtain the sliding window algorithm which outputs a \((\alpha - \epsilon)\)-approximation to \( \text{OPT}(P) \).

The space needed is at most \( k \cdot t_{div} \cdot O(\sqrt{d}/\epsilon)^d \cdot d \cdot \log(d\Delta) \). According to Theorem 4.1, the time needed to
compute a filter function, a bucketing function, and a processing function on any data item is always at most $O(d)$. By taking a more careful analysis, we can get an update time better than that claimed in Theorem 2.1. Consider an iteration of the loop in line 18-20 of Algorithm 1. We only need to compute filter function, bucketing function and processing function $O(\log(d\Delta))$ times since $|O| = O(\log(d\Delta))$ and $t = 2$. It takes $O(d \log(d\Delta))$ time. Since each tuple in the sketch has the same size $\Theta(d)$, the loop in line 14-16 of Algorithm 1 has at most one iteration each time. Similarly, the loop in line 18-21 of Algorithm 1 has at most one iteration each time. Thus, the overall update time is at most $O(d \log(d\Delta))$ time. At the end of the stream, to get the final approximation, we need to evaluate the recover function $O(\log(d\Delta))$ times according to Algorithm 1. Since each sketch contains at most $k \cdot T_{div} \cdot O(\sqrt{d}/\epsilon)^d$ points, the time needed to compute the recover function for each sketch is at most $T(k \cdot T_{div} \cdot O(\sqrt{d}/\epsilon)^d)$. Thus, the overall running time to get the approximation at the end of the stream is at most $T(k \cdot T_{div} \cdot O(\sqrt{d}/\epsilon)^d) \cdot O(\log(d\Delta))$.

5 k-Clustering in the Sliding Window Model

In the $\ell_p$ $k$-clustering problem ($p \in [1, \infty)$), we consider the data universe $X$ as the (discretized) $d$-dimensional Euclidean space $[\Delta]^d$. Given a point set $X \subseteq [\Delta]^d$ and a parameter $k \in \mathbb{Z}_{\geq 1}$, the goal is to find a subset of $k$ centers $B \subseteq \mathbb{R}^d$ ($|B| \leq k$) such that the clustering cost

$$\text{cost}(X, B) := \sum_{x \in X} \min_{b \in B} \|x - b\|_2^p$$

is minimized. Notice that this problem formulation is a general case of $k$-median clustering ($p = 1$) and $k$-means clustering ($p = 2$). One popular way to solve $k$-clustering problem in large-scale computational models is to use the coreset technique. A coreset is a small weighted subset of data points which approximately preserves the clustering cost for any set of centers. The formal definition of $\epsilon$-coreset is defined as the following.

**Definition 5.1. ($\epsilon$-coreset)** Let $k \in \mathbb{Z}_{\geq 1}, p \in [1, \infty)$. Given a set of points $X \subseteq [\Delta]^d$, if a subset $S \subseteq X$ together with the weights $w : S \rightarrow \mathbb{R}_{\geq 0}$ satisfies that

$$\forall B \subset \mathbb{R}^d \text{ with } |B| \leq k, (1 - \epsilon) \cdot \text{cost}(X, B) \leq \text{cost}(S, w, B) \leq (1 + \epsilon) \cdot \text{cost}(X, B),$$

where $\text{cost}(S, w, B) := \sum_{x \in S} w(x) \cdot \min_{b \in B} \|x - b\|_2^p$, then $(S, w)$ is called an $\epsilon$-coreset of $X$ for the $\ell_p$ $k$-clustering problem.

In the remaining subsections, we will first review an offline coreset construction algorithm and then we will show how to construct a bucketing-based sketch (see Definition 2.1) for an $\epsilon$-coreset. Thus, it will imply an efficient sliding window algorithm.

5.1 Offline Coreset Construction

The offline construction is almost the same as the algorithm proposed by [HSYZ18]. We put all analysis into Appendix B for completeness.

Suppose the data set is a set of at most $n$ points $X \subseteq [\Delta]^d$. We first partition the space $[\Delta]^d$ via a hierarchical grid structure [Che09, BFL+17, HSYZ18]. We sample a random vector $v \in \mathbb{R}^d$ such that each entry $v(i)$ is drawn independently and uniformly at random from $[0, \Delta]$ Then we impose a standard hierarchical grids shifted by $-v$. Let $L = \lceil \log(n d \Delta) \rceil + 10$. The grids have $L + 1$ levels. $\forall i \in \{0, 1, \cdots, L\}$, the grid $G_i$ partitions $\mathbb{R}^d$ into cells with side length $\Delta_i = \Delta/2^i$. In particular,

$$\forall i \in \{0, 1, \cdots, L\}, \quad G_i = \{C \mid C = [t_1 \Delta_i - v(1), (t_1 + 1) \Delta_i - v(1)] \times \cdots \times [t_d - v(d) \Delta_i, (t_d + 1) \Delta_i - v(d)], t_1, \cdots, t_d \in \mathbb{Z}\}.$$ 

It is easy to see that each cell $C \in G_i$ is always partitioned by $2^d$ cells in $G_{i+1}$. For convenience in notation, we also define $G_{-1}$ in the same way where each cell in $G_{-1}$ has side length $\Delta_{-1} = 2\Delta$. It is easy to verify that there is a unique cell in $G_{-1}$ which contains $[\Delta]^d$ entirely. Consider two cells $C \in G_i$ and $C' \in G_j$. If $C \subseteq C'$, we call $C'$ an ancestor of $C$. If $C$ is a cell in $G_i$ for $i \in \{0, 1, \cdots, L\}$ and a cell $C' \in G_{i-1}$ is an ancestor of $C$, then we say that $C$ is a child cell of $C'$. Consider a point $x \in [\Delta]^d$. If the cell containing $x$ in $G_i$ is $C$, we define $c_i(x) := C$. 

3019 Copyright © 2022 by SIAM
Unauthorized reproduction of this article is prohibited
Let OPT denote the optimal \( l_p \) k-clustering cost for a point set \( X \), i.e., \( \text{OPT} := \min_{B \subseteq \mathbb{R}^d : |B| \leq k} \text{cost}(X, B) \). The \( \epsilon \)-coreset construction is shown in Algorithm 3 which uses Algorithm 2 as a subroutine.

The guarantee of Algorithm 3 is shown in the following theorem. We include the proof of Theorem 5.1 in Appendix B for completeness.

**Theorem 5.1. (Generalization of Theorem 8 of [HSYZ18])** Suppose for every \( i \in \{0, 1, \cdots, L\} \) and every cell \( C \in G_i \), the estimated value \( \lambda(C) \) in line 7 of Algorithm 3 satisfies \( \lambda(C) \in |C \cap X| \pm 0.1 R_i \) or \( \lambda(C) \in (1 \pm 0.01) |C \cap X| \), and for every \( i \in \{0, 1, \cdots, L\} \), the estimated value \( \lambda(X_i) \) in line 8 of Algorithm 3 satisfies \( \lambda(X_i) \in |X_i| \pm 0.1 \gamma R_i \) or \( \lambda(X_i) \in (1 \pm 0.01) \epsilon \cdot |X_i| \). If the parameter \( o \) in Algorithm 3 satisfies \( o \in [1, \text{OPT}] \), then \((S, w)\) outputted by Algorithm 3 is an \( \epsilon \)-coreset of \( X \) for the \( l_p \) k-clustering problem with probability at least \( 1 - \delta \).

### 5.2 Coreset Construction via bucketing-based Sketches

To simulate Algorithm 2 and Algorithm 3 in the sliding window model, we only need to construct a bucketing-based sketch to simulate the algorithms.

#### 5.2.1 Heavy Cell Partitioning via Bucketing-based Sketches

We first describe a bucketing-based sketch to simulate Algorithm 2. Our sketch \( Z(\cdot) \) is composed by \( L + 1 \) sub-sketches \( Z_0(\cdot), Z_1(\cdot), \cdots, Z_L(\cdot) \). Let \( \delta \in (0, 1) \) be a probability parameter. To describe \( Z_i(\cdot) \) for \( i \in \{0, 1, \cdots, L\} \), we need to specify the filter function \( h_i(\cdot) \), the bucketing function \( g_i(\cdot) \), the processing function \( \zeta_i(\cdot) \) and the threshold \( T_i \) (see Definition 2.1):

1. For all \( x \in |\Delta|^d \), \( h_i(x) \) are independent, and

\[
h_i(x) = \begin{cases} 
1, & \text{w.p. } p_i = \min(1, 10^6 \ln(nL/\delta)/R_i), \\
0, & \text{o.w.}
\end{cases}
\]

2. Let \( B \) be all cells of randomly shifted grids generated by Algorithm 2, i.e., \( B = G_0 \cup G_1 \cup \cdots \cup G_L \). Then, \( g_i(x) \) is the cell in \( G_i \) that contains \( x \), i.e.,

\[
\forall x \in |\Delta|^d, g_i(x) = C : C \in G_i, x \in C.
\]
3. \( \zeta_i(x) = x \).

4. \( T_i = 10 \cdot p_i \cdot R_i \cdot d. \)

Lemma 5.1. (Simulation of Algorithm 2 via bucketing-based sketch) Algorithm 2 can be simulated via sketch \( Z(X) \) described above. Furthermore, with probability at least \( 1 - \delta \), \( \forall i \in \{0, 1, \cdots, L\}, C \in G_i \), the estimated value \( \lambda(C) \) in line 7 of Algorithm 2 during the simulation satisfies \( \lambda(C) \in |C \cap X| \pm 0.1R_i \) or \( \lambda(C) \in (1 \pm 0.01) \cdot |C \cap X| \). The running time of the simulation is \( O(|Z(X)|) \).

Proof. In the proof, we will show how to use \( Z(X) \) to simulate Algorithm 2. Let us define \( \forall i \in \{0, 1, \cdots, L\}, C \in G_i, \lambda(C) := |\{x \in X \mid h_i(x) = 1, g_i(x) = C\}|/p_i \). Notice that we do not need to explicitly compute \( \lambda(C) \) during the simulation of Algorithm 2.

We first prove that \( \lambda(C) \) is a good approximation to \( |C \cap X| \). If \( p_i = 1 \), it is obvious that \( \lambda(C) = |C \cap X| \). We only need to consider the case when \( p_i < 1 \). Consider \( i \in \{0, 1, \cdots, L\} \) and cell \( C \in G_i \). We have \( \mathbb{E}(|\{x \in X \mid h_i(x) = 1, g_i(x) = C\}|) = p_i \cdot |C \cap X| \). There are two cases. If \( |C \cap X| > R_i \), by Bernstein inequality, we have:

\[
\Pr \left[ \left| \{x \in X \mid h_i(x) = 1, g_i(x) = C\} \right| - p_i \cdot |C \cap X| \geq 0.01 \cdot p_i \cdot |C \cap X| \right] \\
\leq 2 \exp \left( -\frac{0.5 \cdot 0.01^2 \cdot p_i^2 \cdot |C \cap X|^2}{\frac{1}{3} \cdot 0.01 \cdot p_i \cdot |C \cap X|} \right) \\
\leq \delta/(n(L + 1)),
\]

where the last inequality follows from that \( |C \cap X| > R_i \) and \( p_i \geq 10^6 \ln(nL/\delta)/R_i \). In the second case, we have \( |C \cap X| \leq R_i \). In this case, by Bernstein inequality, we have:

\[
\Pr \left[ \left| \{x \in X \mid h_i(x) = 1, g_i(x) = C\} \right| - p_i \cdot |C \cap X| \geq 0.1 \cdot p_i \cdot R_i \right] \\
\leq 2 \exp \left( -\frac{0.5 \cdot 0.1^2 \cdot p_i^2 \cdot R_i^2}{p_i \cdot |C \cap X| + \frac{1}{3} \cdot 0.1 \cdot p_i \cdot R_i} \right) \\
\leq \delta/(n(L + 1)),
\]

where the last inequality follows from that \( |C \cap X| \leq R_i \) and \( p_i \geq 10^6 \ln(nL/\delta)/R_i \). Notice that \( \lambda(C) = |\{x \in X \mid h_i(x) = 1, g_i(x) = C\}|/p_i \). Thus, by taking union bound over all \( i \in \{0, 1, \cdots, L\} \) and all non-empty cells \( C \in G_i \), with probability at least \( 1 - \delta \), \( \forall i \in \{0, 1, \cdots, L\}, C \in G_i \), \( \lambda(C) \) satisfies \( \lambda(C) \in |C \cap X| \pm 0.1R_i \) or \( \lambda(C) \in (1 \pm 0.01) \cdot |C \cap X| \).

Now let us go back to Algorithm 2. All steps of Algorithm 2 can be implemented without any information of \( X \) except line 7 and line 8. Let us focus on line 7 and line 8 of Algorithm 2. The goal to determine whether \( \lambda(C) \geq R_i \). Since \( \lambda(C) = |\{x \in X \mid h_i(x) = 1, g_i(x) = C\}|/p_i \), it is equivalent to determine whether \( \lambda(C) \geq 1/p_i \cdot R_i \). Let \( \lambda(C) \) be \( 1/p_i \) times the number of tuples \( (x, \zeta_i(x)) \) stored in \( Z_i(X) \). Notice that \( \lambda(C) \) can be computed during the simulation via the sketch \( Z(X) \). Consider the condition 2b of Definition 2.1. There are two cases. In the first case, \( \sum_{x \in X : h_i(x) = 1, g_i(x) = C} \zeta_i(x) \leq T_i \). In this case, \( \forall x \in X \) with \( h_i(x) = 1, g_i(x) = C \), we have \( (x, \zeta_i(x)) \in Z_i(x) \). Therefore, we have \( \lambda(C) \leq \lambda(C) \) and thus line 7 and line 8 of Algorithm 2 can be successfully simulated. In the second case, \( \sum_{x \in X : h_i(x) = 1, g_i(x) = C} \zeta_i(x) > T_i \). By our definition of \( \lambda(C) \), it implies that \( \lambda(C) \geq R_i \). In this case, it is also easy to verify that \( \lambda(C) \geq R_i \) according to the condition 2b of Definition 2.1. Thus, the outcome of using \( \lambda(C) \) to simulate line 7 and line 8 of Algorithm 2 is always the same as the outcome of using \( \lambda(C) \).

Finally let us consider the running time of the simulation process. The overall running time is linear in the number of times that line 7 and line 8 are executed. Observe that each tuple \( (x, \zeta_i(x)) \in Z_i(X) \) can be considered at most once. The total running time is at most linear in the size of \( Z(X) \).

Lemma 5.2. (Size of the sketch \( Z(X) \)) With probability at least \( 1 - 2\delta \), the space budget of \( Z(X) \) is at most \( O((kd + d^{O(p)}) \cdot \text{OPT} / \alpha) \cdot L \log(nL/\delta)/\delta) \).
Before proving Lemma 5.2, let us prove the following useful lemma. Let $B^* \subset \mathbb{R}^d$ be the optimal centers for $X$, i.e., $\text{cost}(X, B^*) = \text{OPT}$. A center cell $C \in G_i$ in level $i \in \{0, 1, \ldots, L\}$ is a cell such that $\inf_{x \in C,y \in B^*} \|x - y\|_2 \leq \Delta_i/(2d)$. The following Lemma shows that the total number of center cells cannot be too large since the grids are randomly shifted. This is an observation made by [Che09, BFL+17, HSYZ18]. We include the proof here for completeness.

**Lemma 5.3.** With probability at least $1 - \delta$, the total number of center cells is at most $6kL/\delta$.

**Proof.** Let $B^* = \{b_1, b_2, \ldots, b_t\}$. Consider the grid $G_i$ in level $i \in \{0, 1, \ldots, L\}$. Let $Y_{j,\alpha}$ denote an indicator variable of the event that the Euclidean distance from $b_j$ to the boundary of $G_i$ in the $\alpha$-th dimension is at most $\Delta_i/(2d)$. Notice that if a center is close to a boundary of $G_i$ in a dimension, it will contribute a factor of at most 2 to the number of center cells. Thus, the number of cells that have distance to $b_j^*$ at most $\Delta_i/(2d)$ is at most

$$2^{\sum_{\alpha=1}^{d} Y_{j,\alpha}}.$$

Notice that

$$\mathbb{E}\left[2^{\sum_{\alpha=1}^{d} Y_{j,\alpha}}\right] = \prod_{\alpha=1}^{d} \mathbb{E}[2^{Y_{j,\alpha}}] = (1 + 1/d)^d = e \leq 3,$$

where the second inequality follows from that $\Pr[Y_{j,\alpha} = 1] = (2 \cdot \Delta_i/(2d))/\Delta_i = 1/d$ and $2^{Y_{j,\alpha}} = 1 + Y_{j,\alpha}$. Since there are $k$ centers, the expected total number of center cells in $G_i$ is at most $3k \cdot 3 = 9k$. Since there are $L + 1$ levels, the expected total number of center cells over all levels is at most $3k \cdot (L + 1) \leq 6kL$. By Markov’s inequality, with probability at least $1 - \delta$, the number of center cells in all levels is at most $6kL/\delta$. \qed

**Proof of Lemma 5.2.** We are going to prove Lemma 5.2. The space budget of $Z(X)$ can be upper bounded as the following:

The space budget of $Z(X)$

$$= \sum_{i=0}^{L} \sum_{C \in G_i} \min\left(T_i, \sum_{x \in X : h_i(x) = 1, g_i(x) = C} |G_i(x)|\right)$$

$$\leq \sum_{i=0}^{L} \sum_{C \in G_i} T_i + \sum_{C \in G_i : C \text{ is not a center cell}} \{\{x \in X \mid h_i(x) = 1, g_i(x) = C\}\cdot d\}$$

(5.1)

$$= \sum_{i=0}^{L} \sum_{C \in G_i : C \text{ is a center cell}} T_i$$

(5.2)

$$+ d \cdot \sum_{i=0}^{L} \sum_{x \in X : x \text{ is not in any center cell in } G_i} h_i(x).$$

According to Lemma 5.3, part (5.1) is at most $6kL/\delta \cdot 10^7 \ln(nL/\delta) \cdot d$ with probability at least $1 - \delta$. In the remaining of the proof let us bound part (5.2).

Consider $i \in \{0, 1, \ldots, L\}$. If $x \in X$ is not in any center cell in $G_i$, then $\min_{b \in B^*} \|x - b\|_2 \geq \Delta_i/(2d)$. Thus, $|\{x \in X \mid x \text{ is not in any center cell in } G_i\}| \leq (\text{OPT})^{\Delta_i/(2d)p} \cdot (2d^{1.5})^p \cdot R_i \cdot (\text{OPT})^{\Delta_i/(2d)p}$ where the last inequality follows from that $R_i = 0.01n/\sqrt{d\Delta_i}$. Since $p_i = \min(1, 10^6 \ln(nL/\delta)/R_i)$, we have

$$\mathbb{E}\left[\sum_{x \in X : x \text{ is not in any center cell in } G_i} h_i(x)\right] \leq 10^8 \cdot (2d^{1.5})^p \cdot \ln(\text{OPT}/\delta).$$

By Markov’s inequality, with probability at least $1 - \delta$, we have:

$$\sum_{i=0}^{L} \sum_{x \in X : x \text{ is not in any center cell in } G_i} h_i(x) \leq 10^8 \cdot (2d^{1.5})^p \cdot (L + 1) \cdot \ln(\text{OPT}/\delta).$$

Copyright © 2022 by SIAM

Unauthorized reproduction of this article is prohibited
Thus, we can bound part (5.2) by $2 \cdot 10^8 \cdot (2d^{1.5}) \cdot dL \cdot \ln(nL/\delta)/\delta \cdot OPT$. By combining the part (5.1) with the part (5.2), we conclude the proof. □

5.2.2 Sampling Process via Bucketing-based Sketches Now we will describe the simulation of Algorithm 3 via bucketing-based sketches. We divided Algorithm 3 into two parts. The first part contains line 1-7 of Algorithm 3. The second part contains remaining steps of Algorithm 3.

Let us first consider the first part. In Section 5.2.1, we described how to simulate Algorithm 2 via bucketing-based sketches. Thus, to simulate line 1-7 of Algorithm 3, we only need to obtain $\lambda(X^i)$ for all $i \in \{0,1,\cdots,L\}$ via bucketing-based sketches. We describe a bucketing-based sketch $Z{\cdot}$ as the following. Our sketch $Z{\cdot}$ is composed by $L+1$ sub-sketches $Z_0{\cdot},Z_1{\cdot},\cdots,Z_L{\cdot}$. Let $\delta \in (0,0.5)$ be a probability parameter. Let $\epsilon \in (0,0.5)$ be an error parameter. To describe $Z_i{\cdot}$ for $i \in \{0,1,\cdots,L\}$, we need to specify the filter function $\wt h_i{\cdot}$, the bucketing function $g_i{\cdot}$, the processing function $\rho_i{\cdot}$, and the threshold $T'_i$ (see Definition 2.1):

1. $\forall x \in [\Delta]^d, h'_i(x)$ are independent, and

$$ h'_i(x) = \begin{cases} 1, & \text{w.p. } p'_i = \min(1, 10^8 \ln(nL/\delta)/(\epsilon^2\gamma R_i)), \\ 0, & \text{o.w.} \end{cases} $$

2. Let $B$ be all cells of grids described by Algorithm 3 i.e., $B = G_0 \cup G_1 \cup \cdots \cup G_L$. Then $g'_i(x)$ is the cell in $G_i$ that contains $x$, i.e.,

$$ \forall x \in [\Delta]^d, g'_i(x) = C : C \in G_i, x \in C. $$

3. $\zeta'_i(x) = x$.

4. $T'_i = 10 \cdot p'_i \cdot R_i \cdot d$.

Note that $Z'\cdot$ is very similar to $Z\cdot$. The main differences are that $h'_i$ has a higher sampling rate than $h_i$ and $T'_i > T_i$.

**Lemma 5.4. (Obtaining $\lambda(X^i)$ of Algorithm 3 via bucketing-based sketch)** If during the simulation of Algorithm 2 in line 2 of Algorithm 3, $\forall i \in \{0,1,\cdots,L\}, C \in G_i$ $\lambda(C)$ satisfies $\lambda(C) \in |C \cap X| \pm 0.1 R_i$ or $\lambda(C) \in (1 \pm 0.01) \cdot |C \cap X|$, then $\forall i \in \{0,1,\cdots,L\}, \lambda(X^i)$ can be obtained via $Z(X)$ such that with probability at least $1-2\delta$, $\forall i \in \{0,1,\cdots,L\}, \lambda(X^i) \in |X^i| \pm 0.1 \epsilon R_i$ or $\lambda(X^i) \in (1 \pm 0.01 \epsilon) \cdot |X^i|$. Line 1-7 of Algorithm 3 can be simulated once $\lambda(X^i)$ for all $i \in \{0,1,\cdots,L\}$ are obtained. The running time to simulate line 1-7 of Algorithm 3 is at most $O(|Z(X)| + L)$.

**Proof.** Consider $i \in \{0,1,\cdots,L\}$ and $C \in G_i$. If $C$ is a crucial cell, since $\lambda(C)$ is a good estimation of $|C \cap X|$, we know that $|C \cap X| \leq 1.1 R_i$. Notice that $E[\{x \in X \mid h'_i(x) = 1, g'_i(x) = C\}] = p'_i \cdot |C \cap X|$. By Bernstein inequality, we have:

$$ \Pr \left[ \|x \in X \mid h'_i(x) = 1, g'_i(x) = C\| - p'_i \cdot |C \cap X| \geq 0.1 \cdot p'_i \cdot R_i \right] \leq \exp \left( -\frac{0.5 \cdot 0.1^2 \cdot p'_i^2 \cdot R_i^2}{\frac{1}{3} \cdot 0.1 \cdot p'_i \cdot R_i} \right) \leq \delta/(nL + 1), $$

where the last inequality follows from that $|C \cap X| \leq 1.1 R_i$ and $p'_i \geq 10^4 \ln(nL/\delta)/R_i$. Thus, for a crucial cell $C$, with probability at least $1 - \delta/(nL + 1)$, $\|x \in X \mid h'_i(x) = 1, g'_i(x) = C\| \leq 1.1 \cdot p'_i \cdot R_i$. Thus, by taking union bound over all non-empty crucial cells, with probability at least $1 - \delta$, $\forall i \in \{0,1,\cdots,L\}, \forall C \in G_i$ that is a crucial cell, $\|x \in X \mid h'_i(x) = 1, g'_i(x) = C\| \leq 1.1 \cdot p'_i \cdot R_i$. Thus, with probability at least $1 - \delta$, $\forall i \in \{0,1,\cdots,L\}, \forall C \in G_i$ that is a crucial cell, $\sum_{x \in X, h'_i(x) = 1, g'_i(x) = C} |\zeta'_i(x)| \leq T'_i$. According to condition 2.1 of Definition 2.1, and the construction of $X^i$ in Algorithm 3, it implies that with probability at least $1 - \delta$, $\forall i \in \{0,1,\cdots,L\}, \forall x \in X^i$, if $h'_i(x) = 1$, then $x, \zeta'_i(x) \in Z'_i(X)$.

Notice that for each $(x, \zeta'_i(x)) \in Z'_i(X)$, we are able to know whether $x \in X^i$ since we know all heavy cells. By the construction of $X^i$, $x \in X^i$ if and only if $x$ is not in a heavy cell of $G_i$ and $x$ is in a heavy cell of $G_{i-1}$. Thus,
we are able to compute \( \lambda(X^i) = |\{(x, \zeta_i(x)) \in Z'_i(X) \mid x \in X^i\}|/p'_i = |\{x \in X^i \mid h'_i(x) = 1\}|/p'_i. \) Next, we will show that \( \lambda(X^i) \) is a good approximation to \( |X^i| \). There are two cases. If \( |X^i| > \gamma R_i \), by Bernstein inequality, we have:

\[
\Pr[|\{x \in X^i \mid h'_i(x) = 1\}| - p'_i \cdot |X^i| \geq 0.01 \cdot \epsilon \cdot p'_i \cdot |X^i|] \\
\leq 2 \exp \left(- \frac{0.5 \cdot 0.01^2 \cdot \epsilon^2 \cdot p'_i^2 \cdot |X^i|^2}{p'_i \cdot |X^i| + \frac{1}{3} \cdot 0.01 \cdot \epsilon \cdot p'_i \cdot |X^i|} \right) \\
\leq \delta/(L + 1),
\]

where the last inequality follows from that \( |X^i| > \gamma R_i \) and \( p'_i \geq 10^8 \ln(L/\delta)/(\epsilon^2 \gamma R_i). \) In the second case, we have \( |X^i| \leq \gamma R_i \). In this case, by Bernstein inequality, we have:

\[
\Pr[|\{x \in X^i \mid h'_i(x) = 1\}| - p'_i \cdot |X^i| \geq 0.1 \cdot \epsilon \cdot p'_i \cdot \gamma \cdot R_i] \\
\leq \exp \left(- \frac{0.5 \cdot 0.1^2 \cdot \epsilon^2 \cdot p'_i^2 \cdot \gamma^2 \cdot R_i^2}{p'_i \cdot |X^i| + \frac{1}{3} \cdot 0.1 \cdot \epsilon \cdot p'_i \cdot \gamma \cdot R_i} \right) \\
\leq \delta/(L + 1),
\]

where the last inequality follows from that \( |X^i| \leq \gamma R_i \) and \( p'_i \geq 10^8 \ln(L/\delta)/(\epsilon^2 \gamma R_i). \) Thus, by taking union bound over all \( i \in \{0, 1, \ldots, L\} \), with probability at least \( 1 - \delta \), \( \forall i \in \{0, 1, \ldots, L\} \), \( \lambda(X^i) \) satisfies \( \lambda(X^i) \in |X^i| \pm 0.1 \epsilon \gamma R_i \) or \( \lambda(X^i) \in (1 \pm 0.01\epsilon) |X^i| \).

By using such \( \lambda(X^i) \), we are able to simulate line 3-line 7 of Algorithm 3. It is easy to verify that we only need to scan each item in \( Z'(X) \) at most once to compute \( \lambda(X^i) \) for all \( i \in \{0, 1, \ldots, L\} \). Thus the total running time to simulate line 3-line 7 of Algorithm 3 is at most \( O(L + |Z'(X)|) \).

**Lemma 5.5. (Size of \( Z'(X) \))** With probability at least \( 1 - 2\delta \), the space budget of \( Z'(X) \) is at most \( O(kd + d^{O(p)} \cdot \OPT/\omega \cdot 2^{O(p)} L^2 \log(nL/\delta)/(\epsilon d^3)) \).

**Proof.** The proof is similar to the proof of Lemma 5.2. We will still use the concept of center cells (see Lemma 5.3).

The space budget of \( Z'(X) \):

\[
\sum_{i=0}^{L} \sum_{C \in G_i} \min \left( T'_i, \sum_{x \in X \mid h'_i(x) = 1, g'_i(x) = C} |\zeta'_i(x)| \right) \\
\leq \sum_{i=0}^{L} \left( \sum_{C \in G_i; C \text{ is a center cell}} T'_i + \sum_{C \in G_i; C \text{ is not a center cell}} |\{x \in X \mid h'_i(x) = 1, g'_i(x) = C\}| \cdot d \right) \\
= \sum_{i=0}^{L} \sum_{C \in G_i; C \text{ is a center cell}} T'_i \\
+ d \cdot \sum_{i=0}^{L} \sum_{x \in X \mid x \text{ is not in any center cell in } G_i} h'_i(x).
\]

According to Lemma 5.3 part (5.3) is at most \( 6kL/\delta \cdot 10^7 \ln(nL/\delta)/(\epsilon d^2) \cdot d \) with probability at least \( 1 - \delta \). In the remaining of the proof let us upper bound part (5.4).

Recall that \( B^* \) is the optimal centers for \( X \), i.e. \( B^* \) satisfies \( \text{cost}(X, B^*) = \OPT \). Consider \( i \in \{0, 1, \ldots, L\} \). If \( x \in X \) is not in any center cell in \( G_i \), then \( \min_{b \in B^*} \|x - b\|_2 \geq \Delta_i/(2d) \). Thus, \( |\{x \in X \mid x \text{ is not in any center cell in } G_i\}| \leq \frac{\OPT}{(2\Delta_i/(2d))^p} \leq 100 \cdot (2d^{1.5})^p \cdot R_i \cdot \frac{\OPT}{\omega} \), where the last inequality follows from that \( R_i = 0.01\omega/(\sqrt{d} \Delta_i)^p \). Since \( p'_i = \min(1, 10^8 \ln(nL/\delta)/(\epsilon d^2 R_i)) \), we have

\[
E \left[ \sum_{x \in X \mid x \text{ is not in any center cell in } G_i} h'_i(x) \right] \leq 10^8 \cdot (2d^{1.5})^p \cdot \ln(nL/\delta)/(\epsilon d^2) \cdot \frac{\OPT}{\omega}.
\]

\[\text{E}
\]
By Markov’s inequality, with probability at least $1 - \delta$, we have:
\[
\sum_{i=0}^{L} \sum_{x \in X_i : x \text{ is not in any center cell in } G_i} h'_i(x) \leq 10^8 \cdot (2d^{1.5})^p \cdot \ln(nL/\delta)/(\epsilon^2\gamma) \cdot \frac{\OPT}{o} \cdot (L + 1)/\delta.
\]
Thus, we can upper bound part (5.4) by $10^8 \cdot (2d^{1.5})^p \cdot \ln(nL/\delta)/(\epsilon^2\gamma) \cdot \frac{\OPT}{o} \cdot d(L + 1)/\delta$. By combining with part (5.3) with part (5.4), and since $\gamma = \Theta(\epsilon/(2^{O(p)}L))$, we conclude the proof. $\square$

In the remaining of the section, we will describe how to use a bucketing-based sketch to simulate the sampling procedure shown in line 8-12 of Algorithm 3. The most challenging part is line 10 since each time we need to draw an independent uniform sample from $X^i$ while we cannot explicitly store the entire $X^i$. We use the following idea to draw a uniform sample from $X^i$: we choose a small proper sampling probability $p_i^\prime$ and we construct independent random subsets $Y^i_1, Y^i_2, \ldots, Y^i_m \subseteq X^i$, where $\forall j \in [m]$, each $x \in X^i$ is independently added into $Y^i_j$ with probability $p^\prime_i$. Since $p_i^\prime$ is very small, $Y^i_j$ may be an empty set. But for a non-empty set $Y^i_j$, if we choose a uniform sample from $Y^i_j$, then such sample is also a uniform sample from $X^i$. Thus, when each time we want to draw an independent uniform sample from $X^i$, we just choose an arbitrary unused non-empty set $Y^i_j$ and draw a uniform sample from $Y^i_j$ as a uniform sample from $X^i$. Since $p_i^\prime$ is properly chosen, we may afford to maintain all the subsets $Y^i_1, Y^i_2, \ldots, Y^i_m$ via our sketch and the number of non-empty subsets is large enough. In the following, we formalize the above idea by first introducing an another bucketing-based sketch $Z''(\cdot)$.

Our sketch $Z''(\cdot)$ is composed by $L + 1$ sub-sketches $Z''_0(\cdot), Z''_1(\cdot), \ldots, Z''_L(\cdot)$ where $Z''_i(\cdot)$ will be used to handle the uniform sampling over $X^i$. Let $\delta \in (0, 0.5)$ be a probability parameter. Let $\epsilon \in (0, 0.5)$ be an error parameter. To describe $Z''_i(\cdot)$ for $i \in \{0, 1, \ldots, L\}$, we need to specify the filter function $h''_i(\cdot)$, the bucketing function $g''_i(\cdot)$, the processing function $\zeta''_i(\cdot)$ and the threshold $T''_i(\cdot)$ (see Definition 2.1):

1. $\forall x \in [\Delta]^d$, $h''_i(x)$ is determined by $\zeta''_i(x)$: $h''_i(x) = 1$ if $\zeta''_i(x) \neq \emptyset$; $h''_i(x) = 0$ otherwise.
2. Let $B$ be all cells of grids described by Algorithm 3, i.e., $B = G_0 \cup G_1 \cup \cdots \cup G_L$. Then $g''_i(x)$ is the cell in $G_i$ that contains $x$, i.e.,
   \[
   \forall x \in [\Delta]^d, g''_i(x) = C : C \in G_i, x \in C.
   \]
3. Let
   \[
   \hat{m} = 10^9 \cdot 2^{2p+2} \cdot \epsilon^{-3} \cdot (kL + (2d^{1.5})^p) \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot \frac{L}{\delta}.
   \]
   Let $\zeta''_i(x)$ be a random subset of $[\hat{m}]$ such that $\forall j \in [\hat{m}], j$ is added into $\zeta''_i(x)$ independently with probability $p''_i = \min((2000 \cdot (kL + (2d^{1.5})^p) \cdot R_i)^{-1}, 1)$.
4. $T''_i = 10 \cdot p''_i \cdot \hat{m} \cdot R_i \cdot d$.

We describe the simulating process of line 8-12 via $Z''(X)$ as the following:

1. For each $i \in \{0, 1, \ldots, L\}, j \in [\hat{m}]$, construct $\hat{Y}^i_j = \{x \in X^i \mid \{x, \zeta''_i(x)\} \in Z''(X), j \in \zeta''_i(x)\}$. Notice that we are able to verify whether $x \in X^i$ given all heavy cells. Due to the construction of $X^i$, $x \in X^i$ if and only if $x$ is in a non-heavy cell in $G_i$ and is in a heavy cell in $G_{i-1}$.
2. Initialize $A^0 = A^1 = \cdots = A^L = [\hat{m}]$.
3. Repeat $m$ times:
   \[(a) \text{ Sample a level } i \in I \text{ with probability } \lambda(X^i) \cdot \min(2^{2p+1}/R_i, 1)/t'.' \\
   (b) \text{ Pick an arbitrary } j \in A^i \text{ satisfying } \hat{Y}^i_j \neq \emptyset. \text{ If such } j \text{ does not exist, return } \text{FAIL}.
   \[(c) \text{ Uniformly sample a point } x \in \hat{Y}^i_j \text{ and remove } j \text{ from } A^i, \text{ i.e., } A^i \leftarrow A^i \setminus \{j\}.
   \[(d) \text{ Add the point } x \text{ into } S \text{ and set the weight } w(x) := t'/(m \cdot \min(2^{2p+1}/R_i, 1)).\]
For the convenience of notation in our analysis, for \( i \in \{0, 1, \cdots, L\}, j \in [\hat{m}] \), we define \( Y^i_j = \{ x \in X^i \mid j \in \zeta''(x) \} \). It is clear that if \( Y^i_j \) is non-empty, then a uniform sample from \( Y^i_j \) is a uniform sample for \( X^i \). Notice that \( \hat{Y}^i_j \subseteq Y^i_j \), a uniform sample from \( \hat{Y}^i_j \) is a uniform sample from \( X^i \) only when \( \hat{Y}^i_j = Y^i_j \).

**Lemma 5.6.** (\( \hat{Y}^i_j = Y^i_j \) WITH A GOOD PROBABILITY) If line \( 17 \) of Algorithm 3 are simulated such that \( \forall i \in \{0, 1, \cdots, L\}, C \in G_i \), \( \mathcal{V}(C) \in |C \cap X| \pm 0.1R_i \) or \( \mathcal{V}(C) \in (1 \pm 0.01) \cdot |C \cap X| \) and \( \forall i \in \{0, 1, \cdots, L\} \), \( \lambda(X^i) \in |X^i| \pm 0.1e\gamma R_i \) or \( \lambda(X^i) \in (1 \pm 0.01e) \cdot |X^i| \), then with probability at least \( 1 - \delta \), \( \forall i \in \{0, 1, \cdots, L\}, j \in [\hat{m}] \), \( \hat{Y}^i_j = Y^i_j \).

**Proof.** Suppose all \( \lambda(C) \) and \( \lambda(X^i) \) are good enough as claimed in the statement.

Consider \( i \in \{0, 1, \cdots, L\} \) and a crucial cell \( C \in G_i \). Since \( \lambda(C) \) is a good estimation of \( |C \cap X| \), we know that \( |C \cap X| \leq 1.1R_i \). Notice that

\[
\mathbb{E} \left[ \sum_{x \in X : h''(x) = 1, g''(x) = C} |\zeta''(x)| \right] = p''_i \cdot \hat{m} \cdot |C \cap X|.
\]

By Bernstein inequality, we have:

\[
\Pr \left[ \sum_{x \in X : h''(x) = 1, g''(x) = C} |\zeta''(x)| - p''_i \cdot \hat{m} \cdot |C \cap X| \geq 0.1 \cdot p''_i \cdot \hat{m} \cdot R_i \right] \\
\leq \exp \left( - \frac{0.5 \cdot 0.1^2 \cdot p''_i^2 \cdot \hat{m}^2 \cdot R_i^2}{p''_i^2 \cdot \hat{m} \cdot |C \cap X| + \frac{1}{3} \cdot 0.1 \cdot p''_i \cdot \hat{m} \cdot R_i} \right) \\
\leq \delta / (nL + 1),
\]

where the last inequality follows from that \( |C \cap X| \leq 1.1R_i \) and \( p''_i \cdot \hat{m} \geq 10^4 \ln(nL/\delta)/R_i \). By taking union bound over all \( i \in \{0, 1, \cdots, L\} \) and all non-empty crucial cells \( C \in G_i \), with probability at least \( 1 - \delta \),

\[
\sum_{x \in X : h''(x) = 1, g''(x) = C} |\zeta''(x)| \leq 1.2 \cdot p''_i \cdot \hat{m} \cdot R_i \leq T''_i.
\]

Thus, according to the condition \( 2b \) of Definition 2.1, \( \forall x \in X^i \), if \( \zeta''(x) = 0 \), then \( (x, \zeta''(x)) \in Z''(X) \). According to the construction of \( \hat{Y}^i_j \), we have \( \hat{Y}^i_j = Y^i_j \).

**Lemma 5.7.** (GOOD SIMULATION IF NOT FAIL) If line \( 27 \) of Algorithm 3 are simulated such that \( \forall i \in \{0, 1, \cdots, L\}, C \in G_i \), \( \mathcal{V}(C) \in |C \cap X| \pm 0.1R_i \) or \( \mathcal{V}(C) \in (1 \pm 0.01) \cdot |C \cap X| \) and \( \forall i \in \{0, 1, \cdots, L\} \), \( \lambda(X^i) \in |X^i| \pm 0.1e\gamma R_i \) or \( \lambda(X^i) \in (1 \pm 0.01e) \cdot |X^i| \), then conditioned on that the simulating process via \( Z''(X) \) does not output FAIL, it simulates line \( 37 \) of Algorithm 3 with probability at least \( 1 - \delta \).

**Proof.** To show the lemma statement, we only need to show that each sample drawn via \( \hat{Y}^i_j \) is a uniform sample from \( X^i \). According to Lemma 5.6, with probability at least \( 1 - \delta \), \( \forall i \in \{0, 1, \cdots, L\}, j \in [\hat{m}] \), \( \hat{Y}^i_j = Y^i_j \). Thus, a uniform sample from a non-empty \( \hat{Y}^i_j \) is a uniform sample from \( X^i \). If the simulating process does not output FAIL, then the simulating process via \( Z''(X) \) is exactly the same as line \( 812 \) of Algorithm 3.

**Lemma 5.8.** (SIZE OF \( Z''(X) \)) With probability at least \( 1 - 2\delta \), the space budget of \( Z''(X) \) is at most

\[
O \left( 2^{2p+2} \cdot \epsilon^{-3} \left( kd + d^{O(p)} \cdot \frac{\OPT}{o} \right) \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot L^2 / \delta^2 \right).
\]
Proof. The proof is similar to the proof of Lemma 5.2. We will still use the concept of center cells (see Lemma 5.3).

The space budget of \(Z''(X)\)

\[
\sum_{i=1}^{L} \min_{C \in G_i} \left( T''_i, \sum_{x \in X: h''_i(x) = 1, g''_i(x) = C} |\zeta''_i(x)| \right) \leq \sum_{i=0}^{L} \left( \sum_{C \in G_i:C \text{ is a center cell}} T''_i + \sum_{C \in G_i:C \text{ is not a center cell}} \sum_{x \in C \cup X} |\zeta'_i(x)| \right)
\]

(5.5)

(5.6)

According to Lemma 5.3, part (5.5) is at most \(6kL/\delta \cdot 10^7 \cdot 2^{2p+2} \cdot \epsilon^{-3} \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot Ld/\delta\) with probability at least \(1 - \delta\). In the remaining of the proof let us upper bound part (5.6).

Recall that \(B^*\) is the optimal centers for \(X\), i.e., \(B^*\) satisfies \(\text{cost}(X, B^*) = \text{OPT}\). Consider \(i \in \{0, 1, \cdots, L\}\). If \(x \in X\) is not in any center cell in \(G_i\), then \(\min_{b \in B^*} ||x - b||_2 \geq \Delta_i/(2d)\). Thus, \(|\{x \in X \mid x \text{ is not in any center cell in } G_i\}| \leq (\Delta_i/(2d))^p = 100 \cdot (2d^{1.5})^p \cdot R_i \cdot \text{OPT}/\delta\), where the last inequality follows from that \(R_i = 0.010/(\sqrt{d}\Delta_i)^p\). Since \(p''_i \cdot \hat{m} \leq 10^6 \cdot 2^{2p+2} \cdot \epsilon^{-3} \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot Ld/\hat{m}\),

\[
E \left[ \sum_{x \in X : x \text{ is not in any center cell in } G_i} |\zeta'_i(x)| \right] \leq 10^8 \cdot 2^{2p+2} \cdot \epsilon^{-3} \cdot (2d^{1.5})^p \cdot (p \log n \log(kLd) + \log(\delta^{-1})) \cdot \frac{Ld}{\delta}. \frac{\text{OPT}}{o}.
\]

By Markov’s inequality, with probability at least \(1 - \delta\), the part (5.6) can be upper bounded by

\[
10^8 \cdot 2^{2p+2} \cdot \epsilon^{-3} \cdot (2d^{1.5})^p \cdot (p \log n \log(kLd) + \log(\delta^{-1})) \cdot \frac{L^2}{\delta^2} \cdot \frac{\text{OPT}}{o}.
\]

By combining the upper bound of part (5.5) with the upper bound of part (5.6), we complete the proof.

Lemma 5.9. (Number of non-empty \(Y^i_j\) is large) Consider \(o \geq \text{OPT}/16\). Suppose for every \(i \in \{0, 1, \cdots, L\}\) and every cell \(C \in G_i\), the estimated value \(\lambda(C)\) in line 7 of Algorithm 3 satisfies \(\lambda(C) \in [C \cap X] \pm 0.1R_i\) or \(\lambda(C) \in (1 \pm 0.01) \cdot |C \cap X|\), and for every \(i \in \{0, 1, \cdots, L\}\), the estimated value \(\lambda(X^i)\) in line 7 of Algorithm 3 satisfies \(\lambda(X^i) \in |Y^i| \pm 0.1 \gamma R_i\) or \(\lambda(X^i) \in (1 \pm 0.01\epsilon) \cdot |X^i|\). With probability at least \(1 - 2\delta\),

\[
\forall i \in I, \sum_{j=1}^{\hat{m}} 1(Y^i_j \neq \emptyset) \geq 10^5 \cdot 2^{2p+1} \cdot \epsilon^{-2} \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot \frac{Ld}{\delta} \cdot \frac{\lambda(X^i)}{R_i}
\]

Proof. Consider \(i \in I\). According to Lemma 5.3, with probability at least \(1 - \delta\), the number of center cells is at
most $6kL$. According to the construction of $X^i$ by Algorithm 3, none of $x \in X^i$ is in a heavy cell. Therefore,

$$|X^i| \leq \sum_{C \in G_i : C \text{ is a center cell and } C \text{ is crucial}} |C \cap X| + \sum_{C \in G_i : C \text{ is not a center cell}} |C \cap X| \leq 6kL \cdot 1.1R_i + \frac{\text{OPT}}{(\Delta_i/(2d))^p} \leq 7kL \cdot R_i + 100 \cdot (2d^{1.5})^p \cdot \frac{\text{OPT}}{o} \cdot R_i$$

(5.7)

where the second step follows from that a crucial cell contains at most $1.1R_i$ points, the third step follows from that the number of center cells is at most $6kL$ and the distance from any point outside a center cell to the optimal centers is at least $(\Delta_i/(2d))^p$, the forth step follows from that $R_i = 0.01 \cdot o/(\sqrt{d} \cdot \Delta_i)^p$ and the last step follows from $\text{OPT} \leq 16 \cdot o$.

By union bound over all $x \in X^i$, we have that

$$\forall i, j \in [\hat{m}], \Pr[Y^i_j \neq \emptyset] \leq |X^i| \cdot p''_i < 1,$$

where the last inequality follows from Equation (5.7) and $p''_i < 1/(1600(kL + (2d^{1.5})^p)R_i)$. Since, $|X^i| \cdot p''_i < 1$, we have $|x \cup [x^{1/p}_p] | \leq |x^{1/p}_p|$. Let $b = 10 \cdot |x^{1/p}_p|$. Let $r = |\hat{m}/b| \geq \frac{1}{2} \cdot \hat{m} \cdot |X^i| \cdot p''_i - 1$. Since $X^i \neq \emptyset$, it means that there is at least one non-empty crucial cell in $G_i$. By the construction of $X^i$, we know that a non-empty crucial cell in $G_i$ implies that $R_i \geq \frac{1}{2}$. Since $p''_i = \min(1/(2000 \cdot (kL + (2d^{1.5})^p)) \cdot R_i), 1$, we have $p''_i = 1/(2000 \cdot (kL + (2d^{1.5})^p)) \cdot R_i)$. Thus,

$$r \geq \frac{1}{2} \cdot \hat{m} \cdot \frac{1}{2000(kL + (2d^{1.5})^p)} \cdot |X^i| - 1 \geq \frac{1}{4} \cdot \hat{m} \cdot \frac{1}{2000(kL + (2d^{1.5})^p)} \cdot |X^i| \cdot R_i$$

(5.9)

where the last inequality follows from that $|X^i| \geq \frac{2}{\gamma} \cdot R_i$ since $i \in I$ and $\hat{m} \geq 10^4(kL + (2d^{1.5})^p)/\gamma$. For $s \in [r]$, we can define a random variable $Q_s$:

$$Q_s = \sum_{j=(s-1) \cdot b+1}^{s \cdot b} \sum_{x \in X^i} 1[j \in \zeta''_i(x)].$$

We have that $\mathbb{E}[Q_s] = b \cdot |X^i| \cdot p''_i$. By our choice of $b$, we know that $\mathbb{E}[Q_s] \in [10, 20]$. Since $Q_s$ is a sum of independent random variables from $\{0, 1\}$, the variance $\text{Var}[Q_s] \leq \mathbb{E}[Q_s] \leq 20$. By Chebyshev’s inequality, we have $\Pr[Q_s < 1] \leq \frac{1}{4}$.

Notice that $Q_s \geq 1$ means that $\exists j \in \{(s-1) \cdot b+1, (s-1) \cdot b+2, \cdots, s \cdot b\}$ such that $Y^i_j \neq \emptyset$. Define random variable $P_s = 1(Q_s \geq 1)$. Then we have $\sum_{s \in [r]} P_s \geq \sum_{j=1}^{\hat{m}} 1(Y^i_j \neq \emptyset)$. Notice that $\mathbb{E}[\sum_{s \in [r]} P_s] \geq 0.75 r$. By Chernoff bound, we have

$$\Pr \left[ \sum_{s \in [r]} P_s \leq 0.5r \right] \leq 2^{-r/20} \leq \delta / (L + 1),$$

where the last inequality follows from that $r \geq \frac{1}{4} \cdot \hat{m} \cdot \frac{1}{2000(kL + (2d^{1.5})^p)} \cdot |X^i| \geq 20 \log(2L/\delta) \text{ since } |X^i| \geq \frac{\gamma}{2}$ and $\hat{m} \geq 10^7(kL + (2d^{1.5})^p) \cdot \log(L/\delta)/\gamma$.

Since $\hat{m} \geq 10^9 \cdot 2^{2p+2} \cdot \epsilon^{-2} \cdot (kL + (2d^{1.5})^p) \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot \frac{1}{\delta}$ and $\lambda(X^i) \leq 2|X^i|$, according to Equation (5.9), we have

$$r \geq 10^5 \cdot 2^{2p+2} \cdot \epsilon^{-2} \cdot (p \log n \log(kLd) + \log(1/\delta)) \cdot \frac{L}{\delta} \cdot \lambda(X^i) / R_i.$$
Thus, by taking union bound over all $i \in I$, with probability at least $1 - \delta$, 
\[
\forall i \in I, \sum_{j=1}^{m} 1(Y_j^i \neq \emptyset) \geq 10^5 \cdot 2^{2p+1} \cdot e^{-2}\cdot (p \log n \log (kLd) + \log(1/\delta)) \cdot \frac{L}{\delta} \cdot \frac{\lambda(X^i)}{R_i}.
\]

\[
\square
\]

**Lemma 5.10. (Number of Samples Needed)** Suppose for every $i \in \{0, 1, \ldots, L\}$ and every cell $C \in G_i$, the estimated value $\lambda(C)$ in line 7 of Algorithm 3 satisfies $\lambda(C) \in |C \cap X| \pm 0.1R_i$ or $\lambda(C) \in (1 \pm 0.01) \cdot |C \cap X|$, and for every $i \in \{0, 1, \ldots, L\}$, the estimated value $\lambda(X^i)$ in line 3 of Algorithm 3 satisfies $\lambda(X^i) \in |X^i| \pm 0.1\gamma R_i$ or $\lambda(X^i) \in (1 \pm 0.01) \cdot |X^i|$. Consider Algorithm 3. With probability at least $1 - 2\delta$, all of the following things happen:

1. $t' = \sum_{i \in I} \lambda(X^i) \cdot \min(2^{2p+1}/R_i, 1) \leq 200 \cdot 2^{2p+2} \cdot (k + (2d^{1.5})^p \cdot \frac{\text{OPT}}{o}) \cdot L$.

2. The times that $i \in I$ is sampled by line 9 of Algorithm 3 is at most

\[
4000 \cdot 2^{2p+1} \cdot \epsilon^{-2}(\log n \log(2t') + \log(1/\delta)) \cdot \frac{\lambda(X^i)}{R_i} \cdot \frac{L}{\delta}.
\]

**Proof.** Let us first bound $t'$. According to Lemma 5.3, with probability at least $1 - \delta$, the number of center cells is at most $6kL$. According to the construction of $X^i$ by Algorithm 3, none of the point $x \in X^i$ for $i \in \{0, 1, \ldots, L\}$ is in a heavy cell. Therefore,

\[
\sum_{i=0}^{L} |X^i|/R_i \leq \sum_{i=0}^{L} \sum_{C \in G_i; C \text{ is a center cell and } C \text{ is crucial}} |C \cap X|/R_i + \sum_{i=0}^{L} \sum_{C \in G_i; C \text{ is not a center cell}} |C \cap X|/R_i \\
\leq \sum_{i=0}^{L} \sum_{C \in G_i; C \text{ is a center cell}} 1 + \sum_{i=0}^{L} \sum_{C \in G_i; C \text{ is not a center cell}} |C \cap X|/R_i \\
\leq 6kL \cdot 1 + \sum_{i=0}^{L} \frac{\text{OPT}}{(\Delta_i/(2d))^p \cdot R_i} \\
\leq 7kL \cdot 100 \cdot (2d^{1.5})^p \cdot \frac{\text{OPT}}{o} \cdot (L + 1),
\]

where the second step follows from that a crucial cell contains at most $1.1R_i$ points, the third step follows from that the number of center cells is at most $6kL$ and the distance from any point outside a center cell to the optimal centers is at least $(\Delta_i/(2d))^p$, and the forth step follows from that $R_i = 0.01 \cdot o/(\sqrt{d} \cdot \Delta_i)^p$.

Then we have:

\[
t' = \sum_{i \in I} \lambda(X^i) \cdot \min(2^{2p+1}/R_i, 1) \\
\leq \sum_{i \in I} 2 \cdot |X^i| \cdot \min(2^{2p+1}/R_i, 1) \\
\leq \sum_{i=0}^{L} 2 \cdot |X^i| \cdot \min(2^{2p+1}/R_i, 1) \\
\leq 200 \cdot 2^{2p+2} \cdot (k + (2d^{1.5})^p \cdot \frac{\text{OPT}}{o}) \cdot L,
\]

where the second step follows from that $\lambda(X^i)$ is a good estimation of $|X^i|$ and $\forall i \in I, \lambda(X^i) \geq \gamma R_i$, the third step follows from that $I \subseteq \{0, 1, \ldots, L\}$ and the last step follows from Equation (5.10).
Next, let us consider the times that \( i \in I \) is sampled by line 9 of Algorithm 3. The expected number of times that \( i \in I \) is sampled is:

\[
m \cdot \frac{\lambda(X^i) \cdot \min(2^{2p+1}/R_i, 1)}{\tilde{p}} \leq 2000 \cdot 2^{2p+1} \cdot \epsilon^{-2}(\log n \log(2t') + \log(1/\delta)) \cdot \frac{\lambda(X^i)}{R_i}.
\]

By Markov’s inequality, with probability at least 1 − \( \delta/(L + 1) \), \( i \in I \) is sampled at most

\[
2000 \cdot 2^{2p+1} \cdot \epsilon^{-2}(\log n \log(2t') + \log(1/\delta)) \cdot \frac{\lambda(X^i)}{R_i} \cdot \frac{L + 1}{\delta}
\]
times. Thus, by taking union bound over all \( i \in I \), with probability at least 1 − \( \delta \), \( \forall i \in I \), \( i \) is sampled

\[
4000 \cdot 2^{2p+1} \cdot \epsilon^{-2}(\log n \log(2t') + \log(1/\delta)) \cdot \frac{\lambda(X^i)}{R_i} \cdot \frac{L}{\delta}
\]
times. \( \square \)

**Lemma 5.11. (Simulating process does not output FAIL with a good probability)** Consider \( o \geq \text{OPT} \). Suppose for every \( i \in \{0, 1, \ldots, L\} \) and every cell \( C \in G_i \), the estimated value \( \lambda(C) \) in line 7 of Algorithm 3 satisfies \( \lambda(C) \in |C \cap X| + 0.1R_i \) or \( \lambda(C) \in (1 \pm 0.01) \cdot |C \cap X| \), and for every \( i \in \{0, 1, \ldots, L\} \), the estimated value \( \lambda(X^i) \) in line 5 of Algorithm 3 satisfies \( \lambda(X^i) \in |X^i| \pm 0.1\gamma R_i \) or \( \lambda(X^i) \in (1 \pm 0.01\epsilon) \cdot |X^i| \). The probability that the simulating process via \( Z^o(X) \) does not output \( \text{FAIL} \) is at least 1 − 5\( \delta \).

**Proof.** The simulating process outputs \( \text{FAIL} \) if and only if \( \exists i \in I \) such that the times that level \( i \) is sampled is more than the number of non-empty \( \tilde{Y}_j \). According to Lemma 5.6 with probability at least 1 − \( \delta \), \( \forall i \in I, j \in [\tilde{m}], \tilde{Y}_j \). According to Lemma 5.9 and Lemma 5.10 with probability at least 1 − 4\( \delta \), \( \forall i \in I \), the times that level \( i \) is sampled is at most the number of non-empty sets \( Y_j \). By union bound, with probability at least 1 − 5\( \delta \), the simulating process via \( Z^o(X) \) does not output \( \text{FAIL} \). \( \square \)

**Theorem 5.2. (Bucketing-based sketch for \( \ell_p \) k-clustering)** Let \( k \geq 1, p \geq 1 \). Consider the \( \ell_p \) k-clustering problem over a point set in \( |\Delta|^d \) with at most \( n \) points. Let \( \epsilon, \delta \in (0, 0.5) \). For any \( o \geq 1, \alpha \geq 1 \), there always exists a weak \( o \)-restricted \( (1 + \epsilon)\alpha \)-approximate bucketing-based sketch \( Z^o(\cdot) \) such that for any data set \( X \subseteq |\Delta|^d \),

1. the recover function over \( Z^o(X) \) either outputs \( \text{FAIL} \) or outputs a \((1 + \epsilon)\alpha\)-approximation to the optimal \( \ell_p \) k-clustering cost with probability at least 1 − \( \delta \) conditioned on that the optimal cost is at least \( o \);

2. if the optimal cost is at most \( 2 \cdot o \), with probability at least 1 − \( \delta \), the space budget of \( Z^o(X) \) is upper bounded by \( \tilde{O}_p(\epsilon^{-3}\delta^{-2} (kd^{O(p)}) \log^3(n\Delta)) \) and the recover function over \( Z^o(X) \) does not output \( \text{FAIL} \).

Furthermore, the time needed to compute a filter function, a bucketing function, and a processing function on any data item \( x \in |\Delta|^d \) is always upper bounded by \( \tilde{O}_p(\epsilon^{-3}\delta^{-1} \cdot (k + d^{O(p)}) \cdot \log^2(n\Delta)) \). The time to compute recover function over \( Z^o(X) \) is at most \( \mathcal{T}(m') \) where \( m' \) is the space budget of \( Z^o(X) \) and \( \mathcal{T}(n') \) denotes the running time needed to compute an \( \alpha \)-approximation for the \( \ell_p \) k-clustering for a weighted point set with \( n' \) points in \( \mathbb{R}^d \).

**Proof.** We simply set \( Z^o(X) \) as the composition of \( Z(X), Z'(X) \) and \( Z''(X) \) described in Section 5.2.1 and Section 5.2.2 i.e.,

\[
Z^o(X) = (Z_0(X), Z_1(X), \ldots, Z_L(X), Z'_0(X), Z'_1(X), \ldots, Z'_L(X), Z''_0(X), Z''_1(X), \ldots, Z''_L(X)).
\]

According to Lemma 5.1, Lemma 5.3 and Lemma 5.7 if the simulating process via \( Z''(X) \) does not output \( \text{FAIL} \), then with probability at least 1 − 4\( \delta \), Algorithm 2 and Algorithm 3 can be simulated via \( Z(X), Z'(X), Z''(X) \) and every \( \lambda(C) \) satisfies \( \lambda(C) \in |C \cap X| + 0.1R_i \) or \( \lambda(C) \in (1 \pm 0.01) \cdot |C \cap X| \) and every \( \lambda(X^i) \) satisfies \( \lambda(X^i) \in |X^i| \pm 0.1\gamma R_i \) or \( \lambda(X^i) \in (1 \pm 0.01\epsilon) \cdot |X^i| \). If \( o \leq \text{OPT} \), according to Theorem 5.1, the output \((S, w)\) is
Theorem 5.3. For any \( \epsilon, \delta' \in (0, 0.5), \alpha \geq 1, k \geq 1, p \geq 1 \), there is a \((1 + \epsilon)\alpha\)-approximate algorithm for the \( \ell_p \) \( k \)-clustering problem whose space budget is at most \( \tilde{O}_p(2^{O(p)}\epsilon^{-3}(kd + d^{O(p)}) \cdot \log nd) \). The update time is at most \( \tilde{O}_p(\epsilon^{-3}d^{-2}(kd + d^{O(p)}) \cdot \log^3 (n\Delta)) \). The running time to get the \((1 + \epsilon)\alpha\)-approximate solution at the end of the stream is at most \( \tilde{O}_p(\epsilon^{-3}d^{-2}(kd + d^{O(p)}) \cdot \log^3 (W\Delta)) \). The success probability is at least \( 1 - \delta' \).

Proof. Suppose the point set of interest at the end of the stream is \( X \subseteq [\Delta]^d \). We first need to handle the case if \( \text{OPT} = 0 \). We have \( \text{OPT} = 0 \) if and only if \( X \) contains at most \( k \) distinct points. We can use the following sliding window procedure to check whether \( X \) has at most \( k \) points:

1. Initialize a list of points \( L = \emptyset \).
2. For the latest \( x \) in the stream:
   (a) Add \( x \) into \( L \).
   (b) If there are 2 points (including \( x \) itself) in \( L \) that are equal to \( x \), remove the point which is equal to \( x \) with earlier timestamp from \( L \).
   (c) Otherwise, if \( L \) has \( k + 2 \) points, remove the point with the earliest timestamp from \( L \).
3. At timestamp \( N \), let \( L' = \{ x \in L \mid \text{the timestamp is at least } N - W + 1, \text{i.e., } x \in X \} \).
4. If \( L' \) has at most \( k \) distinct points, return 0.

In the remaining of the proof, we only need to discuss the case when \( \text{OPT} > 0 \). Since \( X \subseteq [\Delta]^d \) and \( |X| \leq W \), we have \( \text{OPT} \in [1, W \cdot (\sqrt{\Delta})^p] \). Let \( \delta = \tilde{O}_p(\delta'/\log(Wd\Delta)) \). According to Theorem 5.3, for any \( \delta' \in (0, 0.5) \), \( \alpha \in [0, 1], \epsilon, \delta \in (0, 0.5) \), there always exists a weak \( k \)-restricted \((1 + \epsilon)\alpha\)-approximate bucketing-based sketch \( Z''(\cdot) \) for \( \ell_p \) \( k \)-clustering such that the recover function over \( Z''(X) \) either outputs \( \text{FAIL} \) or outputs a \((1 + \epsilon)\alpha\)-approximation to \( \text{OPT} \) with probability at least \( 1 - \delta \) conditioned on that \( \text{OPT} \geq \alpha \). Furthermore, if \( \text{OPT} \leq \epsilon \alpha \), with probability at least \( 1 - \delta \), the space budget of \( Z''(X) \) is at most \( \tilde{O}_p(\epsilon^{-3}\delta^{-2}(kd + d^{O(p)}) \cdot \log^3 (W\Delta)) \) and the recover function over \( Z''(X) \) does not output \( \text{FAIL} \). According to Theorem 2.1 after plugging the bucketing-based sketch into Algorithm 1, we obtain the sliding window algorithm which outputs a \((1 + \epsilon)\alpha\)-approximation to \( \text{OPT} \). The space needed is at most \( \tilde{O}_p(\epsilon^{-3}\delta^{-2}(kd + d^{O(p)}) \cdot \log^3 (W\Delta)) = \tilde{O}_p(\epsilon^{-3}\delta^{-2}(kd + d^{O(p)}) \cdot \log^3 (W\Delta)) = \tilde{O}_p(\epsilon^{-3}\delta^{-2}(kd + d^{O(p)}) \cdot \log^3 (W\Delta)) \). The success probability is at most \( 1 - \delta \cdot \tilde{O}_p(\log(Wd\Delta)) = 1 - \delta' \). According
to Theorem 5.2, the time needed to compute a filter function, a bucketing function and a processing function on any data item is always at most $O_p(e^{-3\delta -1} \cdot (k + dO(p)) \cdot \log^2(W \Delta))$. According to Theorem 2.1, the update time is at most $O_p(log(\log(Wd \Delta)) \cdot \log(Wd \Delta) \cdot e^{-3\delta -1} \cdot (k + dO(p)) \cdot \log^2(W \Delta) + e^{-3\delta -2} (kd + d^O(p))^6 \log(W \Delta) \cdot \log(Wd \Delta)) = O_p(e^{-3\delta -2} (kd + d^O(p)) \log^6(W \Delta))$. At the end of the stream, to get the final approximation, we need to evaluate the recover function $O_p(\log(Wd \Delta))$ times according to Algorithm 1. Since the size of each sketch is at most $O_p(e^{-3\delta -2} (kd + d^O(p)) \log^6(W \Delta))$, according to Theorem 5.2, the time needed to compute the recover function for each sketch is at most $T(O_p(e^{-3\delta -2} (kd + d^O(p)) \log^6(W \Delta)))$. Thus, the overall running time to get the approximate solution at the end of the stream is at most $T(O_p(e^{-3\delta -2} (kd + d^O(p)) \log^6(W \Delta))) \cdot O_p(\log(Wd \Delta))$.

References


A Discussion of the Smoothness of \( k \)-Cover, Diversity Maximization and Clustering

In this section, we briefly review the smooth histogram framework. Then we will show examples that \( k \)-cover, diversity maximization and clustering are not smooth enough to obtain \((1 \pm \epsilon)\)-approximation via smooth histogram.

A.1 Smooth Function and Smooth Histogram

Exponential histogram is an algorithmic framework for sliding window algorithms and is proposed by [DGM02]. It shows that weakly additive functions can be approximated by efficient sliding window algorithms via exponential histogram. Later, [BO07] proposes the smooth histogram framework which can be used to develop efficient sliding window algorithms for approximating smooth functions. It was shown that the class of weakly additive functions is a sub-class of the class of smooth functions. Thus, smooth histogram is a more general framework than exponential histogram.

Suppose \( A \) and \( B \) are two streams of data items. Let \( B \subseteq_r A \) indicate that \( B \) is a suffix of \( A \). The definition of the smooth function is as follows.

**Definition A.1.** \(((\alpha, \beta)\)-Smooth function [BO07]) For \( \alpha \in (0, 1), \beta \in (0, \alpha] \), function \( f(\cdot) \) is \((\alpha, \beta)\)-smooth, if it holds the following properties:

1. \( \exists \) constant \( c > 0 \), \( \forall A, 0 \leq f(A) \leq |A|^c \).
2. \( \forall B \subseteq_r A, f(A) \geq f(B) \).
3. If \( B \subseteq_r A \) and \((1 - \beta)f(A) \leq f(B) \) then \((1 - \alpha)f(A \cup C) \leq f(B \cup C) \) for any subsequent \( C \).
Roughly speaking, for a smooth function, if the function over a suffix of a stream of the data is a good
approximation of the function over the entire stream, then after the arrival of any new data items, the function
over such suffix is always a good approximation over the entire stream of data. Therefore, the idea of smooth
histogram is to maintain a set of start points over the stream. The sliding window algorithm maintains a sketch
of data items from the start point to the current end of the stream for each start point. A start point is created
when a new data item is arrived and a start point can be dropped if the stream from the next start point can
approximate the stream from the previous start point very well, i.e., the value of the function over the stream
starting from the next start point is at least \((1 − β)\) times the value of the function over the stream starting
from the previous start point. If the function is an \((α, β)\)-smooth function, the function values produced by two
adjacent start points are always within \((1 − α)\). Since a start point can be dropped if the function value produced
by the next start point is within \((1 − β)\) of the function value produced by the previous start point, the smooth
histogram only needs to maintain at most \(O\left(\frac{1}{β} \cdot \log W\right)\) start points. The formal statement of the guarantee
of the sliding window algorithm via smooth histogram is shown as the following.

**Theorem A.1. (BHO07)** Let \(f(\cdot)\) be an \((α, β)\)-smooth function. If there is a streaming algorithm which outputs
a \((1 − ε')\)-approximation (or \(1/(1 − ε')\)-approximation) of \(f(\cdot)\) using \(S(ε')\) space, then there exists a sliding
window algorithm which outputs a \((1 − α − ε')\)-approximation (or \(1/(1 − α − ε')\)-approximation) of \(f(\cdot)\) using
\(O(\log(W)/β \cdot S(ε'))\) space where \(W \geq 1\) is the window size.

Thus, if \(f(\cdot)\) is \((α, β)\)-smooth with smaller \(α\), the sliding window algorithm via smooth histogram can output a
better approximation. In the remaining of the section, we will show that there is a constant lower bound of \(α\) for
\(k\)-cover and diversity maximization. \(ℓ_p\) \(k\)-Clustering is not \((α, β)\)-smooth for any \(α, β \geq 0\).

A.2 \((Ω(1), β)\)-Smoothness of \(k\)-Cover The following lemma shows that we cannot use the smooth histogram
to obtain a sliding window algorithm for \(k\)-cover in the edge-arrival model with approximation ratio better than
\(1/2\).

**Lemma A.1.** \(k\)-Cover in edge-arrival model cannot be \((α, β)\)-smooth for any \(α < 1/2\).

**Proof.** Consider \(k = 1\). There are two sets \(S_1, S_2\) and \(2m\) elements \(e_1,e_2,\cdots,e_{2m}\). The stream \(A\
contains edges \((S_1,e_1),(S_1,e_2),\cdots,(S_1,e_m),(S_2,e_{m+1}),(S_2,e_{m+2}),\cdots,(S_2,e_{2m})\). The suffix stream \(B\
contains edges \((S_2,e_{m+1}),(S_2,e_{m+2}),\cdots,(S_2,e_{2m})\). The optimal \(1\)-cover of the both \(A\) and \(B\) are the same, i.e.,
\(OPT_1(A) = OPT_1(B) = m\). Thus, for any \(β \geq 0\), \((1 − β)OPT_1(A) \leq OPT_1(B)\). Let \(C\) be the edges
\((S_1,e_{m+1}),(S_1,e_{m+2}),\cdots,(S_1,e_{2m})\). Then we have \((1 − α)OPT_1(A ∪ C) = (1 − α) \cdot 2m > OPT_1(B ∪ C) = m\).
Thus, it cannot be \((α, β)\)-smooth for any \(α < 1/2\). To generalize to \(k > 1\), we only need to make \(k\) copies of
\(S_1, S_2, e_1, e_2, \cdots, e_{2m}\). □

A.3 \((Ω(1), β)\)-Smoothness of Diversity Maximization Recall that the diversity functions that we are
considering is listed in Table 7. The definition of \(OPT\) is given in Section 4. Let us consider the case \(k = 2\). In this
case, \(OPT(P)\) is always the same for every diversity function, i.e., \(OPT(P) = \max_{Q \subseteq P: |Q| = 2} \min_{u \neq v \in Q} \| u − v \|_2\).

The following lemma shows that we cannot use smooth histogram to obtain a sliding window algorithm for
diversity maximization for \(k = 2\) and any diversity function listed in Table 1 with approximation ratio better
than \(1/\sqrt{2}\).

**Lemma A.2.** Diversity maximization with \(k = 2\) and any diversity function listed in Table 7 cannot be \((α, β)\)-
smooth for any \(α < 1 − 1/\sqrt{2}\).

**Proof.** Let \(A = \{(0,1,1,0),(1,1,0,0),(1,0,1,0)\}\). Let \(B\) be the suffix of \(A\), i.e., \(B = \{(1,1,0,0),(1,0,1,0)\}\). Let
\(C = \{(1,0,0,1)\}\). It is easy to verify that \(OPT(A) = OPT(B) = \sqrt{2}\) and \(OPT(A ∪ C) = 2\), \(OPT(B ∪ C) = \sqrt{2}\).
Thus we have \((1 − α)OPT(A) \leq OPT(B)\) but \((1 − α)OPT(A ∪ C) > OPT(B ∪ C)\). □

In the following, we extend the above lemma to the case when \(k > 2\).

**Lemma A.3.** Consider \(k > 2\).

1. Remote-edge cannot be \((α, β)\)-smooth for any \(α < 1 − 1/\sqrt{2}\).
2. Remote-clique cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(k(k-1))\).
3. Remote-tree cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2(k-1))\).
4. Remote-cycle cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2k)\).
5. Remote \(t\)-trees cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2(k-t))\).
6. Remote \(t\)-cycle cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2k)\).
7. Remote-star cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2(k-1))\).
8. Remote-bipartition cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2|k/2| |k/2|)\).
9. Remote-pseudoforest cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/(2k)\).
10. Remote-matching cannot be \((\alpha, \beta)\)-smooth for any \(\alpha < (2 - \sqrt{2})/k\).

Proof. Let \(A\) contain \(k+1\) \((2k)\)-dimensional vectors:

\[
A = \{v_1 = (1,0,0,\ldots,0,1,0,0,\ldots,0), \ v_2 = (1,1,0,\ldots,0,0,0,\ldots,0), \ v_3 = (0,1,1,0,\ldots,0,0,\ldots,0), \ldots, \ v_{k+1} = (0,0,0,\ldots,1,1,0,0,\ldots,0)\},
\]

i.e., \(v_1 = e_1 + e_{k+1}, v_2 = e_2 + e_1, v_3 = e_3 + e_2, \ldots, v_{k+1} = e_{k+1} + e_k\). Let \(B\) contain \(k\) \((2k)\)-dimensional vectors:

\[
B = \{v_2, v_3, \ldots, v_{k+1}\}.
\]

Let \(C\) contain \(k-1\) \((2k)\)-dimensional vectors:

\[
C = \{u_1 = (0,1,0,\ldots,0,0,1,0,\ldots,0), \ u_2 = (0,0,1,\ldots,0,0,0,1,\ldots,0), \ldots, \ u_{k-1} = (0,0,0,\ldots,1,0,0,0,\ldots,1)\},
\]

i.e., \(u_1 = e_2 + e_{k+2}, u_2 = e_3 + e_{k+3}, \ldots, u_{k-1} = e_k + e_{2k}\).

By symmetry, if we choose arbitrary \(k\) vectors \(Q \subseteq A\), then \(\min_{p \notin Q} \|p - q\|_2 = \sqrt{2}\). Thus, if we choose \(Q = B\), then \(\min_{p \notin Q} \|p - q\|_2 = \sqrt{2}\). Consider \(A \cup C\). If we choose \(Q = \{v_1, u_1, u_2, \ldots, u_{k-1}\} \subseteq A \cup C\), then \(\min_{p \notin Q} \|p - q\|_2 = 2\).

Next we consider \(B \cup C\).

Claim A.1. For any \(i \in [k-1]\), let \(Q_i\) be arbitrary \(i+1\) vectors from \(\{v_2, v_3, \ldots, v_{i+2}, u_1, u_2, \ldots, u_i\}\). Then \(\min_{p \notin Q_i} \|p - q\|_2 = \sqrt{2}\).

Proof. The proof is by induction. Consider the base case: \(Q_1\) contains arbitrary \(2\) vectors from \(\{v_2, v_3, u_1\}\). It is easy to verify that \(\min_{p \notin Q_1} \|p - q\|_2 = \sqrt{2}\). Suppose the claim is true for \(i - 1\). Consider \(Q_i\). There are two cases. In the first case, \(Q_i\) contains at least \(i\) vectors from \(\{v_2, v_3, \ldots, v_{i+1}, u_1, u_2, \ldots, u_{i-1}\}\). By induction hypothesis, since \(Q_i\) contains at least \(i\) vectors from \(\{v_2, v_3, \ldots, v_{i+1}, u_1, u_2, \ldots, u_{i-1}\}\), we have \(\min_{p \notin Q_i} \|p - q\|_2 = \sqrt{2}\). In the second case, \(Q_i\) contains at most \(i\) vectors from \(\{v_2, v_3, \ldots, v_{i+1}, u_1, u_2, \ldots, u_{i-1}\}\). In this case, both \(v_{i+2} = e_{i+2} + e_{i+1}\) and \(u_1 = e_{i+1} + e_{i+k}\) are in \(Q_i\). Thus, \(\min_{p \notin Q_i} \|p - q\|_2 = \|e_{i+2} - e_{i+k}\|_2 = \sqrt{2}\). □

By use \(i = k - 1\) in the above claim, we know that any \(Q \subseteq B \cup C\) with \(|Q| = k\), we have \(\min_{p \notin Q} \|p - q\|_2 = \sqrt{2}\).

Thus,

1. For remote-edge: \(\text{OPT}(A) = \text{OPT}(B) = \sqrt{2}, \text{OPT}(A \cup C) = 2, \text{OPT}(B \cup C) = \sqrt{2}\).
2. For remote-clique: \(\text{OPT}(A) = \text{OPT}(B) = (k-1) \cdot \sqrt{2} + (k-1)(k-2)/2 \cdot 2, \text{OPT}(A \cup C) = k(k-1), \text{OPT}(B \cup C) \leq k(k-1) - (2 - \sqrt{2})\).
3. For remote-tree: $\text{OPT}(A) = \text{OPT}(B) = (k-1)\sqrt{2}$, $\text{OPT}(A \cup C) = 2(k-1)$, $\text{OPT}(B \cup C) \leq 2(k-1)-(2-\sqrt{2})$.
4. For remote-cycle: $\text{OPT}(A) = \text{OPT}(B) = k \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2k$, $\text{OPT}(B \cup C) \leq 2k-(2-\sqrt{2})$.
5. For remote $t$-trees: $\text{OPT}(A) = \text{OPT}(B) = (k-t) \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2(k-t)$, $\text{OPT}(B \cup C) \leq 2(k-t)-(2-\sqrt{2})$.
6. For remote $t$-cycle: $\text{OPT}(A) = \text{OPT}(B) = k \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2k$, $\text{OPT}(B \cup C) \leq 2k-(2-\sqrt{2})$.
7. For remote-star: $\text{OPT}(A) = \text{OPT}(B) = (k-1)\sqrt{2}$, $\text{OPT}(A \cup C) = 2(k-1)$, $\text{OPT}(B \cup C) \leq 2(k-1)-(2-\sqrt{2})$.
8. For remote-bipartition: $\text{OPT}(A) = \text{OPT}(B) = [k/2] \cdot [k/2] \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2 \cdot [k/2] \cdot [k/2]$, $\text{OPT}(B \cup C) \leq 2 \cdot [k/2] \cdot [k/2]-(2-\sqrt{2})$.
9. For remote-pseudoforest: $\text{OPT}(A) = \text{OPT}(B) = k \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2k$, $\text{OPT}(B \cup C) \leq 2k-(2-\sqrt{2})$.
10. For remote-matching: $\text{OPT}(A) = \text{OPT}(B) = k/2 \cdot \sqrt{2}$, $\text{OPT}(A \cup C) = 2 \cdot k/2$, $\text{OPT}(B \cup C) \leq 2k/2-(2-\sqrt{2})$.

A.4 Non-smoothness of \( \ell_p \) k-Clustering
As shown by [BLLM16], \( \ell_p \) k-clustering is not \((\alpha, \beta)\)-smooth for any \( \alpha \) and \( \beta \). Thus, smooth histogram cannot be used to develop a sliding window algorithm for \( \ell_p \) k-clustering for any multiplicative approximation.

Lemma A.4. ([BLLM16]) Consider \( \ell_p \) k-clustering problem for \( p \geq 1 \). There exists sets of points \( A, B, C \subseteq [\Delta]^d \) such that \( \forall 0 < \beta \leq \alpha < 1 \), \((1-\beta)\text{OPT}(A \cup B) \leq \text{OPT}(B)\) but \((1-\alpha)\text{OPT}(A \cup B \cup C) > (1-\alpha)\text{OPT}(B \cup C)\).

A.5 \((p(1), \beta)\)-Smoothness of Toy 1-Median Problem
In the toy 1-median problem, given a (multi-)set \( X \subseteq \{0,1\} \), the goal is to compute \( \text{OPT}(X) = \min_{z \in \{0,1\}} \sum_{x \in X} |x-z| \). Let \( A = \{0,0\}, B = \{0\}, C = \{1,1,1\} \).
Then we have \((1-\beta)\text{OPT}(A) = 0 \leq \text{OPT}(B)\). But \( \frac{1}{2} \cdot \text{OPT}(A \cup C) = \frac{1}{2} \cdot 2 = 1 \geq \text{OPT}(B \cup C) = 1 \).

B Analysis of Offline Coreset Construction for k-Clustering
We give the analysis of the coreset construction shown in Section 5.1. The goal is to prove Theorem 5.1. The analysis is similar to [HSYZ18]. We include the analysis in this section for completeness.

Definition B.1. (Good estimation of number of points) For \( i \in \{0,1,\cdots,L\} \), \( C \in G_i \) if the estimated value \( \lambda(C) \) in line 7 of Algorithm 2 satisfies \( \lambda(C) \in |C \cap X| \pm 0.1 R_i \) or \( \lambda(C) \in (1 \pm 0.01) \cdot |C \cap X| \), then we say that \( \lambda(C) \) is good.

Fact B.1. If \( 0 \leq \text{OPT} \) and the estimation \( \lambda(C) \) is good for every cell \( C \), then for level \( i \in \{-1,0,\cdots,L\} \), every heavy cell \( C \in G_i \) contains at least \( 0.9 R_i \) points, i.e., \( |C \cap X| \geq 0.9 R_i \), and every crucial cell \( C \in G_i \) contains at most \( 1.1 R_i \) points, i.e., \( |C \cap X| \leq 1.1 R_i \).

Proof. For level \( i \in \{0,1,\cdots,L\} \), the claim follows directly from the construction of heavy and crucial cells in Algorithm 2 and Algorithm 3. For \( i = -1, G_{-1} \) does not contain any crucial cell. The unique heavy cell \( C \in G_{-1} \) contains every point in \( X \). Notice that each point can contribute to the clustering cost at most \((\sqrt{d} \cdot \Delta)^p\). Thus, \( |X| \geq \text{OPT} / (\sqrt{d} \cdot \Delta)^p \geq \alpha / (\sqrt{d} \cdot \Delta)^p \geq R_{-1} \). Thus \( |C \cap X| \geq 0.9 R_{-1} \).

Fact B.2. (No heavy cell in \( G_L \)) If the estimated value \( \lambda(C) \) is good for every cell \( C \), none of the cell in \( G_L \) is marked as heavy by Algorithm 3.

Proof. Since \( L = \left[ \log(n d \Delta) \right] + 10 \), we have \( R_L = 0.01 \cdot \frac{1}{(\sqrt{d} \cdot \Delta)^p} \geq 0.01 / (\sqrt{d} \cdot \Delta/2^2)^p \geq 2n \). Consider each cell \( C \in G_L \). According to Fact B.1, if \( C \in G_L \) is a heavy cell, then we have \( |C \cap X| \geq 0.9 R_{-1} \geq 1.8n \) which contradicts to \( |X| \leq n \). Thus, any cell \( C \in G_L \) cannot be heavy.

Fact B.3. (\( X \) is partitioned by crucial cells) If the estimated value \( \lambda(C) \) is good for every cell \( C \), \( X^0, X^1, \cdots, X^L \) in Algorithm 3 is a partition of \( X \), i.e., \( \forall x \in X \), there is an unique \( i \in \{0,1,\cdots,L\} \) such that \( x \in X^i \).
Lemma B.1. (Levels outside crucial.)

Fact B.4. If \( C \) is a heavy cell in \( G_i \) for some \( i \in \{-1, 0, \ldots, L-1\} \), then every child cell \( C' \) of \( C \) is either heavy or crucial.

Proof. Follows directly from the construction of heavy and crucial cells in Algorithm \( \mathcal{A}_2 \) and Algorithm \( \mathcal{A}_3 \).

Fact B.5. If \( \lambda(X^i) \) is good for every \( i \in \{0, 1, \ldots, L\} \), then \( \forall i \in I, |X^i| \geq 0.9\gamma R_i \) and \( \forall i \notin I, |X^i| \leq 1.1\gamma R_i \).

Proof. Follows directly from the construction of heavy and crucial cells in Algorithm \( \mathcal{A}_2 \) and Algorithm \( \mathcal{A}_3 \).

Lemma B.1. (Levels outside \( I \) can be ignored)

Suppose all estimated values \( \lambda(C) \) and \( \lambda(X^i) \) in Algorithm \( \mathcal{A}_2 \) and Algorithm \( \mathcal{A}_3 \) are good. Let \( X^I := \bigcup_{i \in I} X^i \). If \( o \in [1, \text{OPT}] \), then for any \( B \subset \mathbb{R}^d \) with \( |B| = k \), we have

\[
\text{cost}(X^I, B) \leq \text{cost}(X, B) \leq (1 + \epsilon/10) \cdot \text{cost}(X^I, B).
\]

The above lemma shows that we can construct a coreset of \( X^I \) instead. The coreset of \( X^I \) will automatically become a coreset of \( X \). Before proving Lemma B.1, we need to prove the following claim.

Claim B.1. Suppose all estimated values \( \lambda(C) \) and \( \lambda(X^i) \) in Algorithm \( \mathcal{A}_2 \) and Algorithm \( \mathcal{A}_3 \) are good. Let \( X^I := \bigcup_{i \in I} X^i \) and let \( \gamma \) be the same as in Algorithm \( \mathcal{A}_3 \). Suppose \( o \in [1, \text{OPT}] \). For \( i \in \{0, 1, \ldots, L\} \), if \( C \in G_{i-1} \) is marked as heavy, then we have \( |C \cap X^I| \geq (1 - 2^{p+1}(L - i)\gamma) \cdot |C \cap X| \).

Proof. The proof is by induction. Consider the base case \( i = L \). If there is no heavy cell in \( G_{L-1} \), the claim holds for \( i = L \) case. Otherwise consider any heavy cell \( C \in G_{L-1} \). According to Fact B.2, none of the children cells of \( C \) is heavy. According to the construction of crucial cells, every child cell of \( C \) is crucial. By the construction of \( X^L \), we know that \( |C \cap X^L| = |C \cap X| \). Since \( \lambda(C) \) is good, we have \( |C \cap X^I| \geq 0.9R_{L-1} \geq R_L/2^{p+1} \).

Since \( \lambda(X^L) \) is good, we have

\[
\lambda(X^L) \geq \min(|X^L| - 0.1\epsilon R_L, (1 - 0.01\epsilon)|X^L|) \\
\geq \min(|X^L \cap C| - 0.1\epsilon R_L, (1 - 0.01\epsilon)|X^L \cap C|) \\
= \min(|X \cap C| - 0.1\epsilon R_L, (1 - 0.01\epsilon)|X \cap C|) \\
\geq \gamma R_L,
\]

where the last step follows from that \( \gamma \leq 1/2^{p+2} \). Thus, by the construction of \( I \), we have \( L \in I \) and thus \( |C \cap X^I| = |C \cap X^I| = |C \cap X| \).

Now suppose the claim is true for \( i + 1, i + 2, \ldots, L \). If there is no heavy cell in \( G_{i-1} \), the claim directly holds for \( i \). Otherwise consider any heavy cell \( C \in G_{i-1} \). There are two cases. In the first case, \( i \in I \), i.e., \( X^i \subseteq X^I \). In this case, we have

\[
|C \cap X^I| = \sum_{C' \in G_i; C' \text{ is a heavy child of } C} |C' \cap X^I| + \sum_{C' \in G_i; C' \text{ is a crucial child of } C} |C' \cap X^I| \\
= \sum_{C' \in G_i; C' \text{ is a heavy child of } C} |C' \cap X^I| + |C \cap X^I| \\
\geq (1 - 2^{p+1}(L - i - 1)\gamma) \sum_{C' \in G_i; C' \text{ is a heavy child of } C} |C' \cap X| + |C \cap X^I| \\
\geq (1 - 2^{p+1}(L - i - 1)\gamma)|C \cap X| \\
\geq (1 - 2^{p+1}(L - i)\gamma)|C \cap X|,
\]
where the second step follows from \( i \in I \) and the construction of \( X^i \), the third step follows from the induction hypothesis and the forth step follows from that \((C \cap X^i) \cup \cup_{C' \in G_i \cap C'}\) is a heavy child of \( C (C' \cap X) = C \cap X \).

In the second case, \( i \notin I \), i.e., \( X^i \cap X^j = \emptyset \). In this case, we have:

\[
\sum_{C' \in G_i \cap C'} |C' \cap X| \geq |C \cap X| - |X^i| \\
\geq |C \cap X| - 1.1 \gamma \epsilon \\
\geq (1 - 2^{p+1}\gamma)|C \cap X|, \tag{B.1}
\]

where the first step follows from that each child of \( C \) is either heavy or crucial, the second step follows from that \( C' \cap X \subseteq X^i \) for any crucial cell \( C' \in G_i \), the third step follows from \( i \notin I \) and Fact B.5, and the last step follows from that \( C \in G_{i-1} \) is a heavy cell which implies that \( |C \cap X| \geq 0.9 \epsilon \).

Thus, we have

\[
|C \cap X^i| \geq \sum_{C' \in G_i \cap C'} |C' \cap X^i| \\
\geq (1 - 2^{p+1}(L - i - 1)\gamma) \sum_{C' \in G_i \cap C'} |C' \cap X| \\
\geq (1 - 2^{p+1}(L - i - 1)\gamma)(1 - 2^{p+1}\gamma)|C \cap X| \\
\geq (1 - 2^{p+1}(L - i)\gamma)|C \cap X|,
\]

where the second step follows from the induction hypothesis, the third step follows from Equation (B.1).

\[\square\]

**Proof of Lemma B.1** Now we are able to prove Lemma B.1. Observe that \( X^i \subseteq X \), it is easy to verify that \( \text{cost}(X^i, B) \leq \text{cost}(X, B) \) for any \( B \subset \mathbb{R}^d \) with \( |B| \leq k \). In the remaining of the proof, we will show that \( \text{cost}(X, B) \leq (1 + \epsilon/10) \cdot \text{cost}(X^i, B) \) for any \( B \subset \mathbb{R}^d \) with \( |B| \leq k \).

Consider an arbitrary level \( i \in \{0, 1, \cdots, L\} \setminus I \), i.e., \( X^i \cap X^j = \emptyset \). Consider a point \( x \in X^i \). By the construction of \( X^i \) by Algorithm 3, we know that \( c_{i-1}(x) \) is a heavy cell. By Claim B.1, \( |c_{i-1}(x) \cap X^i| \geq (1 - 2^{p+1}L\gamma) \cdot |c_{i-1}(x) \cap X| \). Since \( c_{i-1}(x) \) is a heavy cell, we have \( |c_{i-1}(x) \cap X| \geq 0.9 \epsilon \).

\[
|c_{i-1}(x) \cap X^i| \geq (1 - 2^{p+1}L\gamma) \cdot 0.9 \epsilon \geq 0.5 \epsilon
\]

since \( \frac{\epsilon}{40 \cdot 2^{p+1}L} \). By an averaging argument, we can find a point \( y \in c_{i-1}(x) \cap X^i \) such that

\[
\min_{b \in B} \|x - b\|_2^p \leq 2^{p-1} \cdot \left( \min_{b \in B} \|y - b\|_2^p + \|x - y\|_2^p \right) \\
\leq 2^{p-1} \cdot \left( \min_{b \in B} \|y - b\|_2^p + (\sqrt{d} \Delta_{i-1})^p \right) \\
\leq 2^{p-1} \cdot \left( \sum_{x' \in c_{i-1}(x) \cap X^i} \frac{\|x' - b\|_2^p}{|c_{i-1}(x) \cap X^i|} + (\sqrt{d} \Delta_{i-1})^p \right), \tag{B.3}
\]

where the first step follows from the convexity of \( \| \cdot \|_2^p \), the second step follows from that \( x \) and \( y \) are both in the cell \( c_{i-1}(x) \in G_{i-1} \), and the last step follows from an averaging argument.
Let \( X^N = X \setminus X^I \). We have the following:

\[
\text{cost}(X, B) = \text{cost}(X^I, B) + \text{cost}(X^N, B)
\]

\[
= \text{cost}(X^I, B) + \sum_{i \notin I} \sum_{x \in X^I} \min_{b \in B} \|x - b\|_2^p
\]

\[
\leq \text{cost}(X^I, B) + 2^{p-1} \cdot \sum_{i \notin I} \sum_{x \in X^I} \left( \frac{\sum_{x' \in c_{i-1}(x) \cap X^I} \min_{b \in B} \|x' - b\|_2^p}{|c_{i-1}(x) \cap X^I|} + (\sqrt{d}\Delta_{i-1})^p \right)
\]

\[
\leq \text{cost}(X^I, B) + 2^{p-1} \cdot \sum_{i \notin I} \sum_{x \in X^I} \left( \frac{\sum_{x' \in c_{i-1}(x) \cap X^I} \min_{b \in B} \|x' - b\|_2^p}{0.5R_{i-1}} + (\sqrt{d}\Delta_{i-1})^p \right)
\]

\[
\leq \text{cost}(X^I, B) + 2^{p-1} \cdot 4L\gamma R_i \left( \frac{\sum_{x \in X^I} \min_{b \in B} \|x - b\|_2^p}{0.5R_{i-1}} + (\sqrt{d}\Delta_{i-1})^p \right)
\]

\[
= \text{cost}(X^I, B) + 2^{p-1} \cdot 4L\gamma R_i \left( \frac{\text{cost}(X^I, B)}{0.5R_{i-1}} + (\sqrt{d}\Delta_{i-1})^p \right)
\]

\[
= \text{cost}(X^I, B) + 2^{p-1} \cdot 4L\gamma R_i \left( \frac{\text{cost}(X^I, B)}{0.5R_{i-1}} + 2^{2p+1}L\gamma \cdot 0.01o \right)
\]

\[
\leq (1 + 2^{2p+2} \cdot L\gamma)\text{cost}(X^I, B) + 2^{2p+1}L\gamma \cdot 0.01 \cdot \text{cost}(X, B),
\]

where the first step follows from \( X = X^I \cup X^N \) and \( X^I \cap X^N = \emptyset \), the second step follows from the definition of \( \text{cost}(X^N, B) \), the third step follows from Equation \((B.3)\), the fourth step follows from Equation \((B.3)\), the fifth step follows from \( (c_{i-1}(x) \cap X^I) \subseteq X^I \), the sixth step follows from that \( |X^I| \leq 1.1\gamma R_i \) for \( i \notin I \) by Fact \((B.5)\), the seventh step follows from that \( |\{0, 1, \cdots, L\} \setminus I| \leq L + 1 \leq 2L \), the eighth step follows from the definition of \( \text{cost}(X^I, B) \), the ninth step follows from that \( R_i = 0.01o/(0.5\sqrt{d}\Delta_{i-1})^p \), the tenth step follows from that \( R_i = R_{i-1} \cdot 2^p \), and the last step follows from that \( o \leq \text{OPT} \leq \text{cost}(X, B) \).

Since \( \gamma \leq \frac{\epsilon}{40 \cdot 2^{2p+1} L\gamma} \), the above inequality implies that

\[
\frac{\text{cost}(X, B)}{\text{cost}(X^I, B)} = 1 + 2^{2p+2} \cdot L\gamma \leq 1 + \epsilon/40 \leq 1 + \epsilon/10.
\]

\( \square \)

**Lemma B.2.** (Sensitivity Upper Bound of Each Point) Suppose all estimated values \( \lambda(C) \) and \( \lambda(X^i) \) in Algorithm 2 and Algorithm 3 are good. Consider \( X^I \) computed by Algorithm 3. If \( o \in [1, \text{OPT}] \), then for every \( i \in I \) and every \( x \in X^I \),

\[
\max_{B \subseteq \mathbb{R}^d: |B| \leq k} \frac{\min_{b \in B} \|x - b\|_2^p}{\text{cost}(X^I, B)} \leq \min \left( 2^{2p+1} \cdot \frac{1}{2^{2p+1} \cdot L\gamma R_i} \right).
\]

**Proof.** Since \( x \in X^I \), it is clear that \( \max_{B \subseteq \mathbb{R}^d: |B| \leq k} \frac{\min_{b \in B} \|x - b\|_2^p}{\text{cost}(X^I, B)} \leq 1 \). In the remaining of the proof, we will show that \( \max_{B \subseteq \mathbb{R}^d: |B| \leq k} \frac{\min_{b \in B} \|x - b\|_2^p}{\text{cost}(X^I, B)} \leq 2^{2p+1} \cdot \frac{1}{R_i} \).

Let \( B \subseteq \mathbb{R}^d \) be a subset of at most \( k \) centers. Consider an arbitrary point \( x \in X \). By Fact \((B.3)\), there is a unique \( i \in \{0, 1, \cdots, L\} \) such that \( x \in X_i \), i.e., there is a crucial cell in \( G_i \) that contains \( x \). Let \( C \) be \( c_{i-1}(x) \), i.e., the crucial cell that contains \( x \) is a child cell of \( C \). By the construction of crucial cells, it is easy to verify that \( C \) is a heavy cell in \( G_{i-1} \). According to Claim \((B.1)\) and Fact \((B.1)\), we have

\[(B.4)\]

\[|C \cap X^I| \geq (1 - 2^{p+1}L\gamma) \cdot |C \cap X| \geq 0.5R_{i-1},\]
where the last inequality follows from that $|C \cap X| \geq 0.9 R_{i-1}$ and $\gamma = \epsilon/(40 \cdot 2^{2p+2} \cdot L)$. According to Claim \ref{claim:convex}, the cell $C$ contains at least one point from $X^i$, i.e., $C \cap X^i \neq \emptyset$. By an average argument, we can find a point $y \in C \cap X^i$ such that
\[
\min_{b \in B} \|y - b\|_2^p \leq \frac{\sum_{x' \in C \cap X^i} \min_{b \in B} \|x' - b\|_2^p}{|C \cap X^i|}.
\]
Then we have:
\[
\min_{b \in B} \frac{\|x - b\|_2^p}{\text{cost}(X^i, B)} \leq 2^{p-1} \left( \frac{\min_{b \in B} \|y - b\|_2^p}{\text{cost}(X^i, B)} + \frac{\|x - y\|_2^p}{\text{cost}(X^i, B)} \right)
\]
\[
\leq 2^{p-1} \left( \frac{\sum_{x' \in C \cap X^i} \min_{b \in B} \|x' - b\|_2^p}{|C \cap X^i| \cdot \text{cost}(X^i, B)} + \frac{\|x - y\|_2^p}{\text{cost}(X^i, B)} \right)
\]
\[
\leq 2^{p-1} \left( \frac{\sum_{x' \in X^i} \min_{b \in B} \|x' - b\|_2^p}{|C \cap X^i| \cdot \text{cost}(X^i, B)} + \frac{\|x - y\|_2^p}{\text{cost}(X^i, B)} \right)
\]
\[
= 2^{p-1} \left( \frac{1}{|C \cap X^i|} + \frac{\|x - y\|_2^p}{\text{cost}(X^i, B)} \right)
\]
\[
\leq 2^{p-1} \left( \frac{1}{|C \cap X^i|} + \frac{(\sqrt{d} \Delta_{i-1})^p}{\text{cost}(X^i, B)} \right)
\]
\[
\leq 2^{p-1} \left( \frac{1}{|C \cap X^i|} + (1 + \epsilon/10) \cdot \frac{(\sqrt{d} \Delta_{i-1})^p}{\text{cost}(X, B)} \right)
\]
\[
\leq 2^{p-1} \left( \frac{1}{|C \cap X^i|} + (1 + \epsilon/10) \cdot \frac{(\sqrt{d} \Delta_{i-1})^p}{\text{OPT}} \right)
\]
\[
\leq 2^{p-1} \left( \frac{1}{|C \cap X^i|} + (1 + \epsilon/10) \cdot \frac{(\sqrt{d} \Delta_{i-1})^p}{\text{OPT}} \right)
\]
\[
= 2^{p-1} \left( \frac{1}{|C \cap X^i|} + (1 + \epsilon/10) \cdot \frac{1}{100 \cdot R_{i-1}} \right)
\]
\[
\leq 2^{p+1} \left( \frac{1}{0.5 R_{i-1}} + (1 + \epsilon/10) \cdot \frac{1}{100 \cdot R_{i-1}} \right)
\]
\[
\leq 2^{2p+1} \cdot \frac{1}{R_{i-1}}
\]
where the first step follows from the convexity of $\|x\|_2^p$, the second step follows from Equation \ref{eq:main}, the third step follows from $(C \cap X^i) \subseteq X^i$, the forth step follows from that $\text{cost}(X^i, B) = \sum_{x' \in X^i} \min_{b \in B} \|x' - b\|_2^p$, the fifth step follows from that both $x, y$ are in the cell $C \subseteq G_{X^i}$ and the side length of $C$ is $\Delta_{i-1}$, the sixth step follows from Lemma \ref{claim:convex} that $\text{cost}(X, B) \leq (1 + \epsilon/10) \cdot \text{cost}(X^i, B)$, the seventh step follows from $\text{OPT} \leq \text{cost}(X, B)$, the eighth step follows from $\text{OPT} \leq \text{OPT}$, the ninth step follows from $R_{i-1} = 0.01 \cdot \epsilon/((\sqrt{d} \Delta_{i-1})^p)$, the tenth step follows from Equation \ref{eq:main}, and the twelfth step follows from $R_{i-1} = R_{i-1} \cdot 2^p$. \hfill \Box

**Proof of Theorem 5.1** Now, we are going to prove Theorem 5.1. Before we proceed to more details, we need to introduce the sensitivity sampling theorem. Given a point set $X \subseteq \mathbb{R}^d$, the sensitivity of a point $x \in X$ with respect to the $\ell_p$ $k$-clustering problem is defined as
\[
s(x) := \sup_{B \subseteq \mathbb{R}^d: |B| \leq k} \frac{\min_{b \in B} \|x - b\|_2^p}{\text{cost}(X, B)}.
\]
The following theorem shows the guarantee of the sensitivity sampling based core set construction.
Theorem B.1. Let $X \subset \mathbb{R}^d$ be a set of at most $n$ points. Let $s' : X \to \mathbb{R}_{\geq 0}$ satisfy that $\forall x \in X$, $s'(x)$ is an upper bound of the sensitivity of $x$, i.e., $s'(x) \geq s(x)$. Let $\hat{t} = \sum_{x \in X} s'(x)$. Let $\delta, \epsilon \in (0, 0.5)$. Consider a multiset $S$ of $m$ i.i.d. samples from $X$, where each sample chooses $x \in X$ with probability $\Prob(x) = s'(x)/\hat{t}$. For each sampled point $x$, we assign it an arbitrary weight $w(x) \in (1 \pm \epsilon/4) \cdot 1/(m \cdot \Prob(x))$. If $m \geq \left(50t/\epsilon^2 \cdot (\log n \log \hat{t} + \log(1/\delta)) \right)$, $(S, w)$ is an $\epsilon$-coreset for $X$ with probability at least $1 - \delta$.

Consider the sampling procedure in Algorithm 3. Each sample $x \in X^i \subseteq X^I$ is drawn with probability

$$
\Prob(x) = \frac{x \cdot \min(2^{2p+1} / R_i, 1)}{\sum_{j \in I} x \cdot \min(2^{2p+1} / R_j, 1)} = \frac{\lambda(X^i) \cdot \min(2^{2p+1} / R_i, 1)}{\sum_{j \in I} \lambda(X^j) \cdot \min(2^{2p+1} / R_j, 1)}.
$$

The weight $w(x)$ of the sample is

$$
w(x) = \frac{t'}{m \cdot \min(2^{2p+1} / R_i, 1)} = \frac{1}{m} \cdot \frac{\sum_{j \in I} x \cdot \min(2^{2p+1} / R_j, 1)}{\lambda(X^i) / |X^i|} \cdot \frac{\lambda(X^i)}{|X^i|}.
$$

Thus, we have $w(x) = \frac{\lambda(X^i)}{|X^i|} \cdot 1/(m \cdot \Prob(x))$. Since $i \in I$, we have $|X^i| \geq 0.99 \gamma R_i$ by Fact B.5. Since $\lambda(X^i)$ is a good estimated value of $|X^i|$, we have $\lambda(X^i)/|X^i| \in (1 \pm \epsilon/8)$. Therefore, we have $w(x) \in (1 \pm \epsilon/8) \cdot 1/(m \cdot \Prob(x))$. Let $s'(x) := \frac{\lambda(X^i)}{|X^i|} \cdot \min(2^{2p+1} / R_i, 1)$. Since $\lambda(X^i)/|X^i| \geq 1/2$, we have $s'(x) \geq s(x)$ by Lemma B.2. Let $\hat{t} = \sum_{x \in X^i} s'(x)$. We have $\hat{t} = 2 \cdot t'$ and $\forall x \in X^i, \Prob(x) = s'(x)/\hat{t}$. According to Theorem B.1, $(S, w)$ outputted by Algorithm 3 is an $(\epsilon/2)$-coreset for $X^I$ with probability at least $1 - \delta$. According to Lemma B.1, when $(S, w)$ is an $(\epsilon/2)$-coreset for $X^I$, $(S, w)$ is an $\epsilon$-coreset for $X$. \qed

C \quad \Omega(k) \text{ Space is Necessary for } k\text{-Clustering}

There is a simple reduction from a communication problem to the $k$-clustering problem to show that any multiplicative approximate sliding window algorithm for $k$-clustering needs at least $\Omega(k)$ space. In the INDEX [KN06] problem, Alice has a $k$-bit binary string and Bob has an index $i \in [k]$. Alice can send a message to Bob, and Bob needs to answer whether the $i$-th bit of the Alice’s string is 1 without sending any message to Alice. If Bob needs to answer correctly with probability at least $2/3$, Alice must send $\Omega(k)$ bits to Bob. Now consider the reduction to the sliding window $k$-clustering in 1-dimensional space. Let the window size be $k + 1$. For the $j$-th bit of Alice’s string, if the bit is 0 Alice adds a point $2 \cdot j - 1$ into the stream, and if the bit is 1 Alice adds a point $2 \cdot j$ into the stream. Then Alice simulates the sliding window algorithm and sends the memory to Bob. Bob adds a point $2 \cdot i$ into the stream, and continues simulating the sliding window algorithm. If the $i$-th bit is 1, then there are only $k$ distinct points in the stream and the clustering cost is 0. If the $i$-th bit is 0, then there are $k + 1$ distinct points in the stream, and the clustering cost is at least 1. Thus, if the sliding window algorithm can output any multiplicative approximation, Bob can determine whether the $i$-th bit of Alice’s string is 1. Thus, the sliding window algorithm for $k$-clustering needs $\Omega(k)$ space.

D \quad \text{Dimension Reduction for } k\text{-Clustering}

According to MMR19, there is a simple way to reduce the dimension of points for $\ell_p$ $k$-clustering problems.

Lemma D.1. (Theorem 1.3 in MMR19) Consider $\ell_p$ $k$-clustering problem for $p \geq 1$. Consider an arbitrary point set $P = \{p_1, p_2, \cdots, p_n\} \subset \mathbb{R}^d$. Let $\epsilon \in (0, 0.5)$. Let $S \in \mathbb{R}^{d' \times d}$ be a random matrix where $d' = \Theta(\epsilon^{-2} \log k)$ and each entry of $S$ is drawn uniformly at random from $\{-1/\sqrt{d'}, 1/\sqrt{d'}\}$. Let $P' = \{S \cdot p_1, S \cdot p_2, \cdots, S \cdot p_n\}$. Then, with probability at least 0.99,

$$(1 - \epsilon) \cdot \min_{Z \subset \mathbb{R}^{d'}} \cost(P, Z) \leq \min_{Z \subset \mathbb{R}^{d'}} \cost(P', Z') \leq (1 + \epsilon) \cdot \min_{Z \subset \mathbb{R}^{d'}} \cost(P, Z).$$

According to the above lemma, we only need to solve the clustering problem for the point set $P'$. For a sliding window algorithm, we can sample a matrix $S$ before the algorithm starts. For each point $p$ during the stream, we can compute $S \cdot p$ and apply the algorithm for $S \cdot p$. Thus, the dimension of a point that needed to be handled is at most $O(\log(k)/\epsilon^2)$. The entire algorithm only needs $O(d' \cdot d) = O(kd/\epsilon^2)$ additional space to store $S$. 3042