Thesis for the degree
Master of Science

עבודת גמר (תזה) לתואר
מוסמך למדעים

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
David Reitblat

מאת
דוד רייטבלט

Sliding-Window Streaming Algorithms for Graph Problems and ℓ_p-Sampling

אלגוריתמים במודל של זרם נתונים עם חלון הזזה עבור בעיות גרפים ודגימה בהתפלגות ℓ_p

Advisor:
Prof. Robert Krauthgamer

מנחה:
פרופ' רוברט קראוטגמר

January 2019

טבת ה'תשע"ט

Sliding-Window Streaming Algorithms for Graph Problems and ℓ_p-Sampling

Abstract

We study algorithms for the sliding-window model, an important variant of the data-stream model, in which the goal is to compute some function of a fixed-length suffix of the stream. We focus on two settings, of frequency-vectors and of graph streams, and develop algorithms for several problems, including ℓ_p-sampling and maximum-matching, all in the sliding-window model.

We first extend the smooth-histogram framework of Braverman and Ostrovsky [BO07] to a more general family of *almost-smooth* functions. We show that both the vertex-cover size and the maximum-matching size are 2-almost-smooth, and thus can be approximated using our framework in the sliding-window model. Another example application, we show a $\operatorname{polylog}(n)$ space algorithm estimating the maximum-matching size in bounded-arboricity graphs with n vertices, whose approximation ratio differs by a factor of 2 from the insertion-only algorithm of McGregor and Vorotnikova [MV18].

Second, we design algorithms based on the adaptation by Crouch, McGregor, and Stubbs [CMS13] of the smooth-histogram framework to the graph-streaming model. This includes a 2.164-approximation algorithm for maximum-matching in the vertex-arrival model with $O(n \log^2 n)$ bits of storage, and also the first parameterized sliding-window algorithms for exact maximum-matching and vertex-cover which have space bound $O(k^2 \log^3 n)$, and are parameterized by the maximum-matching size k.

In the frequency-vector model, we show an algorithm for ℓ_p-sampling, i.e., sampling a coordinate from the frequency-vector $x \in \mathbb{R}^n$ with distribution close to the so-called ℓ_p*-distribution*, defined as follows. For $p = 0$ it is a uniform distribution over the non-zero coordinates of x, and for $p > 0$ it has the probability density function $f_p(i) = \frac{|x_i|^p}{\|x\|_p^p}$ for $i \in [n]$. Our ℓ_0-sampling algorithm in the sliding-window model has space bound $\operatorname{poly}\left(\varepsilon^{-1} \cdot \log \frac{n}{\delta}\right)$, and its output distribution is within total variation distance $\delta + \varepsilon$ of the ℓ_0-distribution. We then generalize it to ℓ_p-sampling for every $p \leq 2$, with similar performance guarantees.

Acknowledgements

Upon completing this two year journey, it is with great pleasure that I thank the people around me who have helped and supported me.

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Robert Krauthgamer, who made this work possible. I am thankful for his scientific guidance and expert advice. His immense knowledge and insightful comments had widen my research from various perspectives. Robi, your meticulous demands for my formulations significantly improved my writing! Thanks for everything.

I would also like to thank the Weizmann Institute for providing such an amazing research atmosphere, and for all their support.

I wish to thank my friends from the faculty, Yotam Amar, Roman Beliy, Arnold Filtser, Yevgeny Levanzov, Havana Rika, Roi Sinoff, Ohad Trabelsi, and many others; together they created a joyful atmosphere which made it impossible to study.

I would also like to thank my friends, Shahar Barr, Shani Drori, Avihay Steve Emanuel, Pavel Krivosheev, Oleg Shichelman, Katya Tubis, and many others, for providing plenty of reasons to do anything but research. I take this opportunity to thank them.

Special thanks are due to my loving and supportive partner, Tanya Patroul, for her unending support and understanding. My thanks are extended to my dog, Chebyshev (Chubby), for many long random walks which enable me to relax and clear my mind.

Last, but certainly not least, I wish to thank my family for their love, support and encouragement. You are a great source of support.

Contents

1 Introduction

Nowadays, there is a growing need for algorithms to process huge data sets. The internet, including social networks and electronic commerce, together with astronomical and biological data, provide new challenges for computer scientists and mathematicians, since traditional algorithms are not able to handle such massive sets of data in a reasonable time. First, the data set is too big to be stored on a single machine. Second, even algorithms with time complexity $O(n^2)$ are too slow for various practical uses. Third, and most important, over time the data could change, and algorithms should cope with dynamic changes of the data. Therefore, several models of computation for "Big Data" are researched, such as distributed algorithms and streaming algorithms.

We will concentrate on the *streaming model* (see e.g. [Mut05, BBD$^+$02, Agg07]), where the data is given as a sequence of items (or updates), in some predetermined (usually adversarial) order, and the algorithm can read the data only in that order. Often, the algorithm can only read the data once, although there are also algorithms for multiple passes. More concretely, a data *stream* $\mathcal{S}$ is a sequence (possibly infinite) $\mathcal{S} = \langle \sigma_1, \sigma_2, \ldots, \sigma_i, \ldots \rangle$, where each item σ_i belongs to some universe U. The length of the stream, as well as the size of U, is assumed to be huge, such that storing the entire stream, or even a constant-size information for each item in $\mathcal{S}$, is impractical. A streaming algorithm $\mathcal{A}$ takes $\mathcal{S}$ as input and computes some function f of the stream $\mathcal{S}$. A *query* at a time t to $\mathcal{A}$ is a request to output (an approximation to) $f(\mathcal{S}_t)$. If the stream is infinite then we assume that at every time t the algorithm could calculate f on the prefix $\mathcal{S}_t = \langle \sigma_1, \sigma_2, \ldots, \sigma_t \rangle$, if it is queried at time t. Note that algorithm $\mathcal{A}$ has access to the input in a *streaming fashion*, i.e., $\mathcal{A}$ can read the input once and only in the order it is given.

Typically, storing the whole stream and computing the exact value of f is computationally expensive or even impossible. Hence, we can only expect approximation algorithms. Formally, the goal is to design a streaming algorithm $\mathcal{A}$ with the following constraints. The space complexity should be $\operatorname{polylog}(n)$, where n is the size of the universe U, or some other measure of the stream, e.g. its length. The time complexity of processing a single update from the stream, referred to as the *update time*, should be small, $\operatorname{polylog}(n)$ or even constant. The algorithm's output should approximate f, where what constitutes a good approximation depends on the concrete function f.

There are two variants of the streaming model, namely, the insertion-only model and the insertion-deletion model, also known as the turnstile model. In the *insertion-only* model, all updates are positive, only adding items to the underlying structure. In the *turnstile* model, deletion of previously added items is also allowed, which means that data updates σ_i are of the form $\sigma_i = (b_i, \sigma'_i)$ where $b_i \in \{\pm 1\}$ indicates if the item is inserted $(+1)$ or deleted (-1).

The *sliding-window* model has become a popular model for processing (infinite) data streams, where older data items should be ignored, as they are considered obsolete. Introduced by Datar, Gionis, Indyk and Motwani [DGIM02], this model captures the idea that one wishes to calculate some desired function only on the most recent items.

In this model, the goal is to compute a function f on a suffix of the stream W, referred to as the *active window*. Items in W are called *active*, and older items from the stream are called *expired*. Throughout, the size of the active window W, denoted by w, is assumed to be known (to the algorithm) in advance. At a point of time t, we denote the active window of the last w elements of the stream by $W_t = \langle \sigma_{t-w+1}, \ldots, \sigma_t \rangle$, or W for short when t is clear from the context. At time t, the goal is to approximate $f(W_t)$, and possibly provide a corresponding object, e.g. a feasible matching in a graph when the stream is a sequence of edges and f is the maximum-matching size.

Datar et al. [DGIM02] noted that in the sliding-window model there is a lower bound of $\Omega(w)$ if deletions are allowed, even for relatively simple tasks like approximating (within a factor of 2) the number of distinct items in a stream. Since our goal is to develop algorithms for the sliding-window model, we will only study insertion-only streams. Therefore, we assume throughout that the stream $\mathcal{S}$ has only insertions, and no deletions. We focus on two different (insertion-only) streaming models, the graph streaming model and the frequency-vector model.

Another closely related model is the *timestamp-based sliding-window* model, in which each data item from the stream has an additional parameter, a timestamp of its arrival time, i.e., the stream consists of pairs $\sigma_i = (a_i, t_i)$ which means that item a_i arrives at time t_i. For short we call it the timestamp model. In this model, the active window is defined using a time bound T, and denoted by W^T, indicating that every edge arriving in the last T time units is active, and older edges should be ignored, as they are considered obsolete. Namely, at time t the active window is defined as $W_t^T = \{\sigma_i = (a_i, t_i) \,|t_i \geq t - T\}$. We will use the timestamp model only in Section 3.4, and, unless stated otherwise, when referring to the sliding-window model we mean the fixed-length (non-timestamp-based) sliding-window. The timestamp model was studied by Datar et al. [DGIM02] as well as by many more, e.g. [BOZ09, GL08, BDM02].

1.1 Graph-Streaming Model

A widely studied streaming model is the *graph-streaming* model (see e.g. [FKM+05, McG14]), where the stream $\mathcal{S}$ consists of a sequence of edges (possibly with some auxiliary information, like weights) of an underlying graph $G = (V, E)$. We assume that $V = [n]$ for a known value $n \in \mathbb{N}$ and G is a simple graph without parallel edges. The graph-streaming model is typically studied in the *semi-streaming* model, where algorithms are allowed to use $O(n \cdot \operatorname{polylog}(n))$ space. Observe that for dense graphs in the semi-streaming model the algorithm can not store the whole graph, but is able to store $\operatorname{polylog}(n)$ information for each vertex.

Algorithms in the graph-streaming model compute or approximate some quantity of the graph. We concentrate on the following two related optimization problems, maximum-matching and minimum vertex-cover.

Definition 1.1. A *matching* in a graph $G = (V, E)$ is a set of disjoint edges $M \subseteq E$, i.e., no two edges have a common vertex. Denote by $m(G)$ the size of a matching with maximal number of edges.

A matching with a maximal number of edges is called a *maximum-cardinality matching*, and is usually referred to as a *maximum-matching.* In an edge-weighted graph G a *maximum-weight matching* is a matching with maximal sum of weights.

Definition 1.2. A subset $C \subseteq V$ of the vertices of the graph $G = (V, E)$ is called a *vertex-cover* of G if each edge $e \in E$ is incident to at least one vertex in C. Denote by $VC\left(G\right)$ the smallest size of a vertex-cover of G.

We will use the terminology of Feige and Jozeph [FJ15] to distinguish between estimation algorithms and approximation algorithms for optimization problems (where the goal is to find a feasible solution of optimal value). An *approximation algorithm* is required to output a feasible solution whose value is close to the value of an optimal solution, e.g., output a feasible matching of near-optimal size. An *estimation algorithm* is required to only output a value close to that of an optimal solution, without necessarily outputting a corresponding feasible solution, e.g., output an approximate size of a maximum-matching in a graph, without a corresponding matching.

For ease of exposition, we sometime use the following notation of asymptotic complexity to hide less important factors.

Definition 1.3. The notation $\widetilde{O}(s\left(n\right))$ hides poly-logarithmic dependence on $s\left(n\right)$, i.e., $\widetilde{O}(s\left(n\right)) = O(s\left(n\right) \cdot \operatorname{poly}\log s\left(n\right))$. To suppress dependence on ε we denote $O_\varepsilon(s\left(n\right)) = O(s\left(n\right) \cdot f\left(\varepsilon\right))$, for some positive function $f : \mathbb{R}^+ \to \mathbb{R}^+$.[1] We also combine both notations and define $\widetilde{O}_\varepsilon(s\left(n\right)) = O_\varepsilon(s\left(n\right) \cdot \operatorname{poly}\log s\left(n\right))$.

For maximum-matching, the best approximation semi-streaming algorithm known for insertion-only streams of general graphs is a 2-approximation by a greedy algorithm, which maintains a maximal matching of the stream using $\widetilde{O}(n)$ space [FKM+05]. There is a known lower bound $\frac{e}{e-1} \approx 1.58$ on the approximation factor by any insertion-only algorithm that uses space $\widetilde{O}\left(n\right)$ [Kap13].

Assadi et al. [AKL17] showed a space lower bound $RS\left(n\right) \cdot n^{1-O(\varepsilon)}$ for $(1+\varepsilon)$-estimation of maximum-matching for general graphs in insertion-only streams, where $RS\left(n\right)$ denotes the maximum number of edge-disjoint induced matchings of size $\Theta\left(n\right)$ in an n-vertex graph [RS78]. There is also active research on estimating the maximum-matching size in a restricted family of graphs, the bounded-arboricity graphs. A graph $G = (V, E)$ has *arboricity* α if its set of edges E can be partitioned into at most α forests. A long line of research [ETHL+18, MV16, BS15, CCE+16], culminating in the result of Cormode et al. [CJMM17], and then slightly improved by McGregor and Vorotnikova [MV18], has shown a polylog $\left(n\right)$ space algorithm for estimating the maximum-matching size in bounded-arboricity graphs within factor $O(\alpha)$.

There is a special well-studied model for bipartite graphs, the *vertex-arrival* model, in which we have an additional assumption on the order of the stream. In this model, the underlying graph is bipartite and all edges incident on a vertex from the left side of the graph arrive consecutively in the stream, i.e., for a bipartite graph $G = (V, U, E)$

[1]Throughout, every dependence on ε is polynomial, i.e., in our case $O_\varepsilon(s\left(n\right)) = O(s\left(n\right) \cdot \operatorname{poly}\left(\varepsilon^{-1}\right))$.

a vertex-arrival stream consists of some permutation of the vertices on the left side $V = \{v_1, v_2, \cdots, v_n\}$ and the stream is $\mathcal{S} = \langle E(v_1), E(v_2), \cdots, E(v_n)\rangle$, where for a vertex $v \in V$ its set of edges denoted by $E(v) = \{(v,u) \mid u \text{ is a neighbor of } v\}$. For every vertex $v \in V$ the set of edges $E(v)$ arrives in an arbitrary order. For any stream segment $X_{i,j} = (E(v_i), \cdots, E(v_j))$, for $1 \leq i \leq j \leq n$, define $E(X_{i,j}) = \bigcup_{k=i}^{j} E(v_k)$. Note that the stream consists of edges, in the aforementioned order, and the length of the segment $X_{i,j}$ is the total number of edges in it, i.e., $\sum_{k=i}^{j} |E(v_k)|$. Kapralov [Kap13] showed an optimal deterministic approximation algorithm for the vertex-arrival model with approximation factor $\frac{e}{e-1} \approx 1.58$ and space bound $\widetilde{O}(n)$.

Crouch, McGregor and Stubbs [CMS13] initiated the study of graph problems in the sliding-window model. They showed algorithms for several basic graph problems, such as k-connectivity, bipartiteness, sparsification, minimum spanning tree and spanners. They also showed an approximation algorithm for maximum-matching and for maximum-weight matching.

1.2 Frequency-Vector Model

Another extensively researched streaming model is the *frequency-vector* model, where the stream $\mathcal{S}$ is composed of additive updates to an underlying n-dimensional vector $x \in \mathbb{R}^n$, i.e., each item $\sigma_i = (j_i, a_i)$ represents increments to coordinate $j_i \in [n]$ of x by $a_i \in \mathbb{R}$. Hence, the *frequency-vector* $x \in \mathbb{R}^n$ of the stream $\mathcal{S}$ is defined, for every $j \in [n]$, as $x_j = \sum_{j_i = j} a_i$. Usually, the goal in this model is to compute some function of the frequency-vector x, e.g., the ℓ_p-norm $\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$ or the F_p-moment $F_p(x) = \|x\|_p^p$ for $p > 0$. The support of a vector $x \in \mathbb{R}^n$ is defined to be $\operatorname{supp}(x) = \{i \in [n] \mid x_i \neq 0\}$, and the ℓ_0-norm[2] (and equivalently the F_0-moment) of x is defined as its support size $\|x\|_0 = |\operatorname{supp}(x)|$. Note that it is also referred to as the distinct number of items (in the stream), since the two quantities are identical.

The insertion-only model in the setting of frequency-vector means that only positive updates are allowed, i.e., $a_i > 0$ for all $i \geq 1$, while the turnstile model means that both positive and negative updates are allowed. Often, in the insertion-only model, all updates are implicitly assumed to be 1, and in that case the stream consists only of the indices being updated, i.e., $\sigma_i = j_i \in [n]$ for every i.

We use the following notion for randomized approximation algorithms.

Definition 1.4. For $\varepsilon, \delta \in [0,1)$ and $C \geq 1$, a randomized algorithm Λ is said to $((1+\varepsilon)C, \delta)$-approximate a function f if on every input stream $\mathcal{S}$, its output $\Lambda(\mathcal{S})$ satisfies

$$\Pr\left[(1-\varepsilon) f(\mathcal{S}) \leq \Lambda(\mathcal{S}) \leq (1+\varepsilon) C \cdot f(\mathcal{S})\right] \geq 1 - \delta.$$

[2]Although ℓ_0 is not a norm, since it does not satisfy absolute scalability, we shall refer to it as such, for simplicity.

If Λ is a deterministic algorithm, then $\delta = 0$, and we say for short that it $(1+\varepsilon)\,C$-approximates f if $(1-\varepsilon)\,f(\mathcal{S}) \le \Lambda(\mathcal{S}) \le (1+\varepsilon)\,C \cdot f(\mathcal{S})$.

Remark 1.5. Even for randomized approximation algorithms we sometime use the above notation omitting δ, when the algorithm outputs $(1+\varepsilon)\,C$-approximation with some constant high probability.

The frequency-vector model began with the influential paper of Alon, Matias, and Szegedy [AMS96], who studied the problem of approximating the F_p-moments in the turnstile model. For $p = 0, 2$ they showed a $(1+\varepsilon)$-approximation algorithm with $O_\varepsilon(\text{polylog}\,(n))$ space, and for $p \ge 3$ they showed a $(1+\varepsilon)$-approximation algorithm with space bound $O_\varepsilon(n^{1-\frac{1}{p}})$. They also provided a space lower bound $\Omega\left(n^{1-\frac{5}{p}}\right)$ for all $p \ge 3$, even for the more restrictive insertion-only model. A tremendous amount of research was dedicated for improving the lower and upper bounds for $(1+\varepsilon)$-approximating the F_p-moments in the turnstile streaming model, e.g., [CKS03, BJKS04, CK04, IW05, Ind06, Li08, AKO11, Gan15]. For $p > 2$, Indyk and Woodruff [IW05] presented algorithms that use $\widetilde{O}_\varepsilon(n^{1-\frac{2}{p}})$ space in the turnstile model, which matches the lower bound previously shown by Bar-Yossef, Jayram, Kumar and Sivakumar [BJKS04], which holds even for the insertion-only model. For $p \in [0, 2]$, Indyk [Ind06] presented an algorithm that approximates the ℓ_p-norm using $O_\varepsilon(\text{polylog}\,(n))$ space in the turnstile model.

Another well-studied problem in the frequency-vector model is *ℓ_p-sampling*, where the goal is to sample a coordinate of the frequency-vector x according to its contribution to the ℓ_p-norm, i.e., the ℓ_p-distribution, defined as follows. For $p = 0$, the *ℓ_0-distribution* of a vector $x \in \mathbb{R}^n$ is a uniform distribution over the non-zero coordinates of x. See, e.g. [CF14] for a detailed study of ℓ_0-sampling. For $p > 0$, the *ℓ_p-distribution* of $x \in \mathbb{R}^n$, is defined using the probability density function $f_p^x(i) = \frac{|x_i|^p}{\|x\|_p^p}$ for every $i \in [n]$.

Sometimes, it is not known how to sample exactly according to the ℓ_p-norm distribution, thus a commonly used relaxation is to sample according to a "close" distribution, where close is measured either by using a multiplicative $1+\varepsilon$ factor or by the total variation distance (or both).

Definition 1.6. For a vector $x \in \mathbb{R}^n$ and an accuracy parameter $\varepsilon > 0$ we say that a distribution $\mathcal{D}$ over $[n]$ is a *$(1+\varepsilon)$-approximate ℓ_p-distribution* of the vector x, for $p > 0$, or *$(1+\varepsilon)$ -ℓ_p-distribution* for short, if its probability density function f satisfies

$$\forall i \in [n]\,, \quad f(i) \in (1 \pm \varepsilon)\frac{|x_i|^p}{\|x\|_p^p}.$$

Definition 1.7. The *total variation distance* of two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathrm{d_{TV}}(\mathcal{D}_1, \mathcal{D}_2)$, is defined as the largest possible difference between the probabilities assigned by the two distributions to the same event.

Usually, for two random variables X and Y distributed according to distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, we write $\mathrm{d_{TV}}(X, Y)$ when we mean $\mathrm{d_{TV}}(\mathcal{D}_1, \mathcal{D}_2)$, i.e., $\mathrm{d_{TV}}(X, Y) = \sup_A |\Pr[X \in A] - \Pr[Y \in A]|$ where the supremum is taken over all possible events A.

Definition 1.8. We say that an algorithm S is a $(1+\varepsilon,\delta)$ -*ℓ_p-sampling algorithm* if its output distribution is within total variation distance δ from a $(1+\varepsilon)$ -ℓ_p-distribution.

Remark 1.9. When $\delta = n^{-C}$ for some constant $C > 0$ we omit it and say that algorithm S is $(1+\varepsilon)$ -ℓ_p-sampling, even though its output is only within total variation distance n^{-C} from a $(1+\varepsilon)$ -ℓ_p-distribution.

The ℓ_p-sampling problem was introduced by Monemizadeh and Woodruff [MW10], who gave a $(1+\varepsilon)$ -ℓ_p-sampling algorithm with space poly$(\varepsilon^{-1}\log n)$, for $p \in [1,2]$. It was improved by Andoni, Krauthgamer and Onak [AKO11], to space $O(\varepsilon^{-p}\log^3 n)$, for $p \in [1,2]$. Jowhari, Sağlam and Tardos [JST11] further improved the space bound and presented an algorithm with near-optimal bound for $p \in [0,2]$, and showed a tight space lower bound $\Omega\left(\log^2 n\right)$ (for constant ε).

Recently, Jayaram and Woodruff [JW18] showed an ℓ_p-sampling algorithm for every $0 < p \leq 2$ with accuracy parameter $\varepsilon = 0$, which means their algorithm's output is within total variation distance δ of an ℓ_p-distribution of the frequency-vector. Their algorithm has space bound $O(\log^2 n \log \frac{1}{\delta})$ for $p \in (0,2)$ and $O(\log^3 n \log^2 \frac{1}{\delta})$ for $p = 2$.

1.3 Smooth Histogram Framework

The smooth histogram technique presented by Braverman and Ostrovsky [BO07] is one of only two general techniques for adapting insertion-only algorithms to the sliding-window model. The other one is an earlier technique called exponential histogram, due to Datar et al. [DGIM02]. The approach of [BO07] was to maintain several instances of an insertion-only algorithm on different suffixes of the stream, such that at any point in time, the algorithm can output an approximation of f on W. They showed that for a large family of functions, referred by them as smooth functions, this approach yields a good approximation algorithm for the sliding-window model.

More precisely, assume there is an algorithm Λ that $(1+\varepsilon)$-approximates a monotone non-decreasing function f in the insertion-only model. In the smooth histogram framework, the algorithm for the sliding-window model maintains $k = O(\varepsilon^{-1}\log w)$ instances of Λ. Each instance Λ_i processes the stream from some initial point in time until the end of the stream (or until instance Λ_i is discarded), i.e., it corresponds to some suffix of the stream, referred to as a bucket. The bucket corresponding to Λ_i is denoted by B_i, and we denote by $\Lambda_i(B_i)$ the value of instance Λ_i on the stream B_i. These buckets will satisfy the invariant $B_1 \supseteq W \supsetneq B_2 \supsetneq B_3 \supsetneq \cdots \supsetneq B_k$, where W is the active window. In order to use only a small amount of space, whenever two nonadjacent instances have “close” values, all instances between them will be deleted. Instances Λ_i and Λ_j, for $j > i$, are considered close if the difference between $\Lambda_i(B_i)$ and $\Lambda_j(B_j)$ is smaller than $\alpha(\varepsilon)\cdot\Lambda_i(B_i)$, where $\alpha(\varepsilon)$ is some function of ε depending only on the function f. At each step of receiving a new item from the stream, the sliding-window algorithm updates all the instances, creates a new instance Λ_{k+1}, which initially contains only the new item, and deletes all unnecessary instances, as explained above, and lastly renumber the buckets.

Braverman and Ostrovsky [BO07] showed that this approach applies to many "nice" functions. Their idea of niceness is captured by the definition of smoothness. For disjoint segments A, B of a stream, we denote by AB their concatenation. Intuitively, a monotone non-decreasing function f defined on streams is smooth if its value on two segments of the stream remains close whenever another segment of a stream is appended to it, i.e., if $f(B) \approx f(AB)$ then for every stream C also $f(BC) \approx f(ABC)$. See Remark 3.4 for a more formal description of their definition. They proved that all ℓ_p-norms, for $p > 0$, are smooth, and consequently presented algorithms that $(1+\varepsilon)$-approximate these norms in the sliding-window model, with an overhead (relative to insertion-only algorithm) in the space complexity of $O(\varepsilon^{-1} \log w)$.

1.4 Contributions

We use extensively the smooth histogram technique of Braverman and Ostrovsky [BO07] explained above. We extend their definition of smoothness to almost-smooth functions, and present approximation algorithms in the sliding-window model for several problems that have a constant-approximation algorithm.

Remark 1.10. Throughout, space complexity refers to the storage requirement of an algorithm during the entire input stream, measured in bits. Update time refers to the time complexity of processing a single update from the stream (in the RAM model).

1.4.1 Almost-Smooth Functions and Application to Graph Problems

We adapt the smooth histogram technique of Braverman and Ostrovsky [BO07] to functions that are almost smooth. Informally, we say that a monotone non-decreasing function f is d-almost-smooth if suppose $f(AB) \approx f(B)$ then appending any segment C will maintain this approximation, up to a multiplicative factor d. For a more formal and general definition see Definition 3.1 and Remark 3.2 after it. For example, the maximum-matching size is 2-almost-smooth, as proven in Lemma 3.5, which means that if $m(B)$ is a $(1+\varepsilon)$-approximation of $m(AB)$, then for every sequence of edges C it also holds that $m(BC)$ would $(1+\varepsilon)\,2$-approximate $m(ABC)$.

For an almost-smooth function with an approximation algorithm in the insertion-only model we show in Theorem 3.7 a general way of transforming it to the sliding-window model; below is a less general and formal description of it.

Theorem 1.11 (Informal version of Theorem 3.7)**.** *Suppose the function f is d-almost-smooth and can be C-approximated by an insertion-only randomized algorithm Λ. Then there exists a sliding-window algorithm Λ^{sw} that $(1+\varepsilon)\,dC^2$-approximates f, with only a factor $O(\varepsilon^{-1} \log w)$ larger space and update time.*

Using our generalization of the smooth histogram technique we provide several algorithms for estimating maximum-matching and vertex-cover, in bounded-arboricity graphs (see Tables 1 and 2 for highlights of results). In particular, we show the following theorem for maximum-matching in Section 3.2.

Theorem 1.12. *For every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there is a sliding-window $((2+\varepsilon)(\alpha+2), \delta)$-estimation algorithm for the maximum-matching size in a graph with arboricity α, with space bound $O(\varepsilon^{-3}\log^3 n \log \frac{1}{\varepsilon\delta})$ and update time $O(\varepsilon^{-3}\log^3 n \log \frac{1}{\varepsilon\delta})$.*

For maximum-matching in bounded arboricity graphs we compare in Table 1 our sliding-window algorithm (Theorem 1.12) with the insertion-only algorithms of [CJMM17] and [MV18].

Stream	Approx.	Space	Reference
insertion-only	$22.5\alpha + 6$	$O(\alpha \log^2 n)$	[CJMM17]
insertion-only	$(\alpha+2)+\varepsilon$	$O_\varepsilon(\log^2 n)$	[MV18]
sliding-window	$2(\alpha+2)+\varepsilon$	$O_\varepsilon(\log^4 n)$	Theorem 1.12

Table 1: Estimation algorithms for maximum-matching in graphs with arboricity $\le \alpha$ (considering constant probability of success).

We design also several algorithms for (estimation and approximation of) vertex-cover. We summarize in Table 2 our results and compare them to previous algorithms by [vH16]. For general graphs, we compare our sliding-window approximation algorithm (Theorem 3.20) with the previously known one, showing an improvement in the approximation ratio, essentially from 8 to 4, while the space complexity is the same. For VDP (vertex-disjoint paths[3]) and forest graphs (arboricity equals 1) we compare our sliding-window estimation algorithms to one another as well as to the turnstile estimation algorithm of [vH16]. In Theorem 3.18 the space complexity is much better, $O_\varepsilon(\log^3 n)$ compared to $\widetilde{O}_\varepsilon(\sqrt{n})$ in the other two, Theorem 3.17 and the result of [vH16], although the approximation ratio is slightly worse.

Problem	Graphs	Stream	Approx.	Space	Reference
vertex-cover	general	insertion-only	2	$O(n\log n)$	Folklore
		sliding-window	$8+\varepsilon$	$O_\varepsilon(n\log^2 n)$	[vH16]
		sliding-window	$4+\varepsilon$	$O_\varepsilon(n\log^2 n)$	Theorem 3.20
vertex-cover size (estimation)	VDP	turnstile	$\frac{5}{4}+\varepsilon$	$O_\varepsilon(\sqrt{n}\log^2 n)$	[vH16]
	VDP	sliding-window	$3.125+\varepsilon$	$O_\varepsilon(\sqrt{n}\log^4 n)$	Theorem 3.17
	forests	sliding-window	$4+\varepsilon$	$O_\varepsilon(\log^3 n)$	Theorem 3.18

Table 2: Estimation and approximating algorithms for vertex-cover in different settings (considering constant probability of success).

1.4.2 Maximum-Matching in the Vertex-Arrival Model

Following the approach of Crouch et al. [CMS13] for maximum-matching in the sliding-window model, we design a sliding-window algorithm for the more restrictive vertex-

[3]A graph $G = (V, E)$ is said to be VDP if G is a union of vertex-disjoint paths.

arrival model. Our algorithm achieves better approximation ratio (roughly 2.164 compared with 3), while the space complexity is the same, but it is only applicable for bipartite graphs in the vertex-arrival streaming model. A slightly informal version of Theorem 2.5 is stated here.

Theorem 1.13 (Informal version of Theorem 2.5)**.** *There exists a sliding-window algorithm for maximum-matching in vertex-arrival streams that achieves approximation ratio roughly* 2.164 *using space complexity* $O(n \log^2 n)$*.*

The greedy maximal-matching algorithm is a 2-approximation algorithm for maximum-matching. Crouch et al. [CMS13] used that algorithm for the smooth histogram technique, achieving an approximation ratio $2 \cdot 2 - 1 = 3$, with an involved analysis based on the the fact that the greedy matching is well structured. The algorithm of Kapralov [Kap13], for the vertex-arrival (non-sliding-window) model, has an approximation ratio $\rho = \frac{e}{e-1} \approx 1.582$. We use that algorithm of Kapralov [Kap13] as a black-box for the smooth histogram technique and using a direct analysis we obtain approximation ratio $2\rho - 1 \approx 2.164$.

In Table 3 we compare our approximation algorithm for the sliding-window vertex-arrival model (Theorem 2.5), applicable only for bipartite graphs, with previously known algorithms. The first one is a sliding-window algorithm for general graphs, with slightly worst approximation ratio. The second one is an approximation algorithm for the (non-sliding-window) insertion-only vertex-arrival model with a better approximation ration. All algorithms have the same space complexity, up to polylog (n) factors.

Problem	Graphs	Stream	Approx.	Space	Reference
maximum-matching	general	insertion-only	2	$O(n \log n)$	[FKM+05]
	general	sliding-window	$3 + \varepsilon$	$O_\varepsilon(n \log^2 n)$	[CMS13]
	bipartite	insertion-only vertex-arrival	1.582	$O(n \log n)$	[Kap13]
	bipartite	sliding-window vertex-arrival	$2.164 + \varepsilon$	$O_\varepsilon(n \log^2 n)$	Theorem 2.5

Table 3: Approximation algorithms for maximum-matching in different settings.

Using a reduction due to Crouch and Stubbs [CS14] we immediately obtain an approximation algorithm for the weighted problem (see Theorem 2.6) in the sliding-window model, while increasing the approximation ratio by at most 2, and increasing the space and update time by a factor of $O(\varepsilon^{-1} \log n)$.

1.4.3 Results for the Frequency-Vector Model

Lastly, we present algorithms for ℓ_p-sampling in the sliding-window model. We start by showing our technique in full details for $p = 0$, and then generalize it to every $p > 0$. Specifically, for $p = 0$ we prove the following theorem in Section 4.1, with a slightly better space bound.

Theorem 1.14 (Simpler version of Theorem 4.8)**.** *For every $\varepsilon, \delta \in (0,1)$ there is an algorithm for ℓ_0-sampling in the sliding-window model with space bound*

$$O(\tfrac{1}{\varepsilon^3} \log n \log^2 w \log \tfrac{1}{\delta\varepsilon}).$$

The algorithm's output $X \in [n]$ is within total variation distance $\delta + 3\varepsilon$ of a $(1+\varepsilon)$-uniform distribution over $\operatorname{supp}(W)$*.*

For both cases, $p = 0$ and $p > 0$, our approach is to show a general reduction from the sliding-window model to the insertion-only model. For $p > 0$ we then plug-in a known insertion-only algorithms, while for $p = 0$ we present an ℓ_0-sampling algorithm in the insertion-only model, based on a minwise-independent family of hash functions due to Indyk [Ind99], from which we deduce an ℓ_0-sampling algorithm in the sliding-window model.

1.5 Preliminaries

Throughout, we make use of a general observation regarding algorithms for the sliding-window model, due to Braverman, which says that without loss of generality, the entire stream can be assumed to have length at most twice the size of the window. We repeat it here for completeness.

Claim 1.15. Every sliding-window algorithm Λ can be modified such that it will not depend on the length of the entire stream, but only depend on at most $2w$ last items from the stream, while using at most a factor 2 more space.

Proof. To avoid dependence on the length of the stream N, and instead be dependent only on the length of the window w, we can argue as follows: partition the entire stream D to segments $D_1, D_2, \ldots, D_t$, of length w each, where $t = \left\lceil \frac{N}{w} \right\rceil$ (except maybe the last segment D_t, which is of length $0 < N - (t-1)\,w \leq w$). At each segment D_i start a new instance of algorithm Λ, and keep running it during the next segment as well, for at most $2w$ updates in total (for each instance of Λ). At any point in time, to answer a query the algorithm queries the instance of Λ on the penultimate segment, which corresponds to a suffix of the stream of length at least w, and thus contains the entire active window. Thus, at each point in time it is enough to store only the two instances of algorithm Λ corresponding to the last two segments, increasing the storage requirement only by a factor of 2. $\square$

2 Maximum-Matching in the Vertex-Arrival Model

We show a deterministic semi-streaming sliding-window algorithm for approximating maximum-matching in vertex-arrival streams, where all edges incident on a vertex arrive consecutively in the stream. The approach of Crouch et al. [CMS13] for approximating maximum-matching in the sliding-window model, which in turns is based on the smooth histogram framework of Braverman and Ostrovsky [BO07], employs the greedy algorithm who maintains maximal-matching in insertion-only streams. We adapt their approach to the vertex-arrival model, but instead of the greedy algorithm we employ the algorithm of Kapralov [Kap13]. By maintaining a near-optimal matching in each suffix of the stream, using the optimal algorithm of Kapralov [Kap13], we achieve better approximation then previously was known.

Previous work by Crouch et al. [CMS13] achieves $(3+\varepsilon)$-approximation for maximum-matching in the sliding-window model for ordinary insertion-only streams for general graphs, not only for bipartite graphs. Our result (Theorem 2.5) achieves roughly $(2.164+\varepsilon)$-approximation for maximum-matching in the sliding-window model, but only for the more restrictive vertex-arrival model, and hence only for bipartite graphs.

Crouch et al. [CMS13] also presented an algorithm for the maximum-weight matching (in the usual model of sliding-window) that achieves roughly 9.027-approximation. Later, Crouch and Stubbs [CS14] presented a general reduction from maximum-weighted matching to maximum-matching by using $O(\varepsilon^{-1}\log n)$ instances of an algorithm for maximum-matching and increasing the approximation factor by $2+\varepsilon$. Their reduction results in a $(6+\varepsilon)$-approximation sliding-window algorithm for the maximum-weight matching for general graphs in the insertion-only model. Our algorithm for maximum-matching implies, by their general reduction, a $(4.328+\varepsilon)$-approximation sliding-window algorithm for the maximum-weight matching in the vertex-arrival model (for bipartite graphs).

Recall that in the vertex-arrival model, a bipartite graph arrives in a streaming fashion, where all edges incident on a vertex from the left side arrive consecutively in the stream, i.e., for a bipartite graph $G=(V,U,E)$ a stream in the vertex-arrival model consists of some permutation of the vertices on the left side $V=\{v_1,v_2,\cdots,v_n\}$ and the stream is $\mathcal{S}=\langle E(v_1),E(v_2),\cdots,E(v_n)\rangle$, where for a vertex $v\in V$ its set of edges denoted by $E(v)=\{(v,u)\,|u \text{ is a neighbor of } v\}$. We assume that for every vertex $v\in V$ the set of edges $E(v)$ arrives in some arbitrary order. For any segment $X_{i,j}=(E(v_i),\cdots,E(v_j))$, for $1\le i\le j\le n$, of the stream $\mathcal{S}$ define $E(X_{i,j})=\bigcup\limits_{k=i}^{j}E(v_k)$. We assume that the actual stream consists of edges, in the aforementioned order, and the length of the segment $X_{i,j}$ is the total number of edges in it, i.e., the length of $X_{i,j}$ is $\sum\limits_{k=i}^{j}|E(v_k)|$.

Kapralov [Kap13] introduced a deterministic algorithm in the vertex-arrival model that achieves an approximation ratio of $\rho=\frac{e}{e-1}\approx 1.582$, i.e., for every stream $\mathcal{S}$ of edges in the vertex-arrival model it holds that $\frac{1}{\rho}\cdot m(\mathcal{S})\le m'(\mathcal{S})\le m(\mathcal{S})$, where $m(\mathcal{S})$

is the maximum-matching size of the underlying graph defined by the stream S, and $m'(S)$ is the size of the matching produced by the algorithm of Kapralov on the same graph. Theorem 2 from [Kap13] adapted to 1-pass states the following.

Theorem 2.1. *[Kap13, Theorem 2 for k=1 passes] There exists an algorithm for approximating maximum-matching in a bipartite graph $G = (V, U, E)$ given by a stream in the vertex-arrival model within factor $\rho = \frac{e}{e-1}$ with space bound $O(n \log n)$, where $n = \max\{|V|, |U|\}$.*

For disjoint segments A, B, C of a stream we denote by AB the concatenation of A and B, and we define $\mu(A, B, C) = m(AB) + m(BC) - m(B)$. For the purpose of analyzing our algorithm in Theorem 2.5 we will need the following lemma bounding the size of $\mu(A, B, C)$.

Lemma 2.2. *For every disjoint segments of a stream in the vertex-arrival model A, B and C, it holds that $m(ABC) \leq \mu(A, B, C) \leq 2m(ABC)$. Moreover, both inequalities are tight.*

Proof. First, for the tightness, observe that for every stream ABC that define a perfect matching there is an equality in the first inequality. Second, for $A = ((a_i, \{x_i\}))_{i=1}^n$, $B = \emptyset$ and $C = ((c_i, \{x_i\}))_{i=1}^n$ it holds that $m(ABC) = m(AB) = m(BC) = n$ and obviously $m(B) = 0$. Hence $\mu(A, B, C) = m(AB) + m(BC) - m(B) = n + n - 0 = 2n$. Thus, there is an equality in the second inequality as well.

Moving to proving the inequalities, we can easily deduce the second inequality as follows,

$$\mu(A, B, C) = m(AB) + m(BC) - m(B) \leq m(AB) + m(BC) \leq 2m(ABC).$$

It remains to prove the first inequality. Let $M(X)$ denote the set of edges of some maximum-matching on a segment of a stream X in the vertex-arrival model, and let $V(M(X)) \subseteq V$ be the vertices from V participating in the matching $M(X)$. We will need the following claim.

Claim 2.3. Fix a maximum-matching $M(B)$. There exist some maximum-matching $M(ABC)$ such that $V(M(B)) \subseteq V(M(ABC))$.

Proof. Let M^* be a maximum-matching in ABC such that the size of the difference $M(B) \setminus M^*$ is minimized over all maximum-matchings in ABC. Let $E_{M^*} = M(B) \setminus M^*$ and $V_{M^*} = V(M(B)) \setminus V(M^*)$. Assume towards contradiction that $V_{M^*} \neq \emptyset$, and let $v \in V_{M^*}$. Denote by $e = (v, u) \in M(B)$ the matched edge corresponding to v from the matching $M(B)$. Since $v \notin V(M^*)$, there is some other edge $e' = (v', u) \in M^*$, for some vertex $v' \neq v$, and so we can exchange them to obtain the matching $M' = \{e\} \cup M^* \setminus \{e'\}$, which is also a maximum-matching on ABC, because $|M^*| = |M'|$. Observe that $|M(B) \setminus M'| = |E_{M^*}| - 1$ because $e' \notin M(B)$ and we removed from E_{M^*} the edge $e \in M(B)$. But this contradicts the way we chose the matching M^*, as the matching that minimized over all maximum-matchings on ABC the difference $M(B) \setminus M^*$. Therefore, $V_{M^*} = \emptyset$ as required. □

Let $V_1 = V(M(ABC)) \cap V(AB)$ and $V_2 = V(M(ABC)) \cap V(BC)$. By Claim 2.3 we can assume without loss of generality that $V(M(B))$ is contained in each of $V(M(ABC)), V(M(AB))$ and $V(M(BC))$. Therefore, each vertex $v \in V(M(B))$ is both in V_1 and in V_2. Moreover, each vertex $v \in V(M(ABC)) \backslash V(M(B))$ is in only one of V_1 or V_2, because v is a matched vertex not in $V(M(B))$ and so it is not a vertex from B (because otherwise $M(B)$ can be augmented). Hence, we deduce that

$$|V(M(ABC))| = |V_1| + |V_2| - |V(M(B))|.$$

Therefore, as $m(X) = |V(M(X))|$ for every segment X of a stream, we can deduce the required inequality:

$$|V(M(ABC))| \le m(AB) + m(BC) - m(B) = \mu(A, B, C).$$

□

Corollary 2.4. *Let $\varepsilon \in (0,1)$ and let A, B, C be as in Lemma 2.2. If $m'(B) \ge (1-\varepsilon) m'(AB)$ then $m(ABC) \le \left(\frac{\rho}{1-\varepsilon} - 1 + \rho\right) m'(BC)$, where $\rho = \frac{e}{e-1}$.*

Proof. The matching produced by the algorithm of Kapralov [Kap13] satisfies $m(AB) \le \rho m'(AB)$, $m(BC) \le \rho m'(BC)$ and $m'(BC) \le m(BC)$, hence using Lemma 2.2,

$$m(ABC) \le m(AB) + m(BC) - m(B) \le \rho m'(AB) + \rho m'(BC) - m'(B).$$

Note that the algorithm of Kapralov [Kap13] produces monotone matchings, i.e., when appending edges to the end of the stream the algorithm will not produce a smaller matching.

If $m'(B) \ge (1-\varepsilon) m'(AB)$ then we can deduce the required inequality:

$$\begin{aligned} m(ABC) &\le \frac{\rho}{1-\varepsilon} m'(B) + \rho m'(BC) - m'(B) \\ &= \left(\frac{\rho}{1-\varepsilon} - 1\right) m'(B) + \rho m'(BC) \\ &\le \left(\frac{\rho}{1-\varepsilon} - 1\right) m'(BC) + \rho m'(BC) \\ &= \left(\frac{\rho}{1-\varepsilon} - 1 + \rho\right) m'(BC). \end{aligned}$$

□

Theorem 2.5. *There exists a sliding-window algorithm for maximum-matching in vertex-arrival streams that achieves approximation ratio $2\rho - 1 + \varepsilon \approx 2.164 + \varepsilon$, for any desired $\varepsilon \in \left(0, \frac{1}{2}\right)$, using space complexity $O(\varepsilon^{-1} n \log^2 n)$.*

The theorem is proved by adapting the approach of Crouch et al. [CMS13], which in turns is based on the smooth histogram method of Braverman and Ostrovsky [BO07]. Crouch et al. [CMS13] employ the greedy algorithm, which is applicable for the insertion-only model (and has approximation ratio 2), while our algorithm employs the algorithm of Kapralov [Kap13], which is only suitable for the vertex-arrival model, but achieves better approximation ratio ($\rho = \frac{e}{e-1} \approx 1.582$ compared to 2). We use the algorithm of Kapralov to maintain a near-optimal matching in each "bucket", where a bucket is a suffix of the stream, i.e., a sequence of edges corresponding to some vertices together with all their neighbors.

Proof. Let $\mathcal{A}$ be the algorithm of Kapralov [Kap13], and denote by $\mathcal{A}(X)$ the matching produced by it when run on the stream X. Our algorithm maintains (not explicitly) $k = O(\varepsilon^{-1} \log n)$ "buckets" $B_1, \dots, B_k$, and for each bucket it maintains a matching according to the algorithm of Kapralov. At all points in time, these buckets will satisfy the invariant $B_1 \supseteq W \supsetneq B_2 \supsetneq B_3 \supsetneq \cdots \supsetneq B_k$, where W is the active window. Hence,

$$m(B_1) \geq m(W) \geq m(B_2) \geq \cdots \geq m(B_k).$$

For each bucket B_i the algorithm maintains a matching, denoted by $M'(B_i)$, and whose size we denote by $m'(B_i)$. In order to use only a small amount of space, whenever two nonadjacent buckets have matchings of similar size, we will delete all buckets between them. For ease of exposition, the algorithm will be defined using these buckets, and later we explain how to not actually store the buckets themselves. In each step of receiving a new edge from the stream, the algorithm updates the buckets and the corresponding matchings in the following way.

Algorithm 1 Update and query procedure

```
Update Procedure: Given the next item $e = (v, u)$, update the current buckets
    $B_1, \dots, B_k$ and the corresponding matchings $M'(B_1), \dots, M'(B_k)$, as follows:
    1. Initialize new bucket $B_{k+1} = \{e\}$, new matching $M'(B_{k+1}) = \mathcal{A}(B_{k+1})$
       according to algorithm $\mathcal{A}$ on the bucket $B_{k+1}$.
    2. For each $1 \leq i \leq k$, add the edge $e$ to the bucket $B_i$ and update the matching
       $M'(B_i) = \mathcal{A}(B_i)$ according to algorithm $\mathcal{A}$.
    3. For $i = 1, \dots, k-2$ do:
       (a) Find the largest $j > i$ such that $m'(B_j) > \left(1 - \frac{\varepsilon}{3}\right) m'(B_i)$.
       (b) For every $i < t < j$ delete the bucket $B_t$, and renumber the buckets
           accordingly.
    4. If $B_2$ contains the active window $W$, delete $B_1$, and renumber the buckets.

Query Procedure: If the first bucket $B_1$ is exactly the active window $W$ then output
    $M'(B_1)$, otherwise output $M'(B_2)$.
```

First, let us explain how to avoid storing the buckets $B_1, \ldots, B_k$ and consequently achieve $O\left(\varepsilon^{-1} n \log^2 n\right)$ space complexity: Instead of adding edges to the buckets and then calculating approximate matching on that bucket, we only need to maintain the storage that algorithm $\mathcal{A}$ requires. Additionally, we need to store a counter $c\left(B_i\right)$ for every bucket, indicating its initialization time (initialized to $c\left(B_i\right) = 1$ and incremented each time a new edge arrives), such that we can perform the last step of the algorithm, by comparing the counter for the bucket B_2 to the size of the active window w. This way, we store for each bucket B_i only $O(n \log n)$ bits of space, according to Theorem 2.1. Since the maximum-matching has size at most n, and because $m'\left(B_{i+2}\right) < \left(1 - \frac{\varepsilon}{3}\right) m'\left(B_i\right)$ for every $1 \leq i \leq k-2$ (as the "unnecessary" buckets were deleted in the process of updating), it follows that the number of buckets bounded by $O\left(\varepsilon^{-1} \log n\right)$. Therefore, the total number of bits used by the algorithm is $O\left(\varepsilon^{-1} n \log^2 n\right)$, as claimed.

For the approximation ratio observe that for every $i < k$ we have either $|B_i| = |B_{i+1}| + 1$ or $m'\left(B_{i+1}\right) > \frac{m(B_i)}{2\rho - 1 + \varepsilon}$ (or both). Assume $|B_i| \neq |B_{i+1}| + 1$, which means that the algorithm has deleted some bucket D between B_i and B_{i+1}. Thus, at the time of the deletion we had $m'\left(B_{i+1}\right) > \left(1 - \frac{\varepsilon}{3}\right) m'\left(B_i\right)$. Let C be the suffix of the stream starting at the time D was deleted, let B equal B_{i+1} at the time of the deletion, and let A be the segment of the stream from B_i that arrived before B_{i+1} was initialized, i.e., $B_i = AB_{i+1}$. Note that at the end of the stream $B_{i+1} = BC$ and $B_i = ABC$, hence using Corollary 2.4 with $\frac{\varepsilon}{3}$ we obtain

$$m\left(B_i\right) \leq \left(\frac{\rho}{1 - \frac{\varepsilon}{3}} - 1 + \rho\right) m'\left(B_{i+1}\right) \leq (2\rho - 1 + \varepsilon)\, m'\left(B_{i+1}\right).$$

Therefore, either $B_1 = W$ and then $m'\left(B_1\right)$ is a ρ-approximation for $m\left(W\right)$, or

$$m\left(W\right) \geq m\left(B_2\right) \geq m'\left(B_2\right) \geq \frac{m\left(B_1\right)}{2\rho - 1 + \varepsilon} \geq \frac{m\left(W\right)}{2\rho - 1 + \varepsilon}$$

which means that $m'\left(B_2\right)$ is $(2\rho - 1 + \varepsilon)$-approximation for $m\left(W\right)$, proving that the above sliding-window algorithm is indeed $(2\rho - 1 + \varepsilon)$-approximates the maximum-matching problem in the bipartite vertex-arrival model. $\square$

Crouch and Stubbs [CS14] presented a general reduction from maximum-weighted matching to maximum-matching. Their reduction increases the space complexity by $O(\varepsilon^{-1} \log n)$ and the approximation ratio by $2 + \varepsilon$. Their idea is to partition the weights to $O(\varepsilon^{-1} \log n)$ classes, run an algorithm for maximum-matching on each class independently, and at the end of the stream combine the matchings in a clever way. Thus, their reduction applicable in the sliding-window model, and hence, using it with Theorem 2.5 we deduce the following theorem.

Theorem 2.6. *There exist a sliding-window algorithm for maximum-weighted matching in the vertex-arrival model that achieves approximation ratio* $2\left(2\rho - 1 + \varepsilon\right) \approx 4.328 + \varepsilon$, *for any desired* $\varepsilon \in \left(0, \frac{1}{2}\right)$, *using space complexity* $O(\varepsilon^{-2} n \log^3 n)$.

3 Almost-Smooth Functions and Application to Graphs

We generalize the smooth-histogram approach of Braverman and Ostrovsky [BO07] to functions that are almost smooth, as per our new definition. We show that the size of a maximum-matching is almost smooth. For graphs of bounded arboricity α, we then use the algorithm of McGregor and Vorotnikova [MV18] for estimating the size of a maximum-matching, and deduce a sliding-window algorithm with approximation factor $O(\alpha^2)$ and space $\operatorname{poly}\log n$. We then improve the approximation ratio to $O(\alpha)$ by observing that the number of α-good edges (the quantity used to approximate the maximum-matching) is itself an almost smooth function, and thus we can argue directly about it. See Table 1.

Next, we show a few results for minimum vertex-cover (again in the sliding-window model), based on its relationship to maximum and maximal matching, and the fact that it is too almost smooth. We show an algorithm with approximation factor $3.125+\varepsilon$ for the size of a minimum vertex-cover in VDP graphs using $\widetilde{O}(\sqrt{n})$ space. We continue and present another algorithm for a larger family of graphs, namely, forest graphs, where the approximation factor grows to $4+\varepsilon$ but the space complexity reduces to $\operatorname{poly}\log n$. We then proceed to show how to report a feasible vertex cover. We reproduce a known algorithm for general graphs with approximation factor $8+\varepsilon$ that computes a vertex cover using $\widetilde{O}(n)$ space. Then we show how to improve the approximation factor to $4+\varepsilon$ by a tighter analysis of that same algorithm, using that the size of a greedy maximal matching is also almost smooth.

Recall that in the usual graph streaming model, the input is a stream of edge insertions to an underlying graph on the set of vertices $V = [n]$, where n is known in advance. The goal is usually to compute some function of the underlying graph, e.g. its maximum-matching size. We assume that the underlying graph does not contain parallel edges, i.e., the stream of edges does not contain the same edge twice. Hence, the length of the entire stream is bounded by n^2.

In the sliding-window model the graph is defined using only the last w edge insertions from the stream, referred to as the active window W. Note that w is known (to the algorithm) in advance, and that $w \leq n^2$, as the length of the entire stream is bounded by n^2.

Throughout, update time refers to the time complexity of processing a single update from the stream. The space bound refers to the space complexity, measured in bits, needed for the algorithm during the entire stream.

3.1 Almost-Smooth Functions

Recall that for disjoint segments A, B of a stream, we denote by AB their concatenation. We use the parameter n to denote some measure of a stream which will be clear from the context. For example, for graph streams n is the number of vertices in the underlying graph. We extend the definition of smoothness due to [BO07] as follows.

Definition 3.1. (Almost Smooth Function) A real-valued function f defined on streams is (c,d)-*almost-smooth*, for $c,d \geq 1$, if it has the following properties:

1. Non-negative: for every stream A it holds that $f(A) \geq 0$.
2. c-monotone: for every disjoint segments A, B of a stream it holds that $f(B) \leq c \cdot f(AB)$.
3. Bounded: for every stream A it holds that $f(A) \leq \text{poly}(n)$.
4. Almost smooth: for every disjoint segments A, B, C of the stream,
$$\frac{f(B)}{f(AB)} \leq d \cdot \frac{f(BC)}{f(ABC)}$$
whenever $f(AB) \neq 0$ and $f(ABC) \neq 0$.

Remark 3.2. Almost-smoothness means that appending any segment C at the end of the stream preserves the approximation of $f(B)$ by $f(AB)$, up to a multiplicative factor d. Observe that this is equivalent to the following condition. For every $\varepsilon > 0$ and every disjoint segments of the stream A, B, C,
$$\varepsilon \cdot f(AB) \leq f(B) \qquad \Longrightarrow \qquad \varepsilon \cdot f(ABC) \leq d \cdot f(BC).$$

Remark 3.3. Throughout, it is more convenient to use this equivalent condition.

For generality we defined (c,d)-almost-smooth for any $c \geq 1$, but in our applications $c = 1$, in which case we simply omit c and refer to such functions as d-almost-smooth.

Remark 3.4. In the original definition of smoothness from [BO07], not only $c = d = 1$, but also property 4 is stated as follows. A function f is $(\varepsilon, \beta(\varepsilon))$-smooth if for every $\varepsilon \in \left(0, \frac{1}{2}\right)$ there exist $\beta(\varepsilon) \leq \varepsilon$ and
$$(1 - \beta(\varepsilon)) \cdot f(AB) \leq f(B) \qquad \Longrightarrow \qquad (1 - \varepsilon) \cdot f(ABC) \leq f(BC).$$

Let $m(S)$ be the size of the maximum-matching in the graph defined by the stream S. Although $m(\cdot)$ is not a smooth function (as shown in Remark 3.6), it is almost smooth, as proved by Crouch et al. [CMS13] and reproduced here for completeness.

Lemma 3.5. *[CMS13] The maximum-matching size $m(\cdot)$ is 2-almost-smooth.*

Proof. The first three requirements are clear, as $m(\cdot)$ is non-negative, bounded and monotone, since on a sub-segment of the stream the maximum-matching can not be larger. Hence, we are only left to show the almost-smoothness property. Let $\varepsilon \in (0,1)$

and let A, B and C be disjoint segments of the stream satisfying $\varepsilon m(AB) \leq m(B)$. Observe that $m(AB)+m(BC) \geq m(A)+m(BC) \geq m(ABC)$, because m is monotone, and therefore,

$$\begin{aligned} 2m(BC) \geq & m(B) + m(BC) \geq \varepsilon m(AB) + m(BC) \\ \geq & \varepsilon (m(AB) + m(BC)) \geq \varepsilon m(ABC). \end{aligned}$$

□

Remark 3.6. The almost-smoothness $d = 2$ in Lemma 3.5 is tight. Let $G = (V, E)$ be a graph composed of n vertex-disjoint paths of length 3, i.e., n paths of the form $e_a = \{x, y\}, e_b = \{y, z\}, e_c = \{z, w\}$. The segment A of the stream contains all the e_a edges, B contains all the e_b edges, and C contains all the e_c edges. Obviously $m(AB) = m(B) = m(BC) = n$ while $m(ABC) = 2n$. In particular, the maximum-matching size is not smooth as per the original definition of [BO07].

Recall that for $\varepsilon, \delta \in (0,1)$ and $C \geq 1$, an algorithm Λ is said to $((1+\varepsilon)C, \delta)$-approximate a function f if on every input stream D, its output X satisfies

$$\Pr\left[(1-\varepsilon) f(D) \leq X \leq (1+\varepsilon) C \cdot f(D)\right] \geq 1 - \delta.$$

If $\delta = 0$ we say for short that the algorithm $(1+\varepsilon)C$-approximates f.

We use the Smooth Histogram algorithm of [BO07] with a modified analysis, but we use a more convenient description, similarly to Crouch et al. [CMS13] and to our algorithm for the ℓ_0-sampling problem.

Theorem 3.7. *[Restatement of Theorem 1.11] Let f be a (c, d)-almost-smooth function. Assume that for every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there exists a streaming algorithm Λ that $((1+\varepsilon)C, \delta)$-approximates f using space $s(\varepsilon, \delta)$ and update time $t(\varepsilon, \delta)$. Then there exists a sliding-window algorithm Λ^{sw} that $(dc^2C^2(1+O(\varepsilon)), \delta)$-approximates f using space $O(\varepsilon^{-1}\log w \cdot s\left(\varepsilon, \frac{\varepsilon\delta}{2w\log w}\right))$ and update time $O(\varepsilon^{-1}\log w \cdot t\left(\varepsilon, \frac{\varepsilon\delta}{2w\log w}\right))$.*

Recall that without loss of generality we can assume that the length of the entire stream is at most $2w$, as explained in Claim 1.15.

Proof. We use the same technique and notation as in algorithm 1, namely, maintain $k = O(\varepsilon^{-1}\log w)$ instances of algorithm Λ, all with the same parameters ε for accuracy and $\frac{\varepsilon\delta}{2w\log w}$ for failure probability. Each instance Λ_i is defined on a suffix B_i of the stream, called a "bucket", where those buckets satisfy the invariant $B_1 \supseteq W \supsetneq B_2 \supsetneq B_3 \supsetneq \cdots \supsetneq B_k$, and W is the active window. Upon receiving a new update from the stream, the algorithm updates the instances similarly to algorithm 1. Specifically, for each $i \in [k-2]$, find the largest $j > i$ such that $\Lambda_j(B_j) > (1-\varepsilon)\Lambda_i(B_i)$, delete all buckets between them and renumber the buckets accordingly. The output of the algorithm is either $\tilde{\Lambda} = \Lambda_1(B_1)$ in the case $B_1 = W$, or $\tilde{\Lambda} = dcC\frac{(1+\varepsilon)}{(1-\varepsilon)^2} \cdot \Lambda_2(B_2)$ otherwise.

Let $\Lambda_1(B_1)$ and $\Lambda_2(B_2)$ be the output of algorithm Λ on the first and second buckets at the end of the stream, respectively. Note that for a fixed instance at a fixed point in time, with probability at most $\frac{\varepsilon\delta}{2w\log w}$ it makes an error. Hence, by union bound, with probability at least $1-\delta$ every instance in the entire run of the algorithm correctly approximates f on the corresponding bucket, every time it is queried. Thus, from now on we assume that every instance of Λ succeeds every time, i.e., it $(1+\varepsilon)C$-approximates f on the corresponding bucket whenever it is invoked.

If $B_1 = W$ then the output $\tilde{\Lambda} = \Lambda(B_1)$ is obviously a $(1+\varepsilon)C$-approximation of f on $B_1 = W$. Otherwise $B_1 \supsetneq W \supsetneq B_2$, which means that at some earlier point in time, denoted by t^*, the algorithm had deleted some buckets between them to make them adjacent (for the first time). Note that at time t^* the buckets B_1 and B_2 first became adjacent. For $i \in \{1,2\}$ denote by B_i' the bucket B_i at the time t^*. Let D be the suffix of the stream starting at time t^*, and observe that $B_1 = B_1'D$ and $B_2 = B_2'D$. At time t^* we had $(1-\varepsilon)\Lambda(B_1') < \Lambda(B_2')$, which implies

$$(1-\varepsilon) f(B_1') \le (1-\varepsilon)\Lambda(B_1') < \Lambda(B_2') \le (1+\varepsilon) C \cdot f(B_2'),$$

namely $\frac{(1-\varepsilon)}{(1+\varepsilon)C} f(B_1') \le f(B_2')$. Since f is (c,d)-almost-smooth, at the end of the stream we have $\frac{1}{d}\cdot\frac{(1-\varepsilon)}{(1+\varepsilon)C} f(B_1) \le f(B_2)$. Now, by monotonicity $\frac{1}{c}\cdot f(B_2) \le f(W) \le c\cdot f(B_1)$, and altogether

$$\frac{1}{cC(1+\varepsilon)}\Lambda(B_2) \le \frac{1}{c}\cdot f(B_2) \le f(W) \le c\cdot f(B_1) \le cdC\cdot\frac{(1+\varepsilon)}{(1-\varepsilon)} f(B_2) \le cdC\cdot\frac{(1+\varepsilon)}{(1-\varepsilon)^2}\Lambda(B_2).$$

Since $\frac{(1+\varepsilon)^2}{(1-\varepsilon)^2} \le 1+20\varepsilon$ for $\varepsilon \le \frac{1}{2}$, we conclude that at the end of the stream the output of the algorithm $\tilde{\Lambda} = dcC\frac{(1+\varepsilon)}{(1-\varepsilon)^2}\cdot\Lambda(B_2)$ approximate $f(W)$ as claimed. □

For certain approximation algorithms we can reduce the dependence on the approximation factor C from quadratic (C^2) to linear (C). Suppose that the approximation algorithm Λ of the function f has the following form: It $(1+\varepsilon,\delta)$-approximates a function g, and this g is a C-approximation of f. Now, if g itself is (c,d)-almost-smooth then we can save a factor of C by arguing directly about approximating g.

Theorem 3.8. *Let f be some function, let g be a (c,d)-almost-smooth function, and assume that g is a C-approximation of f. Assume that for every $\varepsilon,\delta \in \left(0,\frac{1}{2}\right)$, there exists a streaming algorithm Λ that $(1+\varepsilon,\delta)$-approximates g using space $s(\varepsilon,\delta)$ and update time $t(\varepsilon,\delta)$. Then there exists a sliding-window algorithm Λ^{sw} that $(dc^2C(1+O(\varepsilon)),\delta)$-approximates f using space $O(\varepsilon^{-1}\log w \cdot s\left(\varepsilon,\frac{\varepsilon\delta}{2w\log w}\right))$ and update time $O(\varepsilon^{-1}\log w \cdot t\left(\varepsilon,\frac{\varepsilon\delta}{2w\log w}\right))$.*

Proof. By applying Theorem 3.7 to the function g and the algorithm Λ, that approximates it, we obtain a sliding-window algorithm Λ^{sw} that computes a $(dc^2(1+O(\varepsilon)),\delta)$-approximation of g, uses space $O(\varepsilon^{-1}\log w\cdot s\left(\varepsilon,\frac{\varepsilon\delta}{2w\log w}\right))$ and update time $O(\varepsilon^{-1}\log w\cdot t\left(\varepsilon,\frac{\varepsilon\delta}{2w\log w}\right))$. Since g is a C-approximation of f, this algorithm Λ^{sw} is in fact a $(dc^2C(1+O(\varepsilon)),\delta)$-approximation of f, using the same space and update time. □

Application to Negative Frequency Moments Recently, Braverman and Chestnut [BC15] showed an insertion-only streaming algorithm that $(1+\varepsilon)$-approximates negative frequency moments, F_p for $p<0$, with space complexity $O(\varepsilon^{-\frac{2-p}{1-p}} m^{\frac{-p}{1-p}} \log M)$, where $m = \sum\limits_{i=1}^{n} x_i$ and $M = \max\{n, x_i | i \in [n]\}$, where $x = (x_1, \ldots, x_n)$ is the underlying frequency-vector of the stream. It is easy to show that the F_p moment is $(\varepsilon, \varepsilon)$-smooth for every $p<0$ and that the smooth histogram technique (with the right modifications) is applicable also to monotonically decreasing functions. Therefore, for every $p<0$ there is a sliding-window algorithm for $(1+\varepsilon)$-approximation of a negative frequency moment F_p, with space complexity $O(\varepsilon^{-\frac{3-2p}{1-p}} m^{\frac{-p}{1-p}} \log M \log w)$.

3.2 Maximum-Matching

A graph has arboricity α if its set of edges can be partitioned into at most α forests. For example, it is well known that every planar graph has arboricity $\alpha = 3$, see e.g. [GL98]. McGregor and Vorotnikova [MV18], based on the result of Cormode et al. [CJMM17], presented an algorithm that approximates the size of the maximum-matching in a graph with arboricity α within factor $(1+\varepsilon)(\alpha+2)$, with constant probability, using space $O(\varepsilon^{-2}\log n)$ and update time $O(\varepsilon^{-2}\log n)$. To achieve low failure probability δ it is standard to compute a median of $\log \delta^{-1}$ parallel repetitions. Therefore, using the Almost-Smooth-Histogram method explained above we obtain the following theorem.

Theorem 3.9. *For every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there is a sliding-window $\left((2+\varepsilon)(\alpha+2)^2, \delta\right)$-estimation algorithm for maximum-matching size in a graph with arboricity α, with space bound $O(\varepsilon^{-3}\log^3 n \log \frac{1}{\varepsilon\delta})$ and update time $O(\varepsilon^{-3}\log^3 n \log \frac{1}{\varepsilon\delta})$.*

Proof. For $\varepsilon, \delta \in (0,1)$ let Λ_{MV} be the algorithm of McGregor and Vorotnikova [MV18], amplified to have success probability $1-\delta$, providing $((1+\varepsilon)(\alpha+2), \delta)$-approximation for maximum-matching size in graphs with arboricity α. As shown in Lemma 3.5, $m(\cdot)$ is 2-almost-smooth. Therefore, using Theorem 3.7 with $c=1, d=2, C=\alpha+2$ and algorithm Λ_{MV}, we obtain a sliding window algorithm Λ which $\left((2+\varepsilon)(\alpha+2)^2, \delta\right)$-approximate the maximum-matching size in graphs with arboricity α.

The space complexity of Λ_{MV} is $s_{MV}(\varepsilon, \delta) = O(\varepsilon^{-2}\log n \log \delta^{-1})$ and it the update time is $t_{MV}(\varepsilon, \delta) = O(\varepsilon^{-2}\log n \log \delta^{-1})$. Hence the space complexity of Λ is

$$O(\varepsilon^{-1}\log w \cdot s_{MV}\left(\varepsilon, \frac{\varepsilon\delta}{2w\log w}\right)) = O(\varepsilon^{-3}\log^3 n \log \tfrac{1}{\varepsilon\delta}),$$

and similarly for the update time, where we used the fact that $w \leq n^2$. □

For the purpose of approximating the maximum-matching size in graphs with arboricity bounded by α Cormode et al. [CJMM17] introduced the notion of α-good edges. The algorithm of [MV18] used in the above proof is actually approximates the maximum number of α-good edges in prefixes of the stream. Thus, using the same algorithm of [MV18], we can directly approximate the maximum size of the set of α-good edges in the active window W. For completeness we present here the definition of Cormode et al. [CJMM17] for α-good edges in a stream, and the notion of E^*_α due to McGregor and Vorotnikova [MV18].

Definition 3.10. Let $\mathcal{S} = (e_1, e_2, \ldots, e_k)$ be a sequence of k edges on the set of vertices $V = [n]$. We say that an edge $e_i = \{u, v\}$ is α-good (with respect to the stream $\mathcal{S}$) if $d_i(u) \leq \alpha$ and $d_i(v) \leq \alpha$, where $d_i(x)$ is the number of edges incident on the vertex x that appear after edge e_i in the stream, i.e., $d_i(x) = |\{e_j | j > i \wedge x \in e_j\}|$. Denote by $E_\alpha(\mathcal{S})$ the set of α-good edges in the stream S, and let $E^*_\alpha(\mathcal{S}) = \max_{t \in [k]} |E_\alpha(\mathcal{S}_t)|$, where $\mathcal{S}_t = (e_1, e_2, \ldots, e_t)$ is the prefix of S of length t.

Although the size of the set of α-good edges in a stream is not smooth or even almost-smooth, the function $E^*_\alpha(\cdot)$ is almost-smooth.

Lemma 3.11. *The function $E^*_\alpha(\cdot)$ is 2-almost-smooth.*

Proof. The first two requirements are clear, as E^*_α is non-negative and bounded. Let A, B and C be disjoint segments of the stream $S = (e_1, e_2, \ldots, e_k)$. Note that $E^*_\alpha(B) \leq E^*_\alpha(AB)$, since earlier edges do not interfere with later edges being α-good, hence E^*_α is also monotone. Furthermore, $E^*_\alpha(B) \leq E^*_\alpha(BC)$ as taking a maximum on a larger set can not be smaller. Hence, we are only left to show the almost-smoothness property.

If $E^*_\alpha(ABC) = E^*_\alpha(AB)$ then obviously $E^*_\alpha(AB) + E^*_\alpha(BC) \geq E^*_\alpha(ABC)$, as E^*_α is positive. Otherwise, let $1 \leq t \leq k$ be such that $e_t \in C$ and $E^*_\alpha(ABC) = |E_\alpha((ABC)_t)|$, then

$$E_\alpha((ABC)_t) = (E_\alpha((ABC)_t) \cap A) \cup E_\alpha((BC)_t),$$

where it is a disjoint union. Note that $E_\alpha((ABC)_t) \cap A \subseteq E_\alpha(AB)$, as every α-good edge from A with respect to the stream $(ABC)_t$ is also α-good edge in the stream AB. Hence,

$$E^*_\alpha(ABC) \leq |E_\alpha(AB)| + |E_\alpha((BC)_t)| \leq E^*_\alpha(AB) + E^*_\alpha(BC).$$

Therefore, if we assume that $\varepsilon E^*_\alpha(AB) \leq E^*_\alpha(B)$ then we obtain the required inequality

$$\begin{aligned} 2E^*_\alpha(BC) \geq & E^*_\alpha(B) + E^*_\alpha(BC) \geq \varepsilon E^*_\alpha(AB) + E^*_\alpha(BC) \\ \geq & \varepsilon\left(E^*_\alpha(AB) + E^*_\alpha(BC)\right) \geq \varepsilon E^*_\alpha(ABC). \end{aligned}$$

□

McGregor and Vorotnikova [MV18] proved that $m(\mathcal{S}) \leq |E_\alpha(\mathcal{S})| \leq (\alpha + 2) \cdot m(\mathcal{S})$ for every stream $\mathcal{S}$, and thus also $m(\mathcal{S}) \leq E^*_\alpha(\mathcal{S}) \leq (\alpha + 2) \cdot m(\mathcal{S})$. They also designed a $(1 + \varepsilon, \delta)$-approximation algorithm for $E^*_\alpha(\cdot)$. Since $E^*_\alpha(\cdot)$ is 2-almost-smooth by Lemma 3.11 we can apply Theorem 3.8, with $g = E^*_\alpha(\cdot)$ and $f = m(\cdot)$, to obtain the following improvement over Theorem 3.9.

Theorem 3.12 (Restatement of Theorem 1.12)**.** *For every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there is a sliding-window $\left(\left(2+\varepsilon\right)\left(\alpha+2\right), \delta\right)$-estimation algorithm for the maximum-matching size in a graph with arboricity α, with space bound $O(\varepsilon^{-3}\log^3 n \log\frac{1}{\varepsilon\delta})$ and update time $O(\varepsilon^{-3}\log^3 n \log\frac{1}{\varepsilon\delta})$.*

Remark 3.13. For arboricity $\alpha = 1$ we can achieve a better approximation. Cormode et al. [CJMM17] showed that in this case $m\left(\mathcal{S}\right) \leq \left|E_1\left(\mathcal{S}\right)\right| \leq 2 \cdot m\left(\mathcal{S}\right)$ and thus $m\left(\mathcal{S}\right) \leq E_1^*\left(\mathcal{S}\right) \leq 2 \cdot m\left(\mathcal{S}\right)$. Therefore, by Theorem 3.8 there is a $(4+\varepsilon, \delta)$-approximation algorithm for the maximum-matching size in forest graphs in the sliding-window model with the same space and update time bounds.

3.3 Minimum Vertex-Cover

For a graph $G = (V, E)$ a subset $C \subseteq V$ of the vertices is called a *vertex cover* of G if each edge $e \in E$ is incident to at least one vertex in C. Denote by $VC\left(G\right)$ the smallest size of a vertex cover of G. There are two different but related problems to consider. The first one is estimating the size of the minimum vertex cover (without providing a corresponding vertex cover of that size), and the second one is computing a feasible vertex cover of approximately minimum size.

We show here that, like the maximum-matching size, the minimum vertex-cover size is almost-smooth.

Lemma 3.14. *The minimum vertex-cover size $VC\left(\cdot\right)$ is 2-almost-smooth.*

The proof is similar to the proof of Lemma 3.5, but differs in the justification for the last inequality. Although $m\left(\cdot\right)$ is a maximization problem and $VC\left(\cdot\right)$ is a minimization problem, they satisfy the same sequence of inequalities, albeit for different reasons.

Proof. The first three requirements are clear, as $VC\left(\cdot\right)$ is non-negative, bounded and monotone, since on a sub-segment of the stream the minimum vertex cover can not be larger. Hence, we are only left to show the almost-smoothness property. Let $\varepsilon \in (0, 1)$ and let A, B and C be disjoint segments of the stream satisfying $\varepsilon VC\left(AB\right) \leq VC\left(B\right)$. Observe that a union of a minimum vertex cover on AB and a minimum vertex cover on BC is clearly a feasible (not necessarily minimum) vertex cover on ABC, and since it is a minimization problem we obtain $VC\left(AB\right) + VC\left(BC\right) \geq VC\left(ABC\right)$. Therefore,

$$\begin{aligned} 2VC\left(BC\right) \geq & VC\left(B\right) + VC\left(BC\right) \geq \varepsilon VC\left(AB\right) + VC\left(BC\right) \\ \geq & \varepsilon\left(VC\left(AB\right) + VC\left(BC\right)\right) \geq \varepsilon VC\left(ABC\right). \end{aligned}$$

□

Remark 3.15. The almost-smoothness parameter $d = 2$ in Lemma 3.14 is tight, namely, there are segments A, B, C of a stream that define a graph, for which $\left(1-\varepsilon\right)VC\left(AB\right) \leq VC\left(B\right)$ and $\frac{1}{2}\left(1-\varepsilon\right)VC\left(ABC\right) = VC\left(BC\right)$. For example, the graph and the segments from Remark 3.6 satisfies $VC\left(AB\right) = VC\left(B\right) = VC\left(BC\right) = n$ while $VC\left(ABC\right) = 2n$. Thus, $VC\left(\cdot\right)$ is not smooth as per the original definition of smoothness due to Braverman and Ostrovsky [BO07].

Hence, we can use the Almost-Smooth-Histogram approach to estimate the size of the minimum vertex-cover in the sliding-window model, as explain in the next section.

3.3.1 Vertex-Cover Estimation

First we consider estimating the size of the minimum vertex cover in the sliding-window model. We provide the first sub-linear space algorithm in the sliding-window model for estimating $VC(\cdot)$, for some families of graphs, as explained below.

A graph $G = (V, E)$ is said to be VDP (stands for vertex-disjoint paths) if G is a union of vertex disjoint paths. We show two sliding-window algorithms for different families of graphs. One with $\widetilde{O}(\sqrt{n})$ space obtaining almost 3.125-approximation for the family of VDP graphs and the other one with $\operatorname{poly}\log n$ space obtaining almost 4-approximation for graphs of arboricity $\alpha = 1$. Observe that the results are incomparable, since the first algorithm has better approximation ratio but its space complexity is much bigger. Also, the second algorithm is applicable for a larger family of graphs.

For the family of VDP graphs there is a randomized algorithm in the turnstile streaming model to approximate $VC(\cdot)$, presented in [vH16]. Using the standard argument of computing a median of $\log\delta^{-1}$ parallel repetitions, to achieve low failure probability δ, we can state this result as follows.

Theorem 3.16. *[vH16, Theorem 1.1] For every $\varepsilon, \delta \in (0, 1)$, there exists a $\left(\frac{5}{4} + \varepsilon, \delta\right)$-approximation streaming algorithm for the size of the minimum vertex cover of an input VDP graph that uses $O(\varepsilon^{-1}\sqrt{n}\log^2 n \log\delta^{-1})$ space.*

Therefore, using Theorem 3.7 we obtain as a corollary the following result for the sliding-window model.

Theorem 3.17. *For every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there exists a sliding-window $(3.125 + \varepsilon, \delta)$-approximation algorithm for $VC(\cdot)$ in VDP graphs that uses $O(\varepsilon^{-2}\sqrt{n}\log^4 n \log\frac{1}{\varepsilon\delta})$ space.*

Observe that a VDP graph has arboricity $\alpha = 1$, because it is a forest, and in particular it is a bipartite graph. Recall that according to Kőnig's theorem, in a bipartite graph the size of a minimum vertex cover equals the size of a maximum-matching. Therefore, we conclude from Remark 3.13 that there is a $(4 + \varepsilon, \delta)$-approximation algorithm for the minimum vertex cover in VDP graphs using $\operatorname{poly}\log$ space. Obviously it extends to all forests, i.e., graphs with arboricity $\alpha = 1$. Comparing to Theorem 3.17, the following Theorem has slightly worse approximation factor but its space complexity is much better, moreover, its applicable for a wider family of graphs.

Theorem 3.18. *For every $\varepsilon, \delta \in \left(0, \frac{1}{2}\right)$, there is a sliding-window $(4 + \varepsilon, \delta)$-approximation algorithm for the size of the minimum vertex cover size in a forest graph, with space bound $O(\varepsilon^{-3}\log^3 n \log\frac{1}{\varepsilon\delta})$ and update time $O(\varepsilon^{-3}\log^3 n \log\frac{1}{\varepsilon\delta})$.*

3.3.2 Vertex-Cover Approximation

Here we consider computing a feasible vertex cover of approximately minimum size. We improve the approximation ratio of the algorithm of [vH16] from $8+\varepsilon$ to $4+\varepsilon$, using a tighter analysis of his algorithm.

A maximal matching is a matching that cannot be extended by adding an edge to it, i.e., a matching M in a graph $G=(V,E)$ is maximal if every edge $e \in E \backslash M$ is adjacent to at least one edge from the matching M. For a stream A of edge insertions, denote by $\widehat{M}(A)$ the greedy matching on A, and denote by $\widehat{m}(A)$ its size. Note that for every stream A the greedy matching $\widehat{M}(A)$ is maximal. Recall that for a matching M we denoted by $V(M)$ the set of all endpoints of edges from M, i.e., $V(M)=\{v \in V \mid \exists u \in V : \{v,u\} \in M\}$.

We show that the greedy-matching size of a stream of edge insertions is almost-smooth.

Lemma 3.19. *The greedy-matching size is* $(2,2)$*-almost-smooth.*

The proof is similar in nature to the proof of Lemma 3.5, but different because $\widehat{m}(\cdot)$ is not monotone, but rather 2-monotone. Furthermore, we can use the actual matching, since it is well structured.

Proof. The first two requirements are clear, as $\widehat{m}(\cdot)$ is non-negative and bounded. For the 2-monotonicity, let A, B be disjoint segments of the stream. Note that for every $e \in \widehat{M}(B)$ at least one of its endpoint is in $V\left(\widehat{M}(AB)\right)$, hence $\widehat{m}(B) \le \left|V\left(\widehat{M}(AB)\right)\right| = 2 \cdot \widehat{m}(AB)$.

For the almost-smoothness property, let $\varepsilon > 0$ and let A, B and C be disjoint segments of the stream satisfying $\varepsilon \widehat{m}(AB) \le \widehat{m}(B)$. For every edge $e \in \widehat{M}(ABC)$, if e is from the stream AB then obviously $e \in \widehat{M}(AB)$, and if e is from the stream C then $e \in \widehat{M}(BC)$, since the greedy matching on the stream BC could add it as did the greedy matching on ABC. Thus, $\widehat{M}(ABC) \subseteq \widehat{M}(AB) \cup \widehat{M}(BC)$, from which we deduce that $\widehat{m}(AB) + \widehat{m}(BC) \ge \widehat{m}(ABC)$. As obviously $\widehat{m}(B) \le \widehat{m}(BC)$ by construction, we obtain

$$\begin{aligned} 2\widehat{m}(BC) \ge& \widehat{m}(B) + \widehat{m}(BC) \ge \varepsilon \widehat{m}(AB) + \widehat{m}(BC) \\ \ge& \varepsilon\left(\widehat{m}(AB) + \widehat{m}(BC)\right) \ge \varepsilon \widehat{m}(ABC). \end{aligned}$$

□

If $M^* \subseteq E$ is a maximal matching in the graph $G=(V,E)$ then the set of vertices $V(M^*)$ is a vertex cover of the graph G, because every edge from E has at least one of its end points in $V(M^*)$ (otherwise the matching M^* would not be maximal). For every stream A the greedy matching $\widehat{M}(A)$ is a maximal matching and thus $V\left(\widehat{M}(A)\right)$ is a vertex cover of the edges from A. Hence, we refer to the greedy matching algorithm also as the greedy vertex cover algorithm, with the only difference that it outputs the vertices $V\left(\widehat{M}(A)\right)$ of the matching, instead of the edges $\widehat{M}(A)$ of the matching.

The greedy vertex cover algorithm achieves 2-approximation in the standard insertion-only streaming model for the minimum vertex cover using $O(n \log n)$ space, because at least one vertex from each matched edge should be in the minimum vertex cover. By using that greedy algorithm and exploiting the 2-almost-smoothness of the minimum vertex cover size we deduce from Theorem 3.7, an $(8+\varepsilon)$-approximation algorithm for reporting a minimum vertex cover in the sliding-window model with $O(\varepsilon^{-1} n \log^2 n)$ space, matching the result of [vH16].

We can do slightly better by using the algorithm of Crouch et al. [CMS13], which is a $(3+\varepsilon)$-approximation to the maximum-matching, with the same space complexity. Their algorithm maintains a greedy matching in various buckets, such that the difference between adjacent buckets is not too large. Specifically, for any adjacent buckets B_i and B_{i+1} it holds that $2\widehat{m}(B_{i+1}) \geq (1-\varepsilon)\widehat{m}(B_i)$. With an easy modification to their algorithm, outputting the greedy matching on the bucket B_1 instead of the bucket B_2, it holds that $V\left(\widehat{M}(B_1)\right)$ is a vertex cover (of $B_1 \supseteq W$) at most $(6+\varepsilon)$-factor larger than the minimum vertex cover on the active window W. Note that the algorithm of [CMS13], and the algorithm of [vH16] is essentially the same. The only difference is that [vH16] storing the vertices instead of the edges, which is what [CMS13] do.

We can do even better, using a tighter analysis of this algorithm of [CMS13]. By leveraging the fact that the greedy-matching size is $(2,2)$-almost-smooth, we obtain a $(4+\varepsilon)$-approximation sliding-window algorithm.

Theorem 3.20. *For every $\varepsilon \in \left(0, \frac{1}{2}\right)$, there is a sliding-window $(4+\varepsilon)$-approximation algorithm for the minimum vertex cover with space bound $O(\varepsilon^{-1} n \log^2 n)$.*

Proof. We use the algorithm of Crouch et al. [CMS13], but output $V\left(\widehat{M}(B_1)\right)$. At the end of the stream we have $2\widehat{m}(B_2) \geq (1-\varepsilon)\widehat{m}(B_1)$, since the greedy matching size is $(2,2)$-almost-smooth. Since the minimum vertex cover is monotone and $W \subseteq B_1$ $VC(W) \leq VC(B_1) \leq \left|V\left(\widehat{M}(B_1)\right)\right| = 2 \cdot \widehat{m}(B_1)$. Note that $V\left(\widehat{M}(B_1)\right)$ is indeed a vertex cover on the active window W, since it is a vertex cover on B_1. Additionally, $VC(W) \geq VC(B_2) \geq \widehat{m}(B_2)$, since $VC(\cdot)$ is monotone, $B_2 \subseteq W$ and the minimum vertex cover size is at least the size of any matching. For $\varepsilon < \frac{1}{2}$ it holds that $\frac{1}{1-\varepsilon} \leq 1+2\varepsilon$, and we obtain

$$\left|V\left(\widehat{M}(B_1)\right)\right| = 2 \cdot \widehat{m}(B_1) \leq 4(1+2\varepsilon) \cdot \widehat{m}(B_2) \leq 4(1+2\varepsilon) \cdot VC(W),$$

which means that $V\left(\widehat{M}(B_1)\right)$ is a vertex cover on the active window W and it is at most a factor $4(1+2\varepsilon)$ larger then $VC(W)$, as required. □

3.4 Parameterized Sliding-Window

Chitnis et al. [CCHM15] showed a deterministic parameterized algorithm in the insertion-only model for the two aforementioned graph problems, minimum vertex-cover and maximum-matching, both parameterized by the maximum-matching size k. Specifically, they showed in [CCHM15, Theorem 1.1] a deterministic exact algorithm outputting a minimum vertex-cover with space $O(k^2)$, where k is an upper bound on the size of the minimum vertex-cover in the stream. It can be verified, using Lemma 3.21, that the same algorithm can be used for outputting a maximum-matching with the same space complexity.

Lemma 3.21. *For a graph $G = (V, E)$ with $m(G) \leq k$, let M be an arbitrary maximal-matching and denote by V_M the set of matched vertices in M. For each vertex $v \in V_M$ let $E(v)$ be a set of $\min\{\deg(v), 2k\}$ arbitrary edges from E incident on v. Define the subgraph $H = (V_H, E_H)$, with $E_H = M \cup \bigcup_{v \in V_M} E(v)$ and $V_H = \{v \in V \mid \exists u \in V, \{u, v\} \in E_H\}$. Then, the subgraph H has the same maximum-matching size as G, i.e., $m(G) = m(H)$.*

Proof. We start by claiming that every edge in G but not in H is incident on a high degree vertex. To see this, let $e \in E \backslash E_H$. At least one of its endpoints is in V_M, since M is a maximal-matching in G. Denote this endpoint by $v \in V_M$, then the degree of v in H is $\deg_H(v) \geq 2k + 1$, because e not in H implies that $E(v)$ is of size $2k$ and there is a matched edge in M incident on v.

Let M^* be a maximum-matching in G. Let $M^*_H = M^* \cap E_H$ be a matching in H, and let $M^* \backslash M^*_H = \{e_1, \cdots, e_d\}$ for some $d \in \mathbb{N}$. Note that $M^*_H = M^* \backslash \{e_1, \cdots, e_d\}$ and thus $|M^*_H| = |M^*| - d$. By our claim above, each edge e_i for $i \in [d]$ has an endpoint $v_i \in e_i$ with degree at least $2k+1$. We can iteratively add d edges to the matching M^*_H, such that each vertex v_i is matched to yet unmatched vertex in H, as follows.

Since $v_1 \in e_1$ has at least $2k + 1$ neighbors in H, while only at most $2 \cdot |M^*_H| = 2(k - d)$ vertices are matched in M^*_H, there are at least $2d + 1$ neighbors of v_1 (in H) that are not matched in M^*_H. Picking arbitrary unmatched neighbor u_1 of v_1, we add the edge $\{v_1, u_1\}$ to M^*_H, obtaining the matching $M_1 = M^*_H \cup \{v_1, u_1\}$. We continue similarly, for every $i = 2, \ldots, d$, selecting a yet unmatched neighbor u_i of v_i to obtain the matching $M_i = M_{i-1} \cup \{v_i, u_i\}$. Observe that v_i indeed has a yet unmatched neighbor (in H), since v_i has at least $2k + 1$ neighbors in H while there are only

$$2 \cdot |M_{i-1}| = 2 \cdot (|M^*_H| + i - 1) = 2 \cdot (|M^*| - d + i - 1) < 2k$$

matched vertices in M_{i-1}. After d iterations M_d is a matching in H of size $|M_d| = |M^*| - d + d = |M^*|$, and hence $m(G) = m(H)$, as required. □

Both $m(\cdot)$ and $VC(\cdot)$ are 2-almost-smooth by Lemmas 3.5 and 3.14, and thus we can use the streaming algorithm of [CCHM15] in our Theorem 3.7 to obtain the following sliding-window algorithm for both problems.

Theorem 3.22. *For every $\varepsilon \in \left(0, \frac{1}{2}\right)$, there is a deterministic $(2+\varepsilon)$-approximation algorithm for maximum-matching and vertex-cover in the sliding-window model with space bound $O(\varepsilon^{-1}k^2 \log n)$, where k is an upper bound on the maximum-matching size in the stream.*

Remark 3.23. The update time of our sliding-window algorithm is $O(\varepsilon^{-1}t\,(n,m) \log n)$, where $t\,(n,m)$ is the time complexity of an algorithm to extract a maximum-matching or a minimum vertex-cover in the (non-streaming) RAM model, in graphs with n vertices and m edges. For maximum-matching there are few algorithms with time bound $O(\sqrt{n}m)$, due to Micali and Vazirani [MV80, Vaz94] and Blum [Blu90]. Note that in our case $|V| = O(k^2)$ and $|E| = O(k^2)$, as every graph composed of a maximal-matching of size at most k and $O(k)$ more edges on each vertex from the matching. Hence, for maximum-matching $t = O(k^3)$. For parameterized minimum vertex-cover, the best known algorithm, due to Chen, Kanj and Xia [CKX10], finds a minimum vertex-cover in time $O(1.2738^k + k \cdot n)$ in graphs $G = (V,E)$ with $|V| = n$ and $|E| = m$, where k is an upper bound on the size of the minimum vertex-cover. Hence, for vertex-cover $t = O(1.2738^k + k^3)$, as $|V| = O(k^2)$.

In the turnstile streaming model, Chitnis et al. [CCE+16] showed a randomized parameterized streaming algorithm for both maximum-matching and minimum vertex-cover with the same space complexity. Their method samples $\widetilde{O}(k^2)$ edges using an ℓ_0-sampling algorithm, in some non-obvious manner (as explain below), such that the sampled subgraph has, with high probability, the same maximum-matching size (and minimum vertex-cover size) as the entire graph (see Theorem 3.26).

Specifically, given a graph $G = (V,E)$, they defined in [CCE+16] a distribution $\text{Sample}_G\,(b,r)$ over subgraphs of G such that sampling according to it, with appropriately chosen parameters b and r, results with high probability in a subgraph G' with $m\,(G) = m\,(G')$. For completeness we repeat here their definitions and notation, although less generally. Given a graph $G = (V,E)$, consider a vertex coloring $c : V \to [b]$ for some $b \in \mathbb{N}$. For each color $i \in [b]$ denote $V_i = \{v \in V : c\,(v) = i\}$. For an edge $e = (v,u) \in E$, define $c\,(e) = \{c\,(v)\,, c\,(u)\}$. Denote $B = \{C \subseteq [b] :\ |C| \leq 2\}$ and we say for each $C \in B$ that an edge e of G is C-colored if $c\,(e) = C$. For each $C \in B$, the set E_C contains a single edge chosen uniformly at random from the set of all C-colored edges. If there are no C-colored edges, then $E_C = \emptyset$. The union of these sets defines the random graph $G' = (V,E')$, i.e.,

$$E' = \bigcup_{C \in B} E_C.$$

Definition 3.24. $\text{Sample}_G\,(b,1)$ is the distribution over subgraphs of G generated as explained above, where c is chosen uniformly at random from a family of pairwise independent hash functions. $\text{Sample}_G\,(b,r)$ is the distribution over subgraphs formed by taking the union of r independent subgraphs sampled from $\text{Sample}_G\,(b,1)$.

Remark 3.25. Note that a graph $G' = (V,E')$ sampled according to $\text{Sample}_G\,(b,r)$ has at most rb^2 edges.

Chitnis et al. [CCE$^+$16] then showed that given a graph $G = (V, E)$, sampling according to the distribution $\text{Sample}_G(b, r)$, for suitable b and r, maintain the maximum-matching size and the size of the minimum vertex-cover, with high probability.

Theorem 3.26. *[CCE$^+$16, Theorem 3.1] For a graph $G = (V, E)$, suppose $m(G) \leq k$. Let $G' = (V, E')$ be sampled according to $\text{Sample}_G(1000k, O(\log k))$. Then, with probability at least $1 - \frac{1}{poly(k)}$,*

$$m(G) = m(G') \qquad \textit{and} \qquad VC(G) = VC(G') \ .$$

During the stream, the update procedure of their algorithm only needs to maintain $\widetilde{O}(k^2)$ instances of an ℓ_0-sampling algorithm, one for each $C \in B$. Following their approach, we implement the algorithm in the sliding-window model, relying on ℓ_0-sampling in the sliding-window model (see Section 4.1). Since the sliding-window model allows only insertion of edges and no edge is inserted more than once, we can use a simpler and more efficient method for ℓ_0-sampling. Braverman et al. [BOZ09] showed a reservoir sampling algorithm for the sliding-window model, which, adapted for graph-streaming, maintains a uniformly distributed edge from the active window W using $O(\log^2 n)$ bits.

Since we need to sample uniformly at random from sub-streams, defined by each color set $C \in B$, the reservoir-sampling technique that would be more appropriate is to sample in the timestamp window. Recall that in the timestamp model, each edge from the stream has an additional parameter, a timestamp of its arrival time, i.e., the stream consists of pairs $\sigma_i = (e_i, t_i)$ which means that edge e_i arrives at time t_i. In this model the active window is defined using a time bound T, and denoted by W^T, indicating that edges arriving in the last T time units are active, and older edges should be ignored, as they are considered obsolete. Namely, at time t the active window is defined as $W_t^T = \{\sigma_i = (e_i, t_i) \,|t_i \geq t - T\}$. We will use the timestamp reservoir-sampling algorithm of [BOZ09]. Specifically, their algorithm adapted to the graph-streaming model can be stated as follows.

Theorem 3.27. *[BOZ09, Theorem 3.9] There exist an algorithm, denoted by $\mathcal{A}_{RS}$, that maintains a random sampled edge over the active window in the timestamp-based model, with space $O(\log n \log N)$, where N is a bound on the number of active edges and n is the number of vertices in the streamed graph.*

We now show how to implement a sampling according to the distribution $\text{Sample}_G(b, 1)$ in the (non-timestamp) sliding-window model.

Lemma 3.28. *Algorithm 2 samples a subgraph, in the sliding-window model, according to $\text{Sample}_G(b, 1)$ using $O(b^2 \log^2 n)$ space.*

Proof. Let $\mathcal{A}_{RS}$ be the timestamp reservoir-sampling algorithm due to [BOZ09], with time bound $T = w$. For every $C \in B$ instance $\mathcal{A}_C$ of algorithm $\mathcal{A}_{RS}$ sees only the sub-stream $\mathcal{S}_C$ defined on edges e_i colored by C. Since $\mathcal{A}_C$ maintains a random sampled

Algorithm 2 Sliding-Window Algorithm for Sampling according to Sample $(b, 1)$

Input: A graph $G = (V, E)$ in a streaming fashion, w for the size of a window and b for the number of colors.

Initialization: Choose uniformly at random a coloring $c : V \to [b]$ from a family of pairwise independent hash functions.

Update Procedure: For each $C \in B$ maintain an instance $\mathcal{A}_C$ (with time bound $T = w$) of algorithm $\mathcal{A}_{RS}$ only on the sub-stream consisting of edges of color C, keeping the current sampled edge in E_C (if there are no active edges colored C in the last w items from stream define $E_C = \emptyset$).

Query Procedure: Report the graph $G' = (V, E')$ where $E' = \bigcup\limits_{C \in B} E_C$.

edge over the active window in the timestamp model, we deduce that E_C is a random sampled edge over the timestamp window of the stream $\mathcal{S}_C$. The edge from E_C (in case it is not empty) is in W^T, for $T = w$, and because the timestamp of each edge is its index in the original stream $\mathcal{S}$, it is also in the active window W. Therefore, E_C contains a single edge chosen uniformly at random from the set of all C-colored edges from W, and hence, the subgraph $G' = (V, E')$, with $E' = \bigcup_{C \in B} E_C$ is indeed sampled according to $\text{Sample}_G (b, 1)$ from the graph G defined by the active edges. Observe that Algorithm 2 maintains $|B|$ instances of $\mathcal{A}_{RS}$, and $|B| \leq b^2$. The space complexity of $\mathcal{A}_{RS}$ is $O(\log n \log N)$, where $N \leq w$. Recall that $w \leq n^2$, and thus, the space complexity of Algorithm 2 is $O(b^2 \log^2 n)$, as required. $\square$

Remark 3.29. By running r independent instances of Algorithm 2, we obtain a sampled graph $G' = (V, E')$ according to $\text{Sample}_G (b, r)$ in the sliding-window model.

We run $O(\log k)$ independent instances of Algorithm 2, with $b = 1000k$, and, according to Remark 3.29, we obtain a sampled graph $G' = (V, E')$ according to $\text{Sample}_G (1000k, O(\log k))$ in the sliding-window model. Then, when queried (or at the end of the stream), we calculate a maximum-matching (or a vertex-cover) on G' using one of the algorithms mentioned in Remark 3.23. We deduce the following theorem as a corollary.

Theorem 3.30. *There is a randomized sliding-window algorithm that outputs, with probability at least $1 - \frac{1}{poly(k)}$, a maximum-matching with space $O(k^2 \log^3 n)$ and update time $O(k^2 \log^2 n)$, where k is an upper bound on the maximum-matching size in the stream.*

Remark 3.31. As mentioned above, the same algorithm can be used for computing a minimum vertex-cover, using the same space and update time.

4 Sampling in the Frequency-Vector Model

In the streaming model, the input data is given by a stream of items in arbitrary order, which can be read sequentially in only one direction. The objective is to compute, or estimate, some statistic of the data without storing the entire stream. In the sliding-window variant, the goal is to compute some statistic of the last w items from the stream. Thus, the last w items from the stream are referred to as the active items, and they form the active window, which is denoted by W throughout.

In this section we concentrate on the frequency-vector setting, where the input is a stream of non-negative updates (increments) to the entries of a vector $x \in \mathbb{R}^n$, and the goal is usually to compute some function of the resulting x, e.g. it's norm. In the ℓ_p-sampling variant the goal is to sample a coordinate $i \in [n]$ proportionally to its contribution to the ℓ_p-norm, i.e., $\|x\|_p^p = \sum_{i=1}^{n} |x_i|^p$. See [CF14] for more information.

We design a sliding-window algorithm for ℓ_p-sampling, based on the smooth histogram method of Braverman and Ostrovsky [BO07]. It uses two known techniques. The first one is a sliding-window algorithm for approximating the F_p moment (of the active window). The second one is an ℓ_p-sampling algorithm in the standard (not sliding-window) streaming model, that will be applied to a suffix of the stream, that approximates the active window. We start by proving the result for $p = 0$, and at the end generalize it to ℓ_p-sampling for every $0 \le p < \infty$.

Recall that the *support* of a vector $x \in \mathbb{R}^n$ is defined to be $\text{supp}(x) = \{i \in [n] \,|\, x_i \neq 0\}$, and the ℓ_0-*norm* of x is defined as its support size $\|x\|_0 = |\text{supp}(x)|$. For $p > 0$, the ℓ_p-*norm* is defined by $\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$, and the F_p-*moment* is defined as the p-th power of the ℓ_p-norm, i.e., $F_p(x) = \|x\|_p^p = \sum_{i=1}^{n} |x_i|^p$. For $p = 0$, the ℓ_0-*norm distribution*, ℓ_0-*distribution* for short, of a vector $x \in \mathbb{R}^n$ is the uniform distribution over the non-zero coordinates of x, i.e., the support of x. For $p > 0$, the ℓ_p-*distribution* of a vector $x \in \mathbb{R}^n$, is defined by the probability density function f satisfying

$$\forall i \in [n], \quad f(i) = \frac{|x_i|^p}{\|x\|_p^p}.$$

Let $\mathcal{S} = ((i_1, a_1), (i_2, a_2), \cdots, (i_m, a_m))$ be a stream, where $i_j \in [n]$ and $a_j \in \mathbb{R}$ for every $j \in [m]$. The frequency-vector $x \in \mathbb{R}^n$ of the stream $\mathcal{S}$ is defined for every $i \in [n]$ to be $x_i = \sum_{i_j = i} a_j$. In the sliding-window model, with window size w, we are only interested in the last w updates from the stream, i.e., the frequency-vector x is defined for every $i \in [n]$ to be $x_i = \sum_{j > m-w : i_j = i} a_j$. Throughout, we assume that the size w of the active window is fixed and known in advance, and denote by W the active window, i.e., the last w updates from the stream seen thus far. For a stream $\mathcal{S}$ with frequency-vector x, we define the F_p moment of $\mathcal{S}$ to be $F_p(\mathcal{S}) = F_p(x)$. For ease of exposition, we sometimes also denote by $\|\mathcal{S}\|_p$ the ℓ_p-norm of x and by $\text{supp}(\mathcal{S})$ the support of x.

Datar et al. [DGIM02] proved a space lower bound of $\Omega(w)$ bits for estimating the ℓ_0-norm when negative increments a_j are allowed. Therefore, we restrict the increments a_j to be positive. Moreover, we assume that for some constant $R \in \mathbb{N}$, known a priori (to the algorithm), it holds that $0 < a_j \leq R$ for every $j \in [m]$.

We use the definition of smoothness due to Braverman and Ostrovsky [BO07], see Remark 3.4. They proved that for $0 < p \leq 1$ the F_p moment is $(\varepsilon, \varepsilon)$-smooth, and for $p > 1$ the F_p moment is $\left(\varepsilon, \frac{\varepsilon^p}{p^p}\right)$-smooth. Hence, the space complexity of our sliding-window algorithm for ℓ_p-sampling for $p \leq 1$ will not depend on p, while for $p > 1$ there will be another factor of $\frac{\varepsilon^p}{p^p}$ in the space and update complexity.

4.1 ℓ_0-sampling

Braverman and Ostrovsky [BO07] did not mention the ℓ_0-norm, although it is also smooth as per their definition of smoothness. Hence we start by proving that easy fact, using their notations.

Lemma 4.1. *The ℓ_0-norm is $(\varepsilon, \varepsilon)$-smooth, as per the definition of [BO07].*

Proof. Obviously the first three out of four requirements are true for the ℓ_0-norm, and we are only left to prove the fourth requirement. Let $\varepsilon \in (0,1)$, let A be a segment of a stream, B a suffix of A, and let C a disjoint segment of a stream adjacent to A. We need to show that if $(1-\varepsilon) L_0(A) \leq L_0(B)$ then $(1-\varepsilon) L_0(AC) \leq L_0(BC)$. Let $S_A, S_B, S_C \subseteq [n]$ be the sets of distinct elements corresponding to the streams A, B and C, respectively. Hence, $L_0(X) = |S_X|$ for each $X \in \{A, B, C\}$, and note that $L_0(AC) = |S_A \cup S_C|, L_0(BC) = |S_B \cup S_C|$ and $S_B \subseteq S_A$ since B is a suffix of A. Observe that since $S_B \subseteq S_A$ it holds that $|S_B \cap S_C| \leq |S_A \cap S_C|$, and so if $(1-\varepsilon)|S_A| \leq |S_B|$, then we obtain the required inequality:

$$\begin{aligned} |S_B \cup S_C| &= |S_B| + |S_C| - |S_B \cap S_C| \\ &\geq (1-\varepsilon)|S_A| + |S_C| - |S_A \cap S_C| \\ &\geq (1-\varepsilon)(|S_A| + |S_C| - |S_A \cap S_C|) \\ &= (1-\varepsilon)|S_A \cup S_C|. \end{aligned}$$

□

For a distribution $\mathcal{D}$ over a set S we denote by $\mathcal{D}(s)$, for $s \in S$, the probability of outcome s, i.e., $\mathcal{D}(s) = \Pr_{R \sim \mathcal{D}}[R = s]$. Note that the probability of outcome $s \notin S$ is $\Pr_{R \sim \mathcal{D}}[R = s] = 0$. For a vector $x \in \mathbb{R}^n$ and an accuracy parameter $\varepsilon > 0$ we say that a distribution $\mathcal{D}$ is $(1+\varepsilon)$-uniform over the support of x if its probability density function f satisfies

$$\forall i \in \operatorname{supp}(x), \quad f(i) \in \frac{1 \pm \varepsilon}{\|x\|_0},$$

and the probability of outcomes not in $\operatorname{supp}(x)$ is zero.

Definition 4.2. We say that S is an ℓ_0-sampling algorithm with parameters $\varepsilon, \delta \in [0,1]$, if on an input stream $\mathcal{S}$ with frequency-vector $x \in \mathbb{R}^n$, its output X is within total variation distance δ of a $(1+\varepsilon)$-uniform distribution over the support of x.

Recall that a random variable Z is said to be an (ε, δ)-approximation of $z \in \mathbb{R}$ if

$$\Pr\left[(1-\varepsilon)\, z \le Z \le (1+\varepsilon)\, z\right] \ge 1-\delta.$$

Definition 4.3. We say that L is an ℓ_0-norm estimation algorithm with parameters $\varepsilon, \delta \in [0,1]$, if on an input stream $\mathcal{S}$ with frequency-vector $x \in \mathbb{R}^n$, its output X is an (ε, δ)-approximation of $\|x\|_0$.

Remark. Usually, in the context of probabilistic algorithms we think of ε as a small constant and δ as an inverse polynomial in n.

Theorem 4.4. *Suppose there are an ℓ_0-sampling algorithm S with parameters ε_1, δ_1, having space bound $s_1(\varepsilon_1, \delta_1, n)$, and an ℓ_0-norm estimation algorithm L with parameters ε_2, δ_2, having space bound $s_2(\varepsilon_2, \delta_2, n)$. Then there is an algorithm for ℓ_0-sampling in the sliding-window model with space bound*

$$O(\tfrac{1}{\varepsilon_2}\log w\left(s_1(\varepsilon_1, \delta_1, n) + s_2\left(\tfrac{\varepsilon_2}{2}, \tfrac{\delta_2\varepsilon_2}{2w\log w}, n\right) + \log w\right)).$$

The algorithm's output $X \in [n]$ is within total variation distance $\delta_1 + \delta_2 + 3\varepsilon_2(1 + \delta_1\|W\|_0)$ of a $(1 + (\varepsilon_1 + \varepsilon_2))$-uniform distribution over $\operatorname{supp}(W)$.

Remark. The algorithm's output actual distribution is over the set $[n] \cup \{\perp\}$, since with probability δ_1 it could return the symbol $\perp$, meaning "failure". Nevertheless, we state the algorithm's output as $X \in [n]$, because the algorithm can change the outcome $\perp$ to some predetermined (or uniform) index in $[n]$.

To prove Theorem 4.4 we use Claim 1.15 to assume without loss of generality that the length of the entire stream is bounded by $2w$.

Proof. Let S be an ℓ_0-sampling algorithm with parameters ε_1 and δ_1, and let L be an ℓ_0-norm algorithm with parameters $\frac{\varepsilon_2}{2}$ and $\frac{\delta_2\varepsilon_2}{2w\log w}$. For $k = O\left(\frac{1}{\varepsilon_2}\log w\right)$, our algorithm maintains k instances of algorithm S and k instances of algorithm L, denoted by $S^1, \dots, S^k$ and $L^1, \dots, L^k$, respectively. Each pair of instances S^i, L^i will correspond to a suffix B_i of the stream, called a "bucket". These buckets will satisfy the invariant $B_1 \supseteq W \supsetneq B_2 \supsetneq B_3 \supsetneq \cdots \supsetneq B_k$, where W is the active window, i.e., the w last items from the stream. Denote by $\|B_i\|_0$ the ℓ_0-norm of the frequency-vector of the bucket B_i and by $L^i(B_i)$ the approximation to the ℓ_0-norm of the frequency-vector of the bucket B_i given by the instance L^i, then $\Pr\left[L^i(B_i) \notin \left(1 \pm \frac{\varepsilon_2}{2}\right)\|B_i\|_0\right] < \frac{\delta_2\varepsilon_2}{2w\log w}$. In order to use only a small amount of space, whenever two nonadjacent instances will correspond to a suffix of the stream that define a vector of almost the same ℓ_0-norm we will delete all instances between them. In each step of receiving a new item from the stream, the algorithm updates the instances of S and L in the following way.

Algorithm 3 Update and query procedure

```
Update Procedure: Given the next item $(l, a)$ from the stream, update the current
    instances $S^1, \cdots, S^k$ and $L^1, \cdots, L^k$, as follows:
    1. Initialize new instances $S^{k+1}$ and $L^{k+1}$ and start running them on the given
       item $(l, a)$, i.e., the instances $S^{k+1}$ and $L^{k+1}$ correspond to the bucket $B_{k+1}$,
       which is a suffix of the stream starting at the item $(l, a)$.
    2. For each $1 \le i \le k$, update $S^i$ and $L^i$ using the new item $(l, a)$.
    3. For $i = 1, \cdots, k-2$ do:
        (a) Find the largest $j > i$ such that $L^j(B_j) > \left(1 - \frac{\varepsilon_2}{2}\right) L^i(B_i)$.
        (b) For every $i < t < j$ delete instances $S^t$ and $L^t$, and renumber the
            remaining instances accordingly.
    4. If $B_2$ contains the active window $W$, delete both instances $S^1$ and $L^1$, and
       renumber the instances.
Query Procedure: Output a sample according to $S^1$, denoted by $X$.
```

Since the maximum ℓ_0-norm of any vector defined by the last w updates is bounded by w, and $L^{i+2}(B_{i+2}) \le \left(1 - \frac{\varepsilon_2}{2}\right) L^i(B_i)$ for every $1 \le i \le k-2$ (as the "unnecessary" instances were deleted in the process of updating), it follows that the number of instances are bounded by $O(\frac{1}{\varepsilon_2}\log w)$. Hence, the number of times any instance of L is invoked is at most $O(\frac{1}{\varepsilon_2}\log w) \cdot m$, where m is the length of the entire stream, which we assumed to be bounded by $2w$. Since we set the failure probability to be $\frac{\delta_2\varepsilon_2}{2w\log w}$ then by union bound the probability that any invocation of any instance of L fails is δ_2, i.e., with probability $1-\delta_2$ every instance of L succeeds every time it is invoked. If some instance of L fails we say it is a failure of our algorithm, and so from now on we assume that every instance of L succeeds every time it is invoked. Note that we acquire here an additional probability of δ_2 for the event that the algorithm could return anything, including an index $i \notin supp(W)$ or "fail", or output a coordinate $i \in supp(W)$ not according to the right distribution.

Now, let us explain how to achieve $O(\frac{1}{\varepsilon_2}\log w\,(s_1 + s_2 + \log w))$ space complexity, where $s_1 = s_1(\varepsilon_1, \delta_1, n)$ and $s_2 = s_2\left(\frac{\varepsilon_2}{2}, \frac{\delta_2\varepsilon_2}{2w\log w}, n\right)$. Additional to the space required by both instances S^i and L^i, we need to store a counter c_i for every bucket B_i, indicating its initialization time (initialized to $c_i = 1$ and incremented each time a new item arrives), such that we can preform the last step of the algorithm, by comparing the counter for the bucket B_2 to the number w (which is the size of the active window W). This way, for each bucket B_i we store $s_1 + s_2 + \log w$ bits. As we have seen previously, the number of instances of S^i and L^i are bounded by $O(\frac{1}{\varepsilon_2}\log w)$. Therefore, the total number of bits used by the algorithm is $O(\frac{1}{\varepsilon_2}\log w\,(s_1+s_2+\log w))$, as claimed.

Denote by $|B_i|$ the length of the bucket B_i, and note that if buckets B_i and B_{i+1} start at consecutive points in time, i.e., there were no (deleted) buckets between them, then $|B_i| = |B_{i+1}| + 1$. The approximation ratio follows from the next claim, which states that if B_i and B_{i+1} are not consecutive suffixes of the stream, then at the end of the stream $\|B_{i+1}\|_0$ and $\|B_i\|_0$ are close.

Claim. For every $i < k$ we have either $|B_i| = |B_{i+1}| + 1$ or $\|B_{i+1}\|_0 > (1-\varepsilon_2)\|B_i\|_0$ (or both).

Proof. Assume $|B_i| \neq |B_{i+1}| + 1$, which means that at some time t the algorithm has deleted the instances of some bucket B between B_i and B_j, for $j = i+1$. Denote by B_i^t the stream B_i at time t, i.e., B_i^t is the part of the stream that started at the initialization of instance L^i and ended when item t has arrived. Because bucket B was deleted at time t we know that $B_i^t \subsetneq B^t \subsetneq B_j^t$ at time t and of course $L^j\left(B_j^t\right) > \left(1-\frac{\varepsilon_2}{2}\right)L^i\left(B_i^t\right)$, as bucket B was deleted , thus

$$(1-\varepsilon_2)\left\|B_i^t\right\|_0 \le \left(1-\frac{\varepsilon_2}{2}\right)\frac{1-\frac{\varepsilon_2}{4}}{1+\frac{\varepsilon_2}{4}}\left\|B_i^t\right\|_0 \le \frac{1-\frac{\varepsilon_2}{2}}{1+\frac{\varepsilon_2}{4}}L^i\left(B_i^t\right) < \frac{1}{1+\frac{\varepsilon_2}{4}}L^j\left(B_j^t\right) \le \left\|B_j^t\right\|_0.$$

Because the ℓ_0-norm is $(\varepsilon,\varepsilon)$-smooth we deduce that $\left\|B_j^T\right\|_0 > (1-\varepsilon_2)\left\|B_i^T\right\|_0$ at any time T after time t, and in particular for $T = m$ we obtain $\|B_{i+1}\|_0 > (1-\varepsilon_2)\|B_i\|_0$, as required. ☐

Lastly, either $B_1 = W$ or $B_2 \subsetneq W \subsetneq B_1$. Recall that X is the sampled coordinate outputted by the instance S^1. If $B_1 = W$ then X is just a sample according to algorithm S From the stream W, which means that with probability $1-\delta_1$ the instance S^1 will not fail, and output a coordinate $X \in \operatorname{supp}(W)$ according to the required distribution, i.e., for every $i \in \operatorname{supp}(W)$ it holds that

$$\Pr[X = i] = (1\pm\varepsilon_1)\frac{1}{\|W\|_0} \pm \delta_1.$$

Otherwise, if $|B_1| \neq |B_2| + 1$, then using the above claim $\|B_2\|_0 > (1-\varepsilon_2)\|B_1\|_0$, hence

$$\|B_1\|_0 \ge \|W\|_0 \ge \|B_2\|_0 \ge (1-\varepsilon_2)\|B_1\|_0 \ge (1-\varepsilon_2)\|W\|_0.$$

Now, X is a sample, according to algorithm S, from the stream B_1, thus with probability $1-\delta_1$ the instance S^1 will not fail, and in that case, for every $i \in \operatorname{supp}(B_1)$ we have

$$\Pr[X = i] \le (1+\varepsilon_1)\frac{1}{\|B_1\|_0} + \delta_1 \le (1+\varepsilon_1)\frac{1}{\|W\|_0} + \delta_1$$

and also

$$\Pr[X = i] \ge (1-\varepsilon_1)\frac{1}{\|B_1\|_0} - \delta_1 \ge (1-\varepsilon_1)\frac{(1-\varepsilon_2)}{\|W\|_0} - \delta_1 \ge (1-(\varepsilon_1+\varepsilon_2))\frac{1}{\|W\|_0} - \delta_1$$

which means that for every $i \in \operatorname{supp}(B_1)$ we have $\Pr[X = i] = (1 \pm (\varepsilon_1 + \varepsilon_2)) \frac{1}{\|W\|_0} \pm \delta_1$.

Let $bad = \operatorname{supp}(B_1) \setminus \operatorname{supp}(W)$ denote the set of indices that correspond to coordinates that are 0 in the window W but not 0 in the suffix B_1. We consider as a bad event the case where the instance S^1 outputs items $i \in bad$, as we are only interested in coordinates which are not zero in the frequency-vector of the active window, hence we want to bound the probability of that event. Note that $W \subseteq B_1$, thus

$$|bad| = \|B_1\|_0 - \|W\|_0 \leq \|B_1\|_0 - (1 - \varepsilon_2)\|B_1\|_0 = \varepsilon_2 \|B_1\|_0 .$$

Therefore, the probability that instance S^1 will output a "bad" index is

$$\begin{aligned}\Pr[X \in bad] = \sum_{i \in bad} \Pr[X = i] &\leq |bad| \left((1 + \varepsilon_1) \frac{1}{\|B_1\|_0} + \delta_1\right) \\ &\leq (1 + \varepsilon_1) \frac{\varepsilon_2 \|B_1\|_0}{\|B_1\|_0} + \varepsilon_2 \delta_1 \|B_1\|_0 . \\ &\leq 2\varepsilon_2 + \varepsilon_2 \delta_1 \|W\|_0 (1 + 2\varepsilon_2) \leq 3\varepsilon_2 (1 + \delta_1 \|W\|_0)\end{aligned}$$

Overall, we proved that the algorithm outputs a coordinate $i \in \operatorname{supp}(W)$ with probability $\frac{1 \pm (\varepsilon_1 + \varepsilon_2)}{\|W\|_0} \pm \delta_1$, with probability at most δ_1 it fails and returns "fail", with probability at most $3\varepsilon_2 (1 + \delta_1 \|W\|_0)$ it could return an index $i \notin \operatorname{supp}(W)$ and with probability at most δ_2 it could return anything, including an index $i \notin \operatorname{supp}(W)$ or "fail", or output a coordinate $i \in \operatorname{supp}(W)$ not according to the right distribution. □

Although there are good constructions for ℓ_0-sampler algorithms in the turnstile streaming model, e.g. Cormode and Firmani [CF14], for the sliding-window model it is enough to use an ℓ_0-sampler algorithm for insertion-only streams, as we restricted the update values to be positive. Hence, using a theorem of Indyk [Ind99] we can achieve better storage complexity, and also obtain an algorithm having $(1 + \varepsilon)$-uniform output distribution.

Corollary 4.5. *For every $\varepsilon \in (0, 1)$ and every $n \in \mathbb{N}$ there is an ℓ_0-sampling algorithm S for the insertion-only streaming model with parameters ε and $\delta = 0$, i.e., on an input stream D with frequency-vector x, its output X has a $(1 + \varepsilon)$-uniform distribution over the support of x. Algorithm S has space bound $O(\log \frac{1}{\varepsilon} \cdot \log \frac{n}{\varepsilon})$.*

To prove Corollary 4.5 we need the following definition of a minwise-independent family of hash functions and a theorem of Indyk that reduces such a family to constructing an l-wise independent family of hash functions, for a carefully chosen l.

Definition 4.6. A family of functions $\mathcal{H} = \{h : [n] \to [n]\}$ is called (ε, s)-minwise independent if for every $X \subseteq [n]$ of size at most s and every $x \in [n] \setminus X$,

$$\Pr_{h \in \mathcal{H}}[h(x) < \min h(X)] = \frac{1 \pm \varepsilon}{|X| + 1},$$

where $\Pr_{h \in \mathcal{H}}$ denotes the probability space obtained by choosing h uniformly at random from $\mathcal{H}$.

Theorem 4.7. *[Ind99, Theorem 1.1] There exist constants $c, c' > 1$ such that for every $\varepsilon > 0$ and $s \leq \frac{\varepsilon n}{c}$ every l-wise independent family of functions, for $l = c' \log \frac{1}{\varepsilon}$, is (ε, s)-minwise independent.*

Proof of Corollary 4.5. Let $c, c' > 1$ be the constants from Theorem 4.7. Given an accuracy parameters $\varepsilon > 0$ and a dimension $n \in \mathbb{N}$ for the underlying vector, define $n' = \frac{cn}{\varepsilon}$ and $l = c' \log \frac{1}{\varepsilon}$. Before seeing the stream, initialize $j = 0$, $y = 0$, and pick a hash function $h : [n'] \to [n']$ uniformly at random from a family $\mathcal{H}$ of l-wise independent hash functions. In the update procedure, upon seeing an increment a to coordinate i of the frequency-vector $x \in \mathbb{R}^n$, do:

1. if $h(i) < h(j)$ then
   (a) $j \leftarrow i$ and $y = a$
2. else if $h(i) = h(j)$ then
   (a) $y \leftarrow y + a$
3. else do nothing

Basically, the algorithm maintains, on the items seen thus far from the stream, the coordinate j that is minimal according to the hash function h, as well as the value of the frequency-vector x in that coordinate. At the end of the stream, the algorithm report j, and the value y for x_j. Since $\|x\|_0 \leq n = \frac{\varepsilon n'}{c}$, as the items in the stream are from $[n]$ but the domain of h is $[n']$, Theorem 4.7 implies that for every $j \in \operatorname{supp}(x)$,

$$\Pr_{h \in \mathcal{H}} [h(j) < \min h(\operatorname{supp}(x) \setminus \{j\})] = \frac{1 \pm \varepsilon}{\|x\|_0}.$$

Therefore, the minimum, over the items in the stream, of the hash function h is $(1+\varepsilon)$-approximate uniform coordinate from the support of x, as required. For the storage requirement, recall that there is a family $\mathcal{H}$ of l-wise independent hash functions, namely the polynomials of degree l over a finite field of $O(n')$ elements, such that every function $h : [n'] \to [n']$ from $\mathcal{H}$ could be stored in memory using $O(l \cdot \log(n')) = O(\log \frac{1}{\varepsilon} \cdot \log \frac{n}{\varepsilon})$ bits. $\square$

After a long line of research Kane et al. [KNW10] presented an optimal algorithm for ℓ_0-norm estimation (in the standard streaming model), which computes a $1 \pm \varepsilon$ approximation with constant probability and space bound $O(\varepsilon^{-2} + \log n)$. To obtain low failure probability δ it is standard to compute a median of $\log \delta^{-1}$ parallel repetitions. However, recently, Błasiok [Bł18] showed how to avoid such a multiplicative space blow-up of $\log \frac{1}{\delta}$, and presented an (ε, δ)-approximation algorithm with space bound $O(\varepsilon^{-2} \log \delta^{-1} + \log n)$.

Finally, using Corollary 4.5 and the algorithm of Błasiok [Bł18] we obtain as a corollary the next theorem for the sliding-window model.

Corollary 4.8 (Restatement of Theorem 1.14)**.** *For every $\varepsilon, \delta \in (0,1)$ there is an algorithm for ℓ_0-sampling in the sliding-window model with space bound*

$$O(\tfrac{1}{\varepsilon} \log w \left(\log \tfrac{1}{\varepsilon} \cdot \log \tfrac{n}{\varepsilon} + \tfrac{1}{\varepsilon^2} \log \tfrac{w \log w}{\delta\varepsilon} + \log n\right)) = O(\tfrac{1}{\varepsilon^3} \log n \log^2 w \log \tfrac{1}{\delta\varepsilon}).$$

The algorithm's output $X \in [n]$ is within total variation distance $\delta + 3\varepsilon$ of a $(1+\varepsilon)$-uniform distribution over $\operatorname{supp}(W)$*.*

Proof. For $0 < \varepsilon, \delta < 1$ set $\varepsilon_1 = \varepsilon_2 = \frac{1}{2}\varepsilon$ and $\delta_2 = \delta$. Let S be the ℓ_0-sampling algorithm for the insertion-only streaming model from Corollary 4.5, instantiated with parameters ε_1 and $\delta_1 = 0$ (failure probability), and with space complexity $s_1(\varepsilon_1, \delta_1, n) = O(\log \varepsilon_1^{-1} \cdot \log \left(n\varepsilon_1^{-1}\right))$. Let L be the ℓ_0-norm estimation algorithm of Błasiok [Bł18], instantiated with the parameters ε_2 and δ_2, which has space complexity $s_2(\varepsilon_2, \delta_2, n) = O(\varepsilon_2^{-2} \log \delta_2^{-1} + \log n)$. Using Theorem 4.4, pluging-in s_1 and s_2, we obtain an algorithm for ℓ_0-sampling in the sliding-window model with space bound

$$O(\tfrac{1}{\varepsilon} \log w \left(\log \tfrac{1}{\varepsilon} \cdot \log \tfrac{n}{\varepsilon} + \tfrac{1}{\varepsilon^2} \log \tfrac{w \log w}{\delta\varepsilon} + \log n\right)).$$

Observe that the space complexity is upper bounded by $O(\frac{1}{\varepsilon^3} \log n \log^2 w \log \frac{1}{\delta\varepsilon})$. The algorithm outputs a coordinate $i \in \operatorname{supp}(W)$ with probability $\frac{1 \pm (\varepsilon_1 + \varepsilon_2)}{\|W\|_0} \pm \delta_1 = \frac{1 \pm \varepsilon}{\|W\|_0}$. There is $\delta_1 = 0$ probability to return "fail", and with probability at most $3\varepsilon_2 (1 + \delta_1 \|W\|_0) + \delta_2 = 3\varepsilon + \delta_2$ it could return any index from $[n]$, including an index $i \notin \operatorname{supp}(W)$. $\square$

4.2 ℓ_p-sampling, for $p > 0$

Moving to the general case, for $0 < p < \infty$, we will need the following general definition for an F_p-moment estimation algorithm. From now on, let $p \in \mathbb{R}$ be a positive constant.

Definition 4.9. We say that F is an F_p-moment estimation algorithm with parameters $\varepsilon, \delta \in [0,1]$, if on an input stream $\mathcal{S}$ with frequency-vector $x \in \mathbb{R}^n$, its output X is an (ε, δ)-approximation of $F_p(x) = \|x\|_p^p$.

Similarly to a $(1+\varepsilon)$-uniform distribution, we define a $(1+\varepsilon)$-approximate ℓ_p-norm distribution, or $(1+\varepsilon)$-ℓ_p-distribution for short, as follows. For a vector $x \in \mathbb{R}^n$ and an accuracy parameter $\varepsilon > 0$ we say that a distribution $\mathcal{D}$ over $[n]$ is a $(1+\varepsilon)$-approximate ℓ_p-norm distribution of the vector x if its probability density function f satisfies

$$\forall i \in [n], \quad f(i) \in (1 \pm \varepsilon) \frac{|x_i|^p}{\|x\|_p^p}.$$

Definition 4.10. We say that S is an ℓ_p-sampling algorithm with parameters $\varepsilon, \delta \in [0,1]$, if on an input stream $\mathcal{S}$ with frequency-vector $x \in \mathbb{R}^n$, its output X is within total variation distance δ of a $(1+\varepsilon)$-ℓ_p-distribution of the vector x.

For two random variables X and Y we write $\mathrm{d}_{\mathrm{TV}}(X,Y)$ to denote their total variation distance. Moreover, we denote throughout $[X > Y] = \{i \in [n] : \Pr[X = i] > \Pr[Y = i]\}$, where X and Y take values only in the set $[n]$. We use extensively the following equivalent more convenient definition for the total variation distance given in [LP17, Remark 4.3].

Claim 4.11. [LP17, Remark 4.3] Let X and Y be two random variables over the set $[n]$. Then,

$$\mathrm{d}_{\mathrm{TV}}(X,Y) = \sum_{i\in[X>Y]} \Pr[X=i] - \Pr[Y=i].$$

For two vectors $x, y \in \mathbb{R}^n$ we write $y \le x$ if $y_i \le x_i$ for every $i \in [n]$. Before stating and proving the main theorem of this section we will need two simple lemmas bounding the total variation distance of similar distributions.

Lemma 4.12. *Let $x \in \mathbb{R}^n$ be a vector such that $0 \le x$. Let X be a random variable with an ℓ_p-distribution of x, and let $\tilde{X}$ be a random variable with a $(1+\varepsilon)$-ℓ_p-distribution of x. Then $\mathrm{d}_{\mathrm{TV}}\left(X,\tilde{X}\right) \le \varepsilon$.*

Proof. Observe that

$$\mathrm{d}_{\mathrm{TV}}\left(X,\tilde{X}\right) = \sum_{i\in\left[\tilde{X}>X\right]} \Pr\left[\tilde{X}=i\right] - \Pr[X=i] \le \sum_{i\in\left[\tilde{X}>X\right]} \frac{(1+\varepsilon)\,x_i^p}{\|x\|_p^p} - \frac{x_i^p}{\|x\|_p^p} = \frac{\varepsilon}{\|x\|_p^p} \sum_{i\in\left[\tilde{X}>X\right]} x_i^p \le \varepsilon.$$

□

Lemma 4.13. *Let $x, y \in \mathbb{R}^n$ be vectors such that $0 \le y \le x$ and $\|x\|_p^p \le (1+\varepsilon)\|y\|_p^p$. Let X be a random variable with an ℓ_p-distribution of x, and let Y be a random variable with an ℓ_p-distribution of y. Then $\mathrm{d}_{\mathrm{TV}}(X,Y) \le \varepsilon$.*

Proof. We know that $\frac{1}{\|y\|_p^p} \le \frac{1+\varepsilon}{\|x\|_p^p}$, and $y_i \le x_i$ for every $i \in [n]$, hence

$$\mathrm{d}_{\mathrm{TV}}(X,Y) = \sum_{i\in[Y>X]} \Pr[Y=i] - \Pr[X=i] \le \sum_{i\in[Y>X]} \frac{(1+\varepsilon)\,x_i^p}{\|x\|_p^p} - \frac{x_i^p}{\|x\|_p^p} = \frac{\varepsilon}{\|x\|_p^p} \sum_{i\in[Y>X]} x_i^p \le \varepsilon.$$

□

Now we can present the algorithm for ℓ_p-sampling in the sliding-window model for $0 < p < \infty$, which is similar to the ℓ_0-sampling algorithm presented above.

Theorem 4.14. *If there is an ℓ_p-sampling algorithm S with space bound $s_1(\varepsilon_1, \delta_1, n)$, and there is an F_p-moment estimation algorithm F with space bound $s_2(\varepsilon_2, \delta_2, n)$, then there is an algorithm for ℓ_p-sampling in the sliding-window model, with space bound*

$$O(\varepsilon_2^{-1} \log Rw \left(s_1(\varepsilon_1, \delta_1, n) + s_2\left(\frac{\varepsilon_2}{2}, \frac{\delta_2 \varepsilon_2}{Cw \log Rw}, n\right) + \log w\right)), \quad \text{for } 0 < p \le 1$$

and

$$O(\frac{p^p}{\varepsilon_2^p}\log Rw\left(s_1\left(\varepsilon_1,\delta_1,n\right)+s_2\left(\frac{\varepsilon_2^p}{2p^p},\frac{\delta_2\varepsilon_2^p}{Cp^pw\log Rw},n\right)+\log w\right)),\quad \textit{for } p\geq 1,$$

for sufficiently large constant C. The algorithm's output $X^{alg}\in[n]$ is within total variation distance $\delta_1+\delta_2+\varepsilon_1+\varepsilon_2$ of an ℓ_p-distribution of the frequency-vector of the active window W.

Remark. Furthermore, the algorithm's output $X^{alg}\in[n]$ is within total variation distance $\delta_1+\delta_2$ of a distribution $\mathcal{D}$ satisfying

$$\forall i\in[n],\quad \mathcal{D}(i)\geq(1-(\varepsilon_1+\varepsilon_2))\frac{|x_i|^p}{\|x\|_p^p}.$$

Proof. Our algorithm for $p>0$ is similar to the algorithm for $p=0$, hence we only explain the needed modifications. For $p\leq 1$ let $\beta=\varepsilon_2$ and for $p>1$ let $\beta=\frac{\varepsilon_2^p}{p^p}$. Let S be an ℓ_p-sampling algorithm with parameters ε_1 and δ_1, and let F be an F_p-moment estimation algorithm with parameters $\frac{\beta}{2}$ and $\frac{\delta_2\beta}{Cw\log Rw}$, for an appropriate constant C which will be chosen later. Similarly to the ℓ_0-sampling algorithm, our algorithm maintains $k=O\left(\beta^{-1}\log Rw\right)$ instances of algorithm S and k instances of algorithm F on different buckets.

The update procedure is almost the same as for $p=0$, except that we replace the ℓ_0-norm estimation algorithm by the F_p-moment estimation algorithm, and replace ε_2 by β in the inequality $F^j(B_j)>\left(1-\frac{\beta}{2}\right)F^i(B_i)$ on step $3.a$ of the sliding window algorithm for ℓ_0-sampling. The query procedure remains unchanged.

In an almost identical way, it can be shown that the number of instances is bounded by $O(\beta^{-1}\log Rw)$, i.e., for some sufficiently large constant C the number of instances is bounded by $\frac{C}{2}\beta^{-1}\log Rw$. Thus the different instances of F are invoked at most $C\beta^{-1}w\log Rw$ times. Therefore, by a union bound, with probability at most $\frac{Cw\log Rw}{\beta}\cdot\frac{\delta_2\beta}{Cw\log Rw}=\delta_2$, one of those $O(\beta^{-1}w\log Rw)$ invocation of algorithm F could return a wrong answer, in which case our algorithm could return anything.

The space complexity is clearly $O(\beta^{-1}\log Rw\left(s_1+s_2+\log w\right))$, where $s_1=s_1\left(\varepsilon_1,\delta_1,n\right)$ and $s_2=s_2\left(\frac{\beta}{2},\frac{\delta_2\beta}{Cw\log Rw},n\right)$, as we use exactly the same arguments as before, only replacing ε_2 by β.

Using the same proof, we deduce that for every $i<k$ we have either $|B_i|=|B_{i+1}|+1$ or $\|B_{i+1}\|_p^p>(1-\varepsilon_2)\|B_i\|_p^p$ (or both). Therefore, if B_1 is not exactly the active window W, then

$$\|B_1\|_p^p\geq\|W\|_p^p\geq\|B_2\|_p^p\geq(1-\varepsilon_2)\|B_1\|_p^p\geq(1-\varepsilon_2)\|W\|_p^p.$$

Let $x\in\mathbb{R}^n$ be the frequency-vector of the active window W, and $x^{B_1}\in\mathbb{R}^n$ be the frequency-vector of the bucket B_1. Recall that the probability that at least one of the $O(\beta^{-1}w\log Rw)$ invocations of F returns a wrong answer is at most δ_2. In addition, the output X^{alg} of the algorithm is just the output of running S on bucket B_1, which we know is within total variation distance δ_1 of a $(1+\varepsilon_1)$-ℓ_p-distribution. Together we get that $\mathrm{d_{TV}}\left(X^{alg}, \widetilde{X^{B_1}}\right) \leq \delta_1 + \delta_2$, for some random variable $\widetilde{X^{B_1}}$ that has a $(1+\varepsilon_1)$-ℓ_p-distribution of x^{B_1}. Let X^{B_1} be a random variable with an ℓ_p-distribution of x^{B_1}, then by Lemma 4.12 $\mathrm{d_{TV}}\left(\widetilde{X^{B_1}}, X^{B_1}\right) \leq \varepsilon_1$. Lastly, let X be a random variable with an ℓ_p-distribution of x, then according to Lemma 4.13 $\mathrm{d_{TV}}\left(X^{B_1}, X\right) \leq \varepsilon_2$. Overall, using the triangle inequality, we deduce the required bound $d_{TV}\left(X^{alg}, X\right) \leq \delta_1 + \delta_2 + \varepsilon_1 + \varepsilon_2$.

Furthermore, up to total variation distance $\delta_1 + \delta_2$ we can obtain a lower bound on the probability to sample coordinate $i \in [n]$. Recall that $\widetilde{X^{B_1}}$ is a random variable with a $(1+\varepsilon_1)$-ℓ_p-distribution of the vector x^{B_1}, and that $d_{TV}\left(X^{alg}, \widetilde{X^{B_1}}\right) \leq \delta_1 + \delta_2$. Notice that $x \leq x^{B_1}$, since B_1 contains the active window W, and that $\frac{1}{\|x^{B_1}\|_p^p} \geq \frac{1-\varepsilon_2}{\|x\|_p^p}$, thus

$$\Pr\left[\widetilde{X^{B_1}} = i\right] \geq (1-\varepsilon_1)\frac{\left|x_i^{B_1}\right|^p}{\|x^{B_1}\|_p^p} \geq (1-\varepsilon_1)\frac{(1-\varepsilon_2)\,|x_i|^p}{\|x\|_p^p} \geq (1-(\varepsilon_1+\varepsilon_2))\frac{|x_i|^p}{\|x\|_p^p}.$$

□

Kane et al [KNPW11] presented an F_p-moment estimation algorithm with space bound $O(\varepsilon^{-2}\log n \log \delta^{-1})$ for $p \in (0,2)$. Recently, Braverman et al [BCI$^+$17] showed that for streams with only positive increments the F_2-moment can be (ε,δ)-approximated using only $O(\varepsilon^{-2}\log\delta^{-1})$ bits of space. Their result, for the insertion-only variant, improved the celebrated result of Alon, Matias and Szegedy [AMS96], for the general turnstile variant.

Recently, Jayaram and Woodruff [JW18] showed an ℓ_p-sampling algorithm for every $0 < p \leq 2$ with accuracy parameter $\varepsilon = 0$, which means their algorithm's output $X \in [n]$ is within total variation distance δ of an ℓ_p-distribution of the frequency-vector. Their algorithm's space bound is $O(\log^2 n \log \delta^{-1})$ for $0 < p < 2$ and $O(\log^3 n \log^2 \delta^{-1})$ for $p = 2$.

Plugging these bounds into Theorem 4.14 we deduce the following corollary for $0 < p \leq 2$.

Corollary 4.15. *There is an algorithm for ℓ_p-sampling in the sliding-window model with space bound*

$$O(\varepsilon^{-1}\log Rw\left(\log^2 n \log\delta^{-1} + \varepsilon^{-2}\log n \log\tfrac{Rw}{\delta\varepsilon}\right)) = O(\varepsilon^{-3}\log^2 Rw \log^2 n \log\tfrac{1}{\delta\varepsilon}), \quad \textit{for } 0 < p \leq 1;$$

and

$$O(p^p\varepsilon^{-p}\log Rw\left(\log^2 n\log\delta^{-1} + \tfrac{p^{2p}}{\varepsilon^{2p}}\log n \log\tfrac{p^pRw}{\delta\varepsilon^p}\right)) = O(\varepsilon^{-6}\log^2 Rw\log^2 n\log\tfrac{1}{\delta\varepsilon}), \quad \textit{for } 1 < p < 2;$$

and

$$O(\varepsilon^{-2}\log Rw\left(\log^3 n \log^2\delta^{-1} + \tfrac{1}{\varepsilon^4}\log\tfrac{Rw}{\delta\varepsilon^2}\right)) = O(\varepsilon^{-6}\log^2 Rw\log^3 n\log^2\tfrac{1}{\delta\varepsilon}), \quad \textit{for } p = 2.$$

The algorithm's output $X \in [n]$ is within total variation distance $2\delta + 2\varepsilon$ of an ℓ_p-distribution of the frequency-vector of the active window W.

References

[Agg07] C. Aggarwal. *Data Streams: Models and Algorithms*, volume 31. 01 2007. doi:10.1007/978-0-387-47534-9.

[AKL17] S. Assadi, S. Khanna, and Y. Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1723–1742, 2017. Available from: http://dl.acm.org/citation.cfm?id=3039686.3039799.

[AKO11] A. Andoni, R. Krauthgamer, and K. Onak. Streaming algorithms via precision sampling. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 363–372, Oct 2011. doi:10.1109/FOCS.2011.82.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29. ACM, 1996. doi:10.1145/237814.237823.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16. ACM, 2002. doi:10.1145/543613.543615.

[BC15] V. Braverman and S. R. Chestnut. Universal sketches for the frequency negative moments and other decreasing streaming sums. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 591–605, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.591.

[BCI+17] V. Braverman, S. R. Chestnut, N. Ivkin, J. Nelson, Z. Wang, and D. P. Woodruff. Bptree: An ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 361–376. ACM, 2017. doi:10.1145/3034786.3034798.

[BDM02] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 633–634. Society for Industrial and Applied Mathematics, 2002. Available from: http://dl.acm.org/citation.cfm?id=545381.545465.

[BJKS04] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004. doi:10.1016/j.jcss.2003.11.006.

[Bł18] J. Błasiok. *Optimal streaming and tracking distinct elements with high probability*, pages 2432–2448. 2018. doi:10.1137/1.9781611975031.156.

[Blu90] N. Blum. A new approach to maximum matching in general graphs. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, ICALP '90, pages 586–597. Springer-Verlag, 1990. Available from: http://dl.acm.org/citation.cfm?id=646244.681623.

[BO07] V. Braverman and R. Ostrovsky. Smooth histograms for sliding windows. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 283–293. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.63.

[BOZ09] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 147–156. ACM, 2009. doi:10.1145/1559795.1559818.

[BS15] M. Bury and C. Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015*, pages 263–274. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48350-3_23.

[CCE$^+$16] R. Chitnis, G. Cormode, H. Esfandiari, M. Hajiaghayi, A. McGregor, M. Monemizadeh, and S. Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1326–1344. Society for Industrial and Applied Mathematics, 2016. Available from: http://dl.acm.org/citation.cfm?id=2884435.2884527.

[CCHM15] R. Chitnis, G. Cormode, M. Hajiaghayi, and M. Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1234–1251. Society for Industrial and Applied Mathematics, 2015. doi:10.1137/1.9781611973730.82.

[CF14] G. Cormode and D. Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014. doi:10.1007/s10619-013-7131-9.

[CJMM17] G. Cormode, H. Jowhari, M. Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:15, 2017. doi:10.4230/LIPIcs.ESA.2017.29.

[CK04] D. Coppersmith and R. Kumar. An improved data stream algorithm for frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 151–156. Society for Industrial and Applied Mathematics, 2004. Available from: http://dl.acm.org/citation.cfm?id=982792.982815.

[CKS03] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. pages 107– 117, August 2003. doi:10.1109/CCC.2003.1214414.

[CKX10] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, September 2010. doi:10.1016/j.tcs.2010.06.026.

[CMS13] M. S. Crouch, A. McGregor, and D. Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms – ESA 2013*, pages 337–348, 2013. doi:10.1007/978-3-642-40450-4_29.

[CS14] M. Crouch and D. M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 96–104. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

[DGIM02] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

[ETHL$^+$18] H. Esfandiari, M. T Hajiaghayi, V. Liaghat, M. Monemizadeh, and K. Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Trans. Algorithms*, 14(4):48:1–48:23, August 2018. Available from: http://doi.acm.org/10.1145/3230819, doi:10.1145/3230819.

[FJ15] U. Feige and S. Jozeph. Separation between estimation and approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 271–276. ACM, 2015. doi:10.1145/2688073.2688101.

[FKM$^+$05] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005. doi:10.1016/j.tcs.2005.09.013.

[Gan15] S. Ganguly. Taylor polynomial estimator for estimating frequency moments. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming*, pages 542–553. Springer Berlin Heidelberg, 2015.

[GL98] R. Grossi and E. Lodi. Simple planar graph partition into three forests. *Discrete Applied Mathematics*, 84(1):121 – 132, 1998. `doi:10.1016/S0166-218X(98)00007-9`.

[GL08] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 379–392. ACM, 2008. `doi:10.1145/1376616.1376657`.

[Ind99] P. Indyk. A small approximately min-wise independent family of hash functions. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 454–456. Society for Industrial and Applied Mathematics, 1999.

[Ind06] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, May 2006. `doi:10.1145/1147954.1147955`.

[IW05] P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 202–208. ACM, 2005. `doi:10.1145/1060590.1060621`.

[JST11] H. Jowhari, M. Sağlam, and G. Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58. ACM, 2011. `doi:10.1145/1989284.1989289`.

[JW18] R. Jayaram and D. P. Woodruff. Perfect ℓ_p sampling in a data stream. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 544–555, Oct 2018. `doi:10.1109/FOCS.2018.00058`.

[Kap13] M. Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1679–1697. Society for Industrial and Applied Mathematics, 2013. Available from: `http://dl.acm.org/citation.cfm?id=2627817.2627938`.

[KNPW11] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the Forty-third*

Annual ACM Symposium on Theory of Computing, STOC '11, pages 745–754. ACM, 2011. doi:10.1145/1993636.1993735.

[KNW10] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52. ACM, 2010. doi:10.1145/1807085.1807094.

[Li08] P. Li. On approximating frequency moments of data streams with skewed projections. *CoRR*, abs/0802.0802, 2008.

[LP17] D. A. Levin and Y. Peres. *Markov chains and mixing times.* American Mathematical Society, 2017.

[McG14] A. McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014. doi:10.1145/2627692.2627694.

[Mut05] S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005. doi:10.1561/0400000002.

[MV80] S. Micali and V. V. Vazirani. An $O(\sqrt{V}E)$ algoithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

[MV16] A. McGregor and S. Vorotnikova. Planar Matching in Streams Revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, volume 60 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.17.

[MV18] A. McGregor and S. Vorotnikova. A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:4. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.14.

[MW10] M. Monemizadeh and D. P. Woodruff. 1-pass relative-error L_p-sampling with applications. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1143–1160. Society for Industrial and Applied Mathematics, 2010. Available from: http://dl.acm.org/citation.cfm?id=1873601.1873693.

[RS78] Z. I. Ruzsa and E. Szemerédi. Triple systems with no six points carrying three triangles. *Combinatorica (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945, 1978.

[Vaz94] V. V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, Mar 1994. `doi:10.1007/BF01305952`.

[vH16] O. van Handel. Vertex cover approximation in data streams. Master's Thesis, 2016. Available from: `http://www.wisdom.weizmann.ac.il/~robi/files/OtnielVanHandel-MScThesis-2017_01.pdf`.