Thesis for the degree
Master of Science

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

עבודת גמר (תזה) לתואר
מוסמך למדעים

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
Otniel van Handel

מאת
עתניאל ון הנדל

אלגוריתמי קירוב לכיסוי בקודקודים בזרם נתונים
Vertex Cover Approximation in Data Streams

Advisor:
Prof. Robert Krauthgamer

מנחה:
פרופ' רוברט קראוטגמר

December 2016

דצמבר 2016

Vertex Cover Approximation in Data Streams

Abstract

We study the well known vertex cover problem in various data streaming settings. Our first result considers a special family of graphs, the *Vertex-Disjoint Paths* (VDP) family. We say that a graph G is in the VDP family if the edges of G are a union of vertex disjoint paths. For a VDP graph with n vertices given in a *dynamic stream*, in which both insertions and deletions are allowed, we provide a randomized algorithm for estimating the size of an optimal vertex cover within $(\frac{5}{4}+\varepsilon)$ factor, using $\widetilde{\mathcal{O}}(\varepsilon^{-1}\sqrt{n})$ bits of space. Our space bound is near-optimal because a result by Esfandiari et al. [EHL+15] implies that even when the input is limited to graphs in the VDP family, a weaker $(\frac{3}{2}-\varepsilon)$-estimation algorithm already requires $\Omega(\sqrt{n})$ space, for every constant $\varepsilon > 0$.

We continue by exploring general graphs in two other streaming models and develop algorithms for vertex cover approximation in these models. For the dynamic multi-pass model, in which both insertions and deletions are permitted and few passes over the stream are allowed, we provide a randomized p-pass algorithm that uses $\widetilde{\mathcal{O}}(n^{1+1/p})$ space and returns a feasible vertex cover at most twice the size of an optimal one. For the sliding window model, in which our interest is only in the last w elements in the stream, we provide a deterministic algorithm that uses $\mathcal{O}(\frac{1}{\varepsilon}n \lg n)$ bits of space and returns a vertex cover at most $(8+\varepsilon)$ times the size of an optimal one.

Acknowledgements

I would like to express my deep appreciation and sincere gratitude to my advisor, Prof. Robert Krauthgamer, for his advice and guidance throughout the process of this work. I would also like to thank Dr. Rajesh Chitnis for his collaboration thorough the past few months. He took a major part in developing Theorem 1.1. Finally, I would like to thank Yinon Nahum, Chen Attias, Inbal Rika, and many others for the intriguing and inspiring conversations we had.

Contents

1 Introduction

As the size of datasets and databases increase, computation over data streams is becoming increasingly essential in data processing. In the *data stream* model the goal is to find a solution while scanning the input in one sequential pass and using storage (space) which is sub-linear in the input size. Some questions can be easily answered in the data streaming model, such as counting the number of elements in a stream. Others cannot be answered exactly using sub-linear space [AMS96], in which case approximation and randomization play a major role in computing a solution.

We consider graph problems in the following streaming setting: the input is a graph $G = (V, E)$, where the set of vertices V is known in advance, and the edges E are revealed in a streaming manner, i.e. at time t some edge $e_t \in E$ is revealed in arbitrary order. This model is referred to as the *insertion-only* model. We also consider some variations of the streaming model, such as the *dynamic model* in which edge insertions and deletions are allowed (though an edge may be deleted only if it was previously inserted), the *sliding window* model in which we are only interested in a window of the last w edges that have arrived, and the *multi-pass* model where a small number of passes is allowed.

A *vertex cover* of a graph G is a subset of vertices $C \subseteq V$ such that each edge in E is incident to at least one vertex in C. A *minimum vertex cover* is an optimization problem where the goal is to find a vertex cover with minimum cardinality. For any optimization problem, one can relax the requirements by considering two variants: the *approximation* variant in which the goal is to report a feasible solution with near-optimal objective value, and the *estimation* variant in which the goal is to merely report a value close to the optimum, and there is no need to explicitly report a feasible solution. For example, in the approximation variant of vertex cover, the goal is to report a set of vertices $C' \subseteq V$ that is incident to all edges in E, and its size is close to the size of an optimal vertex cover. The goal in the estimation variant of vertex cover is to report a number $s \in \mathbb{N}$ close to the cardinality of a minimum vertex cover.

Evidently the approximation variant of a problem is harder than its estimation variant, because given a solution to the optimization variant, it is trivial to compute the value, i.e. the estimation variant. However, a solution to the estimation variant might not provide a solution to the approximation variant. In fact for the set cover problem with n elements and m sets, for any $\alpha \geq 1$, computing the approximation variant within an α multiplicative factor requires $\tilde{\Theta}(mn/\alpha)$ space, and computing the estimation variant within an α multiplicative factor can be done in only $\tilde{\Theta}(mn/\alpha^2)$ space[AKL16].

By convention, a randomized algorithm can toss coins and report a correct output with probability at least 3/4. Amplifying the success probability to high probability can easily be done using an extra $\lg n$ factor of space by standard repetitions. For maximum problems, we say $\widehat{OPT}$ α-approximates OPT if $\frac{1}{\alpha}OPT \leq \widehat{OPT} \leq OPT$. For minimization problems we say that $\widehat{OPT}$ α-approximates OPT if $OPT \leq \widehat{OPT} \leq \alpha \cdot OPT$. Notice that in both cases $\alpha \geq 1$.

1.1 Our contribution

The vertex cover problem in the standard computational model is NP-hard to solve exactly [Kar72]. However the *greedy algorithm* is a linear (in the number of edges) time

2-approximation algorithm. The greedy algorithm scans the edges and for each uncovered edge (u, v) encountered it adds both u and v to the cover. In the insertion-only streaming model, the greedy algorithm can easily be implemented using $\mathcal{O}(n)$ bit of space, one bit for each vertex to mark if it is in the cover, where throughout $n = |V|$. It is therefore natural to use 2-approximation and linear space as a baseline for the vertex cover problem, and to explore the trade-off between approximation and space compared to this baseline.

We explore the approximation-space trade-off, starting with algorithms that use space sub-linear in n. It seems unlikely that a constant factor approximation algorithm exists that uses sub-linear space since it would not even have space to store the solution. Therefore we relax the requirement and consider the estimation variant. Vertex cover estimation algorithms for bounded-arboricity graphs were studied extensively (see Section 1.2), yet no sub-linear space algorithm is known to produce an estimation factor better than 2, even for graphs with arboricity 1, i.e. trees. To obtain an approximation factor better than 2, we therefore consider the following more restricted family of graphs. We say a graph $G = (V, E)$ is in the *vertex disjoint paths* family (VDP for short) if G is a union of vertex disjoint paths. The following theorem is a result of joint work with Rajesh Chitnis.

Theorem 1.1. *For every $\varepsilon > 0$, there exists a randomized $(\frac{5}{4} + \varepsilon)$-estimation algorithm for minimum vertex cover of an input graph in VDP that uses $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon}\sqrt{n})$ space, even in the dynamic streaming model.*

Our space bound is near-optimal because a result by Esfandiari et al. [EHL$^+$15] for maximum matching implies that even for the VDP family, a weaker $(\frac{3}{2} - \varepsilon)$-estimation algorithm for minimum vertex cover already requires $\Omega(\sqrt{n})$ space, for every constant $\varepsilon > 0$. Furthermore, Theorem 1.1 implies that improving over the space lower bounds of Esfandiari et al. requires using a different family of graphs. Our proof of Theorem 1.1 shows that estimating the number of length 2 paths, and the number of vertices with degree 1 and 2, suffices to obtain a $5/4 + \varepsilon$ estimation for vertex cover.

We continue by exploring two other variants of the streaming model for general graphs (not only VDP graphs). For the dynamic model, we provide a 2-approximation multi-pass algorithm for vertex cover.

Theorem 1.2. *There exists a randomized algorithm for vertex cover in the dynamic streaming model that achieves 2-approximation, performs p passes, and uses $\mathcal{O}(n^{1+1/p} \log^5 n)$ bits of space.*

The algorithm is similar in spirit to an algorithm presented by Demaine, Indyk, Mahabadi and Vakilian [DIMV14] for the streaming set cover problem with m sets and n elements. They present a $O(4^{1/\delta})$-algorithm for the set cover problem that performs $O(4^{1/\delta})$ passes and uses $\widetilde{\mathcal{O}}(m \cdot n^{\delta})$ bits of space.

For the sliding window model we show the following.

Theorem 1.3. *For every $\varepsilon > 0$, there exists a single pass $(8 + \varepsilon)$-approximation deterministic algorithm for vertex cover in the sliding window streaming model that uses $\mathcal{O}(\frac{1}{\varepsilon} n \lg n)$ bits of space.*

This algorithm is based on a binning technique introduced by Crouch, McGregor and Stubbs [CMS13] for maximum matching in the sliding window model. Note that

model	approximation	space	reference
insertion	$5c+9$	$\widetilde{\mathcal{O}}(cn^{2/3})$	[EHL+15]
insertion	$c+2$	$\widetilde{\mathcal{O}}(cn^{2/3})$	[MV16b]
insertion	$(2c+1)(2c+2)$	$\widetilde{\mathcal{O}}(c^{2.5}\sqrt{n})$	[CJMM16]
insertion	$(22.5c+6)(1+\varepsilon)$	$\widetilde{\mathcal{O}}(\varepsilon^{-2}c\log^2 n)$	[CJMM16]
insetion	$(c+2)(1+\varepsilon)$	$\mathcal{O}(\varepsilon^{-2}\lg n)$	[MV16a]
insertion	$3/2-\varepsilon$	$\Omega(\sqrt{n})$	[EHL+15]
dynamic	$\mathcal{O}(c)$	$\widetilde{\mathcal{O}}(cn^{4/5})$	[BS15, CCE+16]
dynamic	$(2c+1)(2c+2)$	$\widetilde{\mathcal{O}}(c^{10/3}n^{2/3})$	[CJMM16]

Table 1: Known results for estimating Maximum Matching in graphs with arboricity at most c.

for estimation problems, an algorithm for α-estimating maximum matching implies an algorithm for 2α-estimating minimum vertex cover, because a minimum vertex cover is at least the size of a maximum matching, and at most twice its size. In bipartite graphs, by König's Theorem the size of the maximum matching and minimum vertex cover are equal. However an approximation algorithm for maximum matching is not directly useful for an approximation algorithm for minimum vertex cover.

1.2 Related work

We continue two related but different lines of work. The first line of work is maximum matching estimation for graphs with arboricity at most c. For the insertion-only model, after some work (see Table 1), a remarkable logarithmic space algorithm was presented that achieves a $(22.5c+6)(1+\varepsilon)$-estimation [CJMM16]. Recently, this result was improved to a $(c+2)(1+\varepsilon)$-estimation still using logarithmic space [MV16a]. For the dynamic model, there is no sub-polynomial space algorithm known, and the best algorithm known achieves $(2c+1)(2c+2)$-estimation using $\widetilde{\mathcal{O}}(c^{10/3}n^{2/3})$ space [CJMM16]. For more results, see Table 1.

The second line of work is about different streaming models, such as the dynamic model and sliding window model. For maximum matching approximation in the dynamic model, Assadi et al. [AKLY16] present an n^{ε}-approximation algorithm using $\widetilde{\mathcal{O}}(n^{2-3\varepsilon})$ space, and prove matching lower bounds (up to log factors). For maximum matching approximation in the sliding window model, Crouch, McGregor and Stubbs [CMS13] provide a $(3+\varepsilon)$-approximation algorithm using $\mathcal{O}(\varepsilon^{-1}n\lg^2 n)$ bits of space.

1.3 Preliminaries

We extensively use the following version of Hoeffding's bound.

Theorem 1.4. *(Hoeffding) Given k independent random variables $X_i \in [0,1]$, and $X = \sum_{i=1}^{k} X_i$, then $\Pr[|X - \mathbb{E}[X]| \geq \varepsilon k] \leq 2e^{-2\varepsilon^2 k}$.*

ℓ_0-sampling is defined as follows. Consider an input vector $\mathbf{x} \in \mathbb{R}^n$ given in a streaming manner, where at each time step an update (increment or decrement) to coordinate $\mathbf{x}_i$ is given. The goal is to sample a coordinate i of $\mathbf{x}$, namely, return i and the coordinate $\mathbf{x}_i$

(at the end of the stream), with uniform probability over all non-zero coordinates $\mathbf{x}_i \neq 0$. We refer to such an algorithm as an ℓ_0-sampler.

Lemma 1.5 (ℓ_0-sampler [CF14, Corollary 1])**.** *There is an ℓ_0-sampler that uses $\mathcal{O}(\log^4 n)$ bits, succeeds with probability at least $1 - n^{-c}$ and, conditioned on a successful recovery, outputs a coordinate i with non-zero $\mathbf{x}_i$ with probability $\frac{1}{\|\mathbf{x}\|_0} \pm n^{-c}$ for arbitrarily large constant c.*

The above Lemma can be easily adapted to sampling an edge in a dynamic graph, by representing the edges in the graph as a binary vector $\mathbf{x} \in \{0,1\}^{\binom{n}{2}}$. Each time an edge is added (or deleted) add (or subtract) 1 from the respective coordinate $\mathbf{x}_i$. By using multiple ℓ_0 samplers, one can recover all edges in the final graph (at the end of a dynamic stream).

Lemma 1.6. *There is a randomized algorithm that recovers all edges in the final graph (at the end of a dynamic graph stream), using $\mathcal{O}(m \lg^5 n)$ bits of space, where m is the number of edges in the final graph.*

Proof. Consider the algorithm that throughout the stream maintains $10m \lg n$ independent ℓ_0-samplers. At the end of the stream the algorithm extracts an edge from each sampler, and returns the union (removing duplicates) of all extracted edges.

Now consider an edge e and a sampler l, the probability that l was successful and that e was extracted from l is by a union bound at least $\frac{1}{m} - 2n^{-c} \geq \frac{1}{2m}$, for large enough constant c. Thus the probability it was not extracted from $10m \lg n$ independent ℓ_0-samplers is $(1 - \frac{1}{2m})^{10m \lg n} \leq n^{-5}$. By a union bound on all $m \leq n^2$ edges in the stream, the probability that some edge was not recovered is $\leq n^{-3}$. In other words, the probability that all edges in the stream were recovered is at least $1 - 1/n^3$. □

2 Estimating Vertex Cover in Paths

In this section we explore the approximation-space trade-off for minimum vertex cover by considering algorithms using sub-linear space. It seems unlikely that a constant approximation sub-linear (in n) space algorithm exists, since it would not even have enough space to store the entire solution. Therefore we relax the requirement and consider the estimation variant. Even for bounded arboricity graphs, no sub-linear space algorithm is known to produce an estimation factor better than 2. (See section 1.2.) Therefore we consider a more restricted family of graphs, the *vertex disjoint paths* family, denoted VDP.

By König's Theorem, in every bipartite graph, the size of the maximum matching is equal the size of the minimum vertex cover [BM76, Chapter 5]. Since the VDP family is bipartite, for graphs in this family every result for maximum matching estimation is true also for minimum vertex cover estimation. Esfandiari et al. [EHL+15] proved a lower bound that any streaming algorithm, even for VDP graphs, estimating the maximum matching (and therefore also minimum vertex cover) within $\frac{3}{2} - \varepsilon$ factor, requires at least $\Omega(\sqrt{n})$ space, even in the insertion only model, for every constant $\varepsilon > 0$. We restate the theorem from the Introduction that provides an algorithm that uses nearly matching space.

Theorem 1.1. *For every $\varepsilon > 0$, there exists a randomized $(\frac{5}{4}+\varepsilon)$-estimation algorithm for minimum vertex cover of an input graph in VDP that uses $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon}\sqrt{n})$ space, even in the dynamic streaming model.*

The rest of this section is dedicated to proving Theorem 1.1. Since the results are the same for maximum matching and minimum vertex cover, the analysis is done via maximum matching. We start by showing that with a couple of simple and "local" counters one can estimate the maximum matching. Later we show how to estimate these counters in the dynamic streaming model.

2.1 Graph properties

Let p_i be the number of paths of length i, i.e. containing i edges, and let d_i be the number of vertices of degree i. Counting the number of matched edges in each path, it is easy to verify that for each path of length 1 or 2 an optimal matching has one matched edge, for each path of length 3 or 4 it has 2 matched edges, and so on. In general the size of an optimal solution is

$$OPT = p_1 + p_2 + 2p_3 + 2p_4 + ... + \left\lceil \frac{n-1}{2} \right\rceil p_{n-1} = \sum_{i=1}^{n-1} \left\lceil \frac{i}{2} \right\rceil p_i. \tag{1}$$

We can express the number of degree 2 vertices, denoted d_2, as a function of the number of paths. Notice that in a path of length 1, there are no vertices of degree 2, in a path of length 2 there is one vertex of degree 2, in a path of length 3 there are 2, and so on. In general,

$$d_2 = p_2 + 2p_3 + 3p_4 + ... + (n-2)p_{n-1} = \sum_{i=1}^{n-1}(i-1)p_i.$$

Consider the following estimator for OPT using only d_1, d_2 and p_2

$$\begin{aligned}\widetilde{OPT} &= \frac{2}{5}d_1 + \frac{2}{5}d_2 - \frac{1}{5}p_2 \\ &= \frac{1}{5}\left(2\sum_{i=1}^{n-1} 2p_i + 2\sum_{i=1}^{n-1}(i-1)p_i - p_2\right) \\ &= \frac{2}{5}\sum_{i=1}^{n-1}(i+1)p_i - \frac{1}{5}p_2. \end{aligned} \tag{2}$$

Let $\alpha_i = \lceil i/2 \rceil$ be the coefficient of p_i in OPT according to Equation (1), and let $\tilde{\alpha}_i$ be the coefficient of p_i in $\widetilde{OPT}$ as in Equation (2). Then the (deterministic) error of the estimator is

$$OPT - \widetilde{OPT} = \sum_{i=1}^{n-1}(\alpha_i - \tilde{\alpha}_i)p_i.$$

For $i = 1$ the error is bounded by $\alpha_1 - \tilde{\alpha}_1 \leq \frac{1}{5}\alpha_1$, for $i = 2$ the error is $\alpha_2 - \tilde{\alpha}_2 = 0$, and for $i \geq 3$ there are two cases: even i and odd i. For even $i \geq 4$, the error is bounded by

$$\alpha_i - \tilde{\alpha}_i = \frac{1}{2}i - \frac{2}{5}(i+1) = \frac{1}{10}i - \frac{2}{5} \leq \frac{1}{5}\cdot\frac{1}{2}i \leq \frac{1}{5}\left\lceil \frac{i}{2} \right\rceil = \frac{1}{5}\alpha_i.$$

For odd $i \geq 3$, the error is bounded by

$$\alpha_i - \tilde{\alpha}_i = \frac{i+1}{2} - \frac{2}{5}(i+1) = \frac{1}{10}(i+1) = \frac{1}{5}\left\lceil \frac{i}{2} \right\rceil = \frac{1}{5}\alpha_i.$$

Thus $\alpha_i - \tilde{\alpha}_i \leq \frac{1}{5}\alpha_i$ for all i, therefore

$$OPT - \widetilde{OPT} = \sum_{i=1}^{n-1}(\alpha_i - \tilde{\alpha}_i)p_i \leq \frac{1}{5}\sum_{i=1}^{n-1}\alpha_i p_i = \frac{1}{5}OPT.$$

We continue by showing that $0 \leq OPT - \widetilde{OPT}$. For $i \in \{1,2,3\}$ it can be verified manually that $0 \leq \alpha_i - \tilde{\alpha}_i$, and for $i \geq 4$ it holds that

$$0 \leq \left\lceil \frac{i}{2} \right\rceil - \frac{2}{5}(i+1) = \alpha_i - \tilde{\alpha}_i$$

and therefore

$$0 \leq \sum_{i=1}^{n-1}(\alpha_i - \tilde{\alpha}_i)p_i = OPT - \widetilde{OPT}.$$

Altogether, $\frac{4}{5}OPT \leq \widetilde{OPT} \leq OPT$, concluding that $\widetilde{OPT}$ is a 5/4-estimator for OPT.

2.2 Streaming Algorithm

We are now ready to present our algorithm for estimating $\widetilde{OPT}$ in the dynamic streaming model. Throughout we assume that $\varepsilon = \Omega\left(\frac{1}{n^{1/2}}\log^{\mathcal{O}(1)}(n)\right)$, since otherwise $\widetilde{\mathcal{O}}(\varepsilon^{-1}\sqrt{n})$ suffices to store all the edges of G. If there are at most $10\varepsilon^{-1}\sqrt{n}\lg n$ non-deleted edges remaining at the end of the stream, then by Lemma 1.6 we can recover all the edges with high probability using $\widetilde{\mathcal{O}}(\varepsilon^{-1}\sqrt{n})$ bits of storage. Therefore we assume throughout that the number of edges is greater than $10\varepsilon^{-1}\sqrt{n}\lg n$. The input to the algorithm is a set of vertices V, an error parameter ε, and an estimated number of edges t (it's enough that $|E| \leq t \leq 2|E|$). Let $k = 10\frac{1}{\varepsilon}\sqrt{n}\lg n$. Before the stream starts, sample each vertex in V with probability k/t, and denote the set of sampled vertices by V_R. For each edge (u,v) inserted to (deleted from) the stream, if u or v are in V_R, add the edge (u,v) to E_R (remove it for deletion). Let $\hat{d}_1$ be the number of vertices in V_R of degree 1 in the graph (V, E_R), and similarly let $\hat{d}_2$ be the number of vertices in V_R of degree 2 in the graph (V, E_R). Finally, let $\hat{p}_2$ be the number of length 2 paths in (V, E_R) where both endpoints of the path are in V_R. Return the estimate $\widehat{OPT} = \frac{2}{5}\frac{t}{k}\hat{d}_1 + \frac{2}{5}\frac{t}{k}\hat{d}_2 - \frac{1}{5}\frac{t^2}{k^2}\hat{p}_2$. For more details see Algorithm 1.

The correctness of the algorithm is derived by showing that d_1, d_2 and p_2 are estimated well by appropriately scaling of $\hat{d}_1, \hat{d}_2$ and $\hat{p}_2$. For every sampled vertex $v \in V_R$ the algorithm stores all its incident edges, therefore the degree of v in the original graph G is equal to its degree in the sampled graph (V, E_R), and therefore $\deg_G(v) = \deg_{(V,E_R)}(v)$ for every vertex $v \in V_R$. (Note that this is not true for vertices in $V \setminus V_R$.)

Lemma 2.1. $\Pr[\frac{t}{k}\hat{d}_i \in (d_i \pm \varepsilon t)] \geq 1 - 1/n$, *for* $i \in \{1,2\}$.

Algorithm 1 Estimating $\widetilde{OPT}$

1: **procedure** EstOPT(V, ε, t)
2: $n = |V|, k = 10\frac{1}{\varepsilon}\sqrt{n}\lg n$
3: Let V_R be obtained by sampling each vertex in V with probability k/t.
4: $E_R = \phi$
5: **for** $e = \{u, v\}$ inserted to (deleted from) stream **do**
6: **if** $v \in V_R$ or $u \in V_R$ **then** $E_R = E_R \cup \{e\}$ ($E_R = E_R \setminus \{e\}$ for deletion)
7: Let $\hat{d}_1$ be the number of vertices in V_R of degree 1 in (V, E_R).
8: Let $\hat{d}_2$ be the number of vertices in V_R of degree 2 in (V, E_R).
9: Let $\hat{p}_2$ be the number of ordered pairs (u, w) such that:

a: $u, w \in V_R$

b: there exists $v \in V$ such that $\{u, v\}, \{v, w\} \in E_R$

c: $\deg_{(V,E_R)}(u) = \deg_{(V,E_R)}(v) = 1$, i.e. u and v are endpoints of a path

d: $u < w$ in some fixed ordering.

10: $\widehat{OPT} = \frac{2}{5}\frac{t}{k}\hat{d}_1 + \frac{2}{5}\frac{t}{k}\hat{d}_2 - \frac{1}{5}\frac{t^2}{k^2}\hat{p}_2$
11: **return** $\widehat{OPT}$

Proof. Fix $i \in \{1, 2\}$. To simplify notation, suppose $\{v \in V : \deg_G(v) = i\} = \{v_1, ..., v_{d_i}\}$. For $j \in [d_i]$, define $X_j = 1$ if $v_j \in V_R$ and $X_j = 0$ otherwise, and define $X = \sum_{j=1}^{d_i} X_j$. Each vertex is sampled independently with probability k/t, therefore $\Pr[X_j = 1] = k/t$, and from linearity of expectation

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=1}^{d_i} X_j\right] = d_i k/t.$$

Since the X_js are sampled independently and bounded, using Hoeffding's inequality we obtain that

$$\Pr\left[\left|\frac{t}{k}\hat{d}_i - d_i\right| \geq \varepsilon t\right] = \Pr\left[|X - \mathbb{E}[X]| \geq \varepsilon k\right] \leq 2e^{-2(\frac{\varepsilon}{d_i}k)^2 d_i} = 2e^{-2\frac{\varepsilon^2}{d_i}k^2} \leq \frac{1}{n},$$

where the last inequality is because $d_i \leq n$ and $k^2 = 100\frac{1}{\varepsilon^2}n\lg^2 n$. □

Lemma 2.2. $\Pr[(t/k)^2\hat{p}_2 \in (p_2 \pm \varepsilon t)] \geq 0.99$.

Proof. Let $P_2 \subseteq V \times V$ be the set of endpoints, restricted to $u < w$ for every $(u, w) \in V \times V$, of maximal length 2 paths in G, i.e. not including sub-paths of a longer path. For every $u, w \in V$, let $Y_{u,w}$ be an indicator random variable, such that $Y_{u,w} = 1$ if $(u, w) \in P_2$ and $u, w \in V_R$, i.e. u and w are endpoints of a path of length 2 and sampled by the algorithm, and $Y_{u,w} = 0$ otherwise. Let $Y = \sum_{u<w} Y_{u,w} = \sum_{u,w\in P_2} Y_{u,w}$ be a random variable counting the number of length 2 paths in the sample with both endpoints in V_R. If $(u, w) \in P_2$ then $\Pr[Y_{u,w} = 1] = (k/t)^2$. Note that if $(u, w) \notin P_2$ then $\Pr[Y_{u,w} = 1] = 0$. There are only $p_2 = |P_2|$ such paths, therefore from linearity of expectation $\mathbb{E}[Y] = (k/t)^2 p_2$. Observe that the random variable Y represents exactly $\hat{p}_2$ from step 9 in the algorithm.

It is left to bound the probability that Y is far from its expectation. We do this by bounding the variance, and then applying Chebyshev's inequality. Since the random variables are independent, the variance is bounded by

$$\mathrm{Var}[Y] = \mathrm{Var}[\sum_{(u,w)\in P_2} Y_{u,w}] = \sum_{(u,w)\in P_2} \mathrm{Var}[Y_{u,w}] \leq \sum_{(u,w)\in P_2} \Pr[Y_{u,w}=1] = \left(\frac{k}{t}\right)^2 p_2.$$

Applying Chebyshev's inequality we obtain that

$$\Pr\left[\left|\left(\frac{t}{k}\right)^2 \hat{p}_2 - p_2\right| \geq \varepsilon t\right] = \Pr\left[|Y - \mathbb{E}[Y]| \geq \varepsilon\frac{k^2}{t}\right] \leq \frac{p_2\left(\frac{k}{t}\right)^2}{\left(\varepsilon\frac{k^2}{t}\right)^2} = \frac{p_2}{\varepsilon^2 k^2} \leq \frac{1}{100}$$

where the last inequality is because $p_2 \leq n$ and $k^2 = 100\varepsilon^{-2} n \lg^2 n$. □

Lemma 2.3. $\Pr[\widehat{OPT} \in (1\pm 4\varepsilon)\widetilde{OPT}] > 0.9$, *assuming* $|E| \leq t \leq 2|E|$.

Proof. By Lemma 2.1 with high probability $\frac{t}{k}\hat{d}_i \in (d_i \pm \varepsilon t)$, and by Lemma 2.2 with probability at least 0.99, $\frac{k^2}{t^2}\hat{p}_2 \in (p_2 \pm \varepsilon t)$. By a union bound on the failure probabilities,

$$\Pr\left[\widehat{OPT} = \frac{2}{5}\frac{t}{k}\hat{d}_1 + \frac{2}{5}\frac{t}{k}\hat{d}_2 - \frac{1}{5}\frac{t^2}{k^2}\hat{p}_2 \in \left(\widetilde{OPT} \pm \left(\frac{2}{5}+\frac{2}{5}+\frac{1}{5}\right)\varepsilon t\right)\right] \geq 0.9.$$

Assuming $|E| \leq t \leq 2|E|$, and since $\widetilde{OPT} \geq |E|/2$, therefore $\widetilde{OPT} \geq t/4$, and

$$\widetilde{OPT}(1-4\varepsilon) \leq \widehat{OPT} \pm \varepsilon t \leq \widetilde{OPT}(1+4\varepsilon),$$

concluding that

$$\Pr[\widehat{OPT} \in \widetilde{OPT}(1\pm 4\varepsilon)] \geq 0.9$$

and Lemma 2.3 follows. □

Space There are at most t edges in G. If there are $10\varepsilon^{-1}\sqrt{n}\lg n$ or less edges remaining at the end of the stream, all edges are extracted and the space used is $\widetilde{\mathcal{O}}(\varepsilon^{-1}\sqrt{n})$. Otherwise, the Estimating $\widetilde{OPT}$ algorithm is used. To sample an edge, at least one of its vertices must be sampled, and the probability to sample a vertex is k/t, therefore each edge is sampled with probability at most $2k/t$. Thus the expected number of edges in E_R is $\widetilde{\mathcal{O}}(t\cdot\frac{k}{t}) = \widetilde{\mathcal{O}}(k) = \widetilde{\mathcal{O}}(\frac{1}{\varepsilon}\sqrt{n})$. By Markov's inequality with probability at least 0.99 the space will not exceed its expectation by more than 100 times.

Putting it all together and scaling $\widehat{OPT}$ and ε appropriately, implies Theorem 1.1.

Optimality $\widehat{OPT}$ is optimal in the sense that any estimator using only the counters d_1, d_2, p_1 and p_2, cannot yield a better approximation. This is because these counters have the same value for a graph G_1 consisting of two paths of length 4 and a graph G_2 consisting of one path of length 3 and one path of length 5. But in G_1 the maximum matching is of size 4 and in G_2 it is 5. Consequently, any algorithm using only these counters, cannot do better than a (5/4)-approximation[1].

[1] To find the optimal coefficient for d_1, d_2, p_1 and p_2 we set up and solved a linear program.

Estimating the number of edges Since $|E| \leq t \leq |E|$, we can run $\lg n$ copies of the algorithm, where the ith copy will use parameter $t = 2^i$. Using $\mathcal{O}(\lg n)$ bits, the algorithm can count the number of edges in the stream by incrementing and decrementing a counter for each insertion and deletion, respectively. Finally, return the vertex cover obtained from the copy with $1 \leq \frac{t}{|E|} < 2$. Running $\lg n$ copies and a counter, only increases the space by a multiplicative $\lg n$ factor.

3 Multi-Pass Algorithm in the Dynamic Model

In this section we explore vertex cover approximation for dynamic streams. At the time of writing, the author is not aware of any sub-linear (in the number of edges) space one-pass algorithm for vertex cover approximation in the dynamic model, therefore we relax the requirements to allow multiple passes. We prove Theorem 1.2 by presenting a randomized algorithm that performs p passes on the data stream, uses $\widetilde{\mathcal{O}}(n^{1+1/p})$ space, and returns a 2-approximation.

Before presenting the algorithm, we present some definitions and known results, which are used as building blocks for our algorithm. The *greedy algorithm* iterates over the edges (in an arbitrary order), and for each edge (u, v) visited, the algorithm adds both u and v to the cover if the edge is not covered yet. It is known that the greedy algorithm is a 2-approximation algorithm for vertex cover. We say that a subset of the vertices is an *ε-cover*, if it covers at least a $(1 - \varepsilon)$ fraction of the edges. Given a partial cover $P \subseteq V$ it is useful to extend P by adding more vertices. Let $E_P = E \setminus (P \times V)$ be the set of edges not covered by P. We say that a subset of vertices is an *ε-cover of E_P* if it covers $(1 - \varepsilon)|E_P|$ edges of E_P. In the special case when P is the empty set, an ε-cover of E_P is equivalent to an ε-cover.

We only consider the case when the number of passes permited is small, i.e. $p < n$, because when the number of passes is large, i.e. $p \geq n$, the following simple randomized algorithm will return a 2-approximation vertex cover. In each pass, the algorithm samples an uncovered edge using an ℓ_0-sampler, and adds both its endpoints to the current cover. The algorithm stops when all edges in the stream are covered, i.e. when the ℓ_0-sampler fails. The algorithm performs at most n passes, because at each pass it adds two new vertices to the cover, and if it would perform more than n passes, it would have a vertex cover of size $2n$ when there are only n vertices in the graph. This algorithm is equivalent to the greedy algorithm when the order of edges depends on the samplers, and therefore is a 2-approximation. The algorithm uses $\mathcal{O}(n)$ bits to keep track of the vertex cover, and an additional $\mathcal{O}(\lg^4 n)$ bits for the ℓ_0-sampler (by Lemma 1.5), which can be reused over passes. Thus the algorithm uses $\mathcal{O}(n)$ bits of space.

We start by introducing an algorithm for an ε-cover of E_P, which will later be used to construct an algorithm for obtaining a (full) vertex cover. The following randomized *Partial-Cover* algorithm returns an ε-cover of E_P while performing a single pass over the data stream. The input to the algorithm is: the set of vertices V, a partial cover P, and a number s of ℓ_0-samplers. While reading the sequence of edge updates in a streaming manner, the algorithm maintains s independent copies of an ℓ_0-sampler for the edges in E_P, by ignoring edges already covered by P. At the end of the stream, the algorithm extracts s edges from the ℓ_0-samplers, and denotes them $e_1, ..., e_s$. It then runs the greedy

algorithm on the graph $(V, \{e_1, ..., e_s\})$, and returns the resulting vertex cover denoted $\widetilde{P}$. For details see Algorithm 2. The correctness of the algorithm is asserted by the following Lemma.

Algorithm 2 Partial-Cover

```
1: procedure PartialCover(V, P, s)
2:     for i ∈ [s] do
3:         L[i] = initialize independent copy of ℓ0-sampler
4:     for addition/deletion of (u, v) in stream do
5:         if u ∉ P and v ∉ P then
6:             for i ∈ [s] do add/delete (u, v) from L[i]
7:     for i ∈ [s] do extract an edge e_i from L[i]
8:     P̃ = Greedy-Algorithm(V, {e_1, ..., e_s})
9:     return P̃
```

Lemma 3.1. *Let $s = 4n/\varepsilon$. Then algorithm Partial-Cover performs a single pass over the stream, and with probability $\geq 1/2$ returns an ε-cover of E_P.*

Proof. Assume to the contrary, that with probability $\geq 1/2$ Partial-Cover does not return an ε-cover of E_P, i.e. after the algorithm terminates there are at least ε fraction of E_P edges not covered by $\widetilde{P}$. Denote by Z the event that the algorithm does not return an ε-cover of $|E_P|$, thus $\Pr[Z] \geq 1/2$. Let X_i be the indicator random variable that when the greedy algorithm inspects the random edge $e_i \in E_P$ it is not covered yet by the greedy algorithm, and therefore the greedy algorithm will add both its endpoints to $\widetilde{P}$. Finally denote $X = \sum_{i=1}^{s} X_i$.

For each uncovered edge encountered by the greedy algorithm, it adds 2 vertices to the cover, therefore $2X = |\widetilde{P}| \leq n$ (with probability 1), and therefore $\mathbb{E}[X] \leq n/2$. By conditional probability, the expectation of X is

$$\begin{aligned}
\mathbb{E}[X] &= \Pr[Z] \cdot \mathbb{E}[X|Z] + \Pr[\bar{Z}] \cdot \mathbb{E}[X|\bar{Z}] \\
&\geq \Pr[Z] \cdot \mathbb{E}[X|Z] \\
&\geq \frac{1}{2} \cdot \sum_{i=1}^{s} \Pr[X_i = 1] \\
&= \frac{1}{2} \cdot s \cdot \varepsilon \\
&\geq 2n.
\end{aligned}$$

Thus $2n \leq \mathbb{E}[X] \leq n/2$, which is a contradiction. □

By increasing s we could significantly boost the probability. This is formalized in the following lemma.

Lemma 3.2. *For every $\varepsilon > 0$, let $s = \frac{16}{\varepsilon} n \lg n$. Then with probability at least $1 - 1/n^4$, the algorithm Partial-Cover returns an ε-cover of E_P .*

Proof. Partial-Cover samples edges and runs the greedy algorithm on them. Lemma 3.1 implies that running the greedy algorithm on a sample of $4n/\varepsilon$ edges, will return an ε-cover of E_P with probability at least $1/2$.

Partition the sampled edges $e_1, ..., e_s$ to $4 \lg n$ sets of size $4n/\varepsilon$ each. It is enough to successfully run the greedy algorithm on one of these sets, because that will ensure an ε-cover of E_p. Thus the algorithm fails only when it fails on all sets. Because the samples are independent, the probability to fail on all sets is at most $(1/2)^{4 \lg n} = 1/n^4$. □

We are now ready to present the Full-Cover algorithm, a p-pass randomized algorithm that covers all edges in E. The algorithm gets as input the set of vertices V, and the number of passes p. Let $\varepsilon = n^{-1/p}$. The idea is to construct a feasible vertex cover by iteratively covering more edges in each pass. During the kth pass, the algorithm uses Partial-Cover to obtain an ε-cover of the edges not covered by the previous partial cover $\widehat{P}_{k-1}$, and sets $\widehat{P}_k$ to be the union of the vertex set obtained by Partial-Cover and $\widehat{P}_{k-1}$. Finally it returns $\widehat{P}_p$. For details see Algorithm 3.

Algorithm 3 Full-Cover

1: **procedure** FULLCOVER(V, p)
2: Let $\varepsilon = n^{-\frac{1}{p}}$.
3: $\widehat{P}_0 = \phi$
4: **for** $k \in [p]$ passes **do**
5: $\widehat{P}_k = \widehat{P}_{k-1} \cup$ Partial-Cover$(V, \widehat{P}_k, \frac{16}{\varepsilon} n \lg n)$
6: **return** $\widehat{P}_p$

Lemma 3.3. *With high probability, Full-Cover returns a vertex cover.*

Proof. We say a pass k is successful if Partial-Cover successfully returns an ε-cover of $E_{\widehat{P}_{k-1}}$. In each successful pass the number of uncovered edges is reduced by a fraction of ε. The number of edges is at most n^2, and after $p-1$ successful passes at most $\varepsilon^{p-1} n^2 = n^{2-\frac{p-1}{p}} = n^{1+\frac{1}{p}}$ edges remain uncovered. At the last pass, by Lemma 1.6 with probability at least $1 - \frac{1}{n^4}$ all uncovered edges are sampled by Partial-Cover, therefore it will cover all remaining uncovered edges. By a union bound on the failure probability, the probability of all passes succeeding is at least $1 - (p-1)\frac{1}{n^4} - \frac{1}{n^4} > 1 - \frac{1}{n^3}$, for $p < n$. □

Since at pass k we only use the information from pass $k-1$, the space could be reused over passes. The space used is $\mathcal{O}(n)$ bits to store the information for $\widehat{P}_k$ and $\widehat{P}_{k-1}$ (one bit for each vertex), and an additional $\mathcal{O}(\frac{1}{\varepsilon} n^{1+1/p} \lg^5 n)$ bits to store $\mathcal{O}(\frac{1}{\varepsilon} n^{1+1/p} \lg n)$ ℓ_0-samplers, each using $\mathcal{O}(\lg^4 n)$ bits (by Lemma 1.5). Therefore the total space used is $\mathcal{O}(\frac{1}{\varepsilon} n^{1+1/p} \log^5 n)$ bits.

Since each time the greedy algorithm adds a pair of vertices u and v, it is because it visited an uncovered edge (u, v), therefore either u or v must be in an optimal cover. Therefore the algorithm returns a vertex cover of size at most twice an optimal solution, thus it is a 2-approximation. (A similar argument is used to prove that the greedy algorithm is a 2-approximation.) This completes the proof of the following Theorem.

Theorem 1.2. *There exists a randomized algorithm for vertex cover in the dynamic streaming model that achieves 2-approximation, performs p passes, and uses* $\mathcal{O}(n^{1+1/p}\log^5 n)$ *bits of space.*

4 Sliding Window Model

In this section we explore another streaming model, the sliding window model. In this model the goal is to compute the vertex cover of the graph that has arrived in the last time window. We refer to $e_{t-(w-1)}, e_{t-(w-2),\ldots,e_t}$ as the *active* window where w is the length of the active window. Specifically, given a stream of edges $e_1, e_2, \ldots$, at time t we need to return a vertex cover for edges $e_{t-(w-1)}, e_{t-(w-2),\ldots,e_t}$, where t is not known in advance. The motivation is that by ignoring data prior to the active window, we focus on the "freshest" data and can therefore detect anomalies and trends more quickly.

We prove Theorem 1.3 by showing a single pass deterministic $(8+\varepsilon)$-approximation algorithm for vertex cover in the sliding window model, using $\mathcal{O}(\frac{1}{\varepsilon}n\lg n)$ space. If the window is small, i.e. $w \leq \frac{1}{\varepsilon}n\lg n$, a trivial solution is to store the entire window, therefore we consider the case where $w > \frac{1}{\varepsilon}n\lg n$. Some graph problems were already studied in the semi-streaming sliding window model, especially a $(3+\varepsilon)$-approximation algorithm was presented by Crouch, McGregor and Stubbs [CMS13] for maximum matching. We use some of their ideas for our vertex cover algorithm.

We begin with some notation. For graph $G=(V,E)$, we say a subset of edges $A \subseteq E$ is a *segment*, if the edges of A appear sequentially in the stream. Let $\mathrm{VC}^*(A)$ and $\widetilde{\mathrm{VC}}(A)$ be the size of an optimal and greedy solutions on segment A, respectively. Denote the concatenation of disjoint segments A and B by AB. Intuitively, if the greedy algorithm of a smaller input segment B has similar cardinality as on a larger inputs segment AB, then for any successive segment C the greedy algorithm on the shorter input segment BC has similar cardinality as on the larger input segment ABC. This idea is formalized in the following Lemma.

Lemma 4.1. *For disjoint segments of the stream* A, B *and* C *and for every* $\varepsilon \in (0, 1/2)$*, if* $\widetilde{\mathrm{VC}}(B) \geq (1-\varepsilon)\widetilde{\mathrm{VC}}(AB)$ *then* $\widetilde{\mathrm{VC}}(BC) \geq \frac{1}{4}(1-\varepsilon)\widetilde{\mathrm{VC}}(ABC)$.

Proof. We use some properties of the greedy vertex cover algorithm and an optimal vertex cover, to show the following.

$$\begin{aligned}
2\widetilde{\mathrm{VC}}(BC) &\geq \widetilde{\mathrm{VC}}(B) + \widetilde{\mathrm{VC}}(BC) \\
&\geq (1-\varepsilon)\widetilde{\mathrm{VC}}(AB) + \widetilde{\mathrm{VC}}(BC) && \text{(assumption)} \\
&\geq (1-\varepsilon)\,\mathrm{VC}^*(AB) + (1-\varepsilon)\,\mathrm{VC}^*(BC) \\
&\geq (1-\varepsilon)\,\mathrm{VC}^*(ABC) \\
&\geq \frac{1}{2}(1-\varepsilon)\widetilde{\mathrm{VC}}(ABC)
\end{aligned}$$

□

We continue by showing that under some condition the greedy solution for ABC is a good approximation to an optimal solution on BC. The following lemma formalizes this idea.

Lemma 4.2. *For every $\varepsilon \in (0, 1/2)$ and disjoint segments A, B and C, if $\widetilde{\mathrm{VC}}(B) \geq (1-\varepsilon)\widetilde{\mathrm{VC}}(AB)$ then*

$$\mathrm{VC}^*(BC) \leq \widetilde{\mathrm{VC}}(ABC) \leq (8 + \mathcal{O}(\varepsilon))\,\mathrm{VC}^*(BC).$$

Proof. We use Lemma 4.1 and the fact that greedy vertex cover is a 2-approximation, therefore

$$\begin{aligned}\widetilde{\mathrm{VC}}(ABC) &\leq \frac{4}{1-\varepsilon}\widetilde{\mathrm{VC}}(BC) && \text{(Lemma 4.1)}\\ &\leq \frac{8}{1-\varepsilon}\,\mathrm{VC}^*(BC).\end{aligned}$$

On the other hand

$$\mathrm{VC}^*(BC) \leq \mathrm{VC}^*(ABC) \leq \widetilde{\mathrm{VC}}(ABC).$$

The Lemma follows from the fact that $\frac{1}{1-\varepsilon} = 1 + \mathcal{O}(\varepsilon)$, for $\varepsilon \in (0, 1/2)$. □

We are now ready to present the algorithm. The main idea is to have multiple buckets representing the stream. A new bucket is created each time a new edge is revealed, and it represents the segment starting at the time the bucket was created until the end of the stream. Some buckets may be deleted throughout the execution of the algorithm. After buckets are deleted, the remaining buckets are renumbered such that their order is kept, and their indices are successive. See Algorithm 4 for details.

Algorithm 4 Sliding Window

```
1:  procedure SlidingWindow
2:      for (u, v) revealed in stream do
3:          k = current number of buckets
4:          Create a new empty bucket B_{k+1}
5:          for i ∈ [k + 1] do
6:              if (u, v) is not covered in bucket B_i then
7:                  Add u and v to vertex cover of B_i
8:          Let i = 1
9:          while i < k do
10:             if exists j > i such that VC~(B_j) ≥ (1 − ε)VC~(B_i) then
11:                 Set j* to be the largest j such that VC~(B_j) ≥ (1 − ε)VC~(B_i)
12:                 for r ∈ [i + 1, j* − 1] do Delete B_r
13:                 Let i = j*
14:             else
15:                 Let i = i + 1
16:         Renumber the buckets as B_1, B_2, ...
17:         if B_2 contains the entire active window then
18:             Delete B_1
19:             Renumber the buckets as B_1, B_2, ...
20:     return The vertex cover of B_1
```

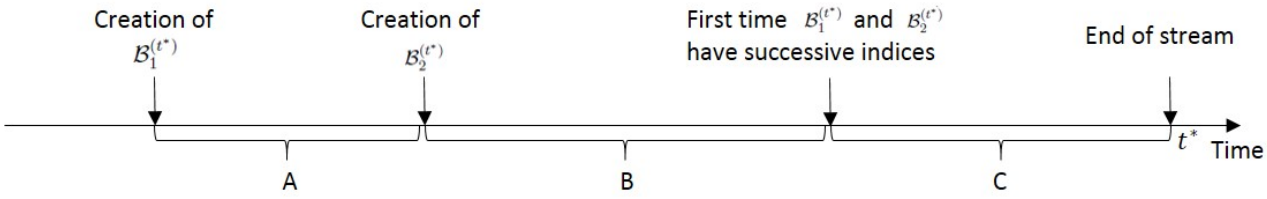

Figure 1: The edges in $\mathcal{B}_1^{(t^*)}$, i.e. bucket $\mathcal{B}_1$ at time t^* are partitioned into segments A, B and C.

Lemma 4.3. *Algorithm Sliding Window returns an* $(8+\mathcal{O}(\varepsilon))$*-approximation vertex cover for the active window, for every* $\varepsilon \in (0, 1/2)$.

Proof. Let W be the set of edges in the active window. At any time during the stream

$$\mathcal{B}_2 \subset W \subseteq \mathcal{B}_1, \tag{3}$$

because at the beginning of the stream when the first edge of W arrives, some bucket contains that single edge. Later, that bucket is deleted only if it become $\mathcal{B}_1$ and $W \subseteq \mathcal{B}_2$, and then $\mathcal{B}_2$ is renumbered to $\mathcal{B}_1$, thus $\mathcal{B}_1$ always contains the active window. If $\mathcal{B}_2$ would contain the active window, $\mathcal{B}_1$ would be deleted, and $\mathcal{B}_2$ would be renumbered, thus $\mathcal{B}_2 \subset W$.

Since the bucket numbering changes throughout the algorithm, let $\mathcal{B}_i^{(t)}$ be the bucket that had index i at time t, and let t^* be the time at the end of the stream. If throughout the stream no buckets were deleted between $\mathcal{B}_1^{(t^*)}$ and $\mathcal{B}_2^{(t^*)}$ (i.e. at time t when $\mathcal{B}_2^{(t^*)}$ was created, we had $\mathcal{B}_1^{(t^*)} = \mathcal{B}_k^{(t)}$ and $\mathcal{B}_2^{(t^*)} = \mathcal{B}_{k+1}^{(t)}$) then $W = \mathcal{B}_1^{(t^*)}$. This is because there is a difference of only one edge between $\mathcal{B}_1^{(t^*)}$ and $\mathcal{B}_2^{(t^*)}$, and $\mathcal{B}_2^{(t^*)} \subset W \subseteq \mathcal{B}_1^{(t^*)}$ by Equation 3. Therefore $\widetilde{\mathrm{VC}}(\mathcal{B}_1^{(t^*)}) = \widetilde{\mathrm{VC}}(W)$, and the algorithm returns a 2-approximation.

Otherwise at some point during the stream there were buckets deleted between $\mathcal{B}_1^{(t^*)}$ and $\mathcal{B}_2^{(t^*)}$. Denote by A the segment between the time $\mathcal{B}_1^{(t^*)}$ was created up to the time $\mathcal{B}_2^{(t^*)}$ was created (not including), denote by B the segment starting at the end of A until the first time $\mathcal{B}_1^{(t^*)}$ and $\mathcal{B}_2^{(t^*)}$ had successive indices (not including), i.e. the earliest time t for which there exists an i such that $\mathcal{B}_1^{(t^*)} = \mathcal{B}_i^{(t)}$ and $\mathcal{B}_2^{(t^*)} = \mathcal{B}_{i+1}^{(t)}$. Finally denote by C the segment starting at the end of B until the end of the stream. See Figure 1.

Since buckets were deleted between $\mathcal{B}_1^{(t^*)}$ and $\mathcal{B}_2^{(t^*)}$ at the end of segment AB, therefore $\widetilde{\mathrm{VC}}(B) \geq (1-\varepsilon)\widetilde{\mathrm{VC}}(AB)$ (lines 10 to 12 in Algorithm 4), and Lemma 4.2 applies. The segment ABC is from the creation of $\mathcal{B}_1^{(t^*)}$ to the end of the stream, therefore $\mathcal{B}_1^{(t^*)} = ABC$. Similarly, the segment BC is from the creation of $\mathcal{B}_2^{(t^*)}$ to the end of the stream, therefore $\mathcal{B}_2^{(t^*)} = BC$. Substituting the values in Equation 3 we get that

$$\mathcal{B}_2^{(t^*)} = BC \subset W \subseteq ABC = \mathcal{B}_1^{(t^*)}.$$

We finish the proof by applying Lemma 4.2, and obtaining that

$$\mathrm{VC}^*(W) \leq \widetilde{\mathrm{VC}}(\mathcal{B}_1^{(t^*)}) = \widetilde{\mathrm{VC}}(ABC) \leq (8+\mathcal{O}(\varepsilon))\,\mathrm{VC}^*(W).$$

□

Space usage At every point during the execution, $\widetilde{\mathrm{VC}}(\mathcal{B}_i) \geq (1-\varepsilon)\widetilde{\mathrm{VC}}(\mathcal{B}_{i+1})$ for every $i \in [k-1]$, and the size of a vertex cover is at most n, therefore at most

$$\mathcal{O}(\lg_{\frac{1}{1-\varepsilon}} n) = \mathcal{O}(\lg_{1+\mathcal{O}(\varepsilon)} n) = \mathcal{O}\left(\frac{1}{\varepsilon} \lg n\right)$$

buckets are used. Each bucket contains a greedy vertex cover, therefore at most n bits are required per bucket, totaling $\mathcal{O}(\frac{1}{\varepsilon} n \lg n)$ bits of space.

Scaling ε appropriately completes the proof of Theorem 1.3 which we restate bellow.

Theorem 1.3. *For every $\varepsilon > 0$, there exists a single pass $(8+\varepsilon)$-approximation deterministic algorithm for vertex cover in the sliding window streaming model that uses $\mathcal{O}(\frac{1}{\varepsilon} n \lg n)$ bits of space.*

References

[AKL16] S. Assadi, S. Khanna, and Y. Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 698–711. ACM, 2016. doi:10.1145/2897518.2897576.

[AKLY16] S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1345–1364, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. Available from: http://dl.acm.org/citation.cfm?id=2884435.2884528.

[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.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. American Elsevier Publishing Co., Inc., New York, 1976.

[BS15] M. Bury and C. Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms-ESA 2015*, pages 263–274. Springer, 2015.

[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, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. Available from: http://dl.acm.org/citation.cfm?id=2884435.2884527.

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

[CJMM16] G. Cormode, H. Jowhari, M. Monemizadeh, and S. Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. *arXiv preprint arXiv:1608.03118*, 2016.

[CMS13] M. S. Crouch, A. McGregor, and D. Stubbs. *Dynamic Graphs in the Sliding-Window Model*, pages 337–348. 2013. doi:10.1007/978-3-642-40450-4_29.

[DIMV14] E. D. Demaine, P. Indyk, S. Mahabadi, and A. Vakilian. On streaming and communication complexity of the set cover problem. In *Distributed Computing*, pages 484–498. Springer, 2014.

[EHL+15] H. Esfandiari, M. T. Hajiaghayi, V. Liaghat, M. Monemizadeh, and K. Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1217–1233, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics. Available from: http://dl.acm.org/citation.cfm?id=2722129.2722210.

[Kar72] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

[MV16a] A. McGregor and S. Vorotnikova. A note on logarithmic space stream algorithms for matchings in low arboricity graphs. *arXiv preprint arXiv:1612.02531*, 2016.

[MV16b] A. McGregor and S. Vorotnikova. Planar Matching in Streams Revisited. In K. Jansen, C. Mathieu, J. D. P. Rolim, and C. Umans, editors, *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, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:http://dx.doi.org/10.4230/LIPIcs.APPROX-RANDOM.2016.17.