




# Space Optimal Vertex Cover in Dynamic Streams

Kheeran K. Naidu   

Department of Computer Science, University of Bristol, UK

Vihan Shah  

Department of Computer Science, Rutgers University, Piscataway, NJ, USA

---

## Abstract

We optimally resolve the space complexity for the problem of finding an  $\alpha$ -approximate minimum vertex cover ( $\alpha$ MVC) in dynamic graph streams. We give a randomised algorithm for  $\alpha$ MVC which uses  $O(n^2/\alpha^2)$  bits of space matching Dark and Konrad’s lower bound [CCC 2020] up to constant factors. By computing a random greedy matching, we identify “easy” instances of the problem which can trivially be solved by returning the entire vertex set. The remaining “hard” instances, then have sparse induced subgraphs which we exploit to get our space savings and solve  $\alpha$ MVC.

Achieving this type of optimality result is crucial for providing a complete understanding of a problem, and it has been gaining interest within the dynamic graph streaming community. For connectivity, Nelson and Yu [SODA 2019] improved the lower bound showing that  $\Omega(n \log^3 n)$  bits of space is necessary while Ahn, Guha, and McGregor [SODA 2012] have shown that  $O(n \log^3 n)$  bits is sufficient. For finding an  $\alpha$ -approximate maximum matching, the upper bound was improved by Assadi and Shah [ITCS 2022] showing that  $O(n^2/\alpha^3)$  bits is sufficient while Dark and Konrad [CCC 2020] have shown that  $\Omega(n^2/\alpha^3)$  bits is necessary. The space complexity, however, remains unresolved for many other dynamic graph streaming problems where further improvements can still be made.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Streaming, sublinear and near linear time algorithms; Theory of computation  $\rightarrow$  Approximation algorithms analysis; Theory of computation  $\rightarrow$  Graph algorithms analysis

**Keywords and phrases** Graph Streaming Algorithms, Vertex Cover, Dynamic Streams, Approximation Algorithm

**Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2022.53

**Category** APPROX

**Funding** Kheeran K. Naidu: EPSRC Doctoral Training Studentship EP/T517872/1.

Vihan Shah: Research supported in part by a NSF CAREER Grant CCF-2047061.

**Acknowledgements** We are grateful to Sepehr Assadi and Christian Konrad for many helpful discussions. We also appreciate the valuable comments from our APPROX 2022 reviewers.

## 1 Introduction

*Graph streaming* is a setting in which a graph is specified by a sequence of edges, typically in arbitrary order. It is particularly useful for processing massive graphs where having random access to the edges of the graph is either impossible or computationally infeasible.

Research in this area began with *insertion-only streams*, where the stream is made up of a sequence of edge insertions only. In their seminal work, Feigenbaum, Kannan, McGregor, Suri, and Zhang [19] showed that for many problems including minimum spanning tree, connectivity, and bipartiteness,  $\Omega(n)$  bits of space is necessary and  $O(n \log n)$  bits is sufficient for any  $n$ -vertex graph. This logarithmic gap was often overlooked and deemed not important

---

A full version of the paper with the same title appears on arXiv.



© Kheeran K. Naidu and Vihan Shah;

licensed under Creative Commons License CC-BY 4.0

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022).

Editors: Amit Chakrabarti and Chaitanya Swamy; Article No. 53; pp. 53:1–53:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

when proving optimality for graph problems, but it left unresolved the question of whether the logarithmic factor was required for simply storing edges or if other techniques could remove it. About a decade later, Sun and Woodruff [39] showed that the logarithmic factor was indeed necessary by improving the lower bounds to  $\Omega(n \log n)$  bits, asymptotically matching the upper bounds up to constant factors.

*Dynamic graph streams*, which allow for sequences of both edge insertions and deletions, prove to be more difficult. Edges that arrive in the stream are not necessarily in the final graph as they may later be deleted. In fact, it is well-known in the community that it is impossible to deterministically return a single edge of a dense graph without storing all of its edges. As a result, almost all dynamic graph streaming algorithms rely on counters which use  $O(\log n)$  bits of space or they rely on  $L_0$ -sampling which optimally uses  $\Theta(\log^3 n)$  bits of space<sup>1</sup> [22, 25]. In essence, counters are used to solve the problem of determining whether an edge is present in an edge induced subgraph [18] (see also [15]), whereas  $L_0$ -sampling also returns the identity of a uniform random edge if one is present [1, 2, 27, 15, 9, 7, 30, 26, 11]. A notable exception includes spectral sparsification [23, 24] which relies on  $L_2$ -heavy-hitters (non-uniform sampling).

Resolving the space complexity up to constant factors for dynamic graph streaming problems has continued to be an elusive task. Ahn, Guha, and McGregor [1] gave an algorithm for connectivity using  $O(n \log^3 n)$  bits of space, and for several years, the best known lower bound was the insertion-only bound of  $\Omega(n \log n)$  bits [39]. However, in 2019, Nelson and Yu [36] improved the lower bound to  $\Omega(n \log^3 n)$  bits in the dynamic graph streaming setting. To the best of our knowledge, this is the only problem in this setting which has space bounds that prove the necessity of the  $\Theta(\log^3 n)$  overhead of randomly sampling an edge (using  $L_0$ -sampling). The approximate minimum cut problem which has a  $\Omega(n \log^3 n)$  bit lower bound [36] (and a  $O(n \log^4 n)$  bit upper bound [1]) similarly shows that logarithmic factors are necessary. A perhaps more surprising result was the recent progress on  $\alpha$ -approximate maximum matching ( $\alpha$ MM). The lower bound of  $\Omega(n^2/\alpha^3)$  bits [18] (see also [9]) and the previous upper bound of  $O(n^2/\alpha^3 \cdot \log^4 n)$  bits [9, 15] seem to indicate that the logarithmic overhead of sampling an edge is required. However, Assadi and Shah [11] improved the upper bound to  $O(n^2/\alpha^3)$  bits showing that this is not the case. On the other hand, for problems such as vertex cover [18], dominating set [26], and spectral sparsification [24], their space bounds have a gap of logarithmic factors, and therefore further improvements can still be made.

**Our Results.** In this work, we optimally resolve the space complexity up to constant factors for the problem of finding an  $\alpha$ -approximate minimum vertex cover ( $\alpha$ MVC) in a dynamic graph stream. In particular, we improve the upper bound to  $O(n^2/\alpha^2)$  bits, matching the  $\Omega(n^2/\alpha^2)$  bits lower bound [18] and showing that the logarithmic overhead is not required. Our main result is the following:

► **Theorem 1.** *There exists a randomised dynamic graph streaming algorithm for  $\alpha$ MVC that succeeds with high probability and uses  $O(n^2/\alpha^2)$  bits of space for any  $\alpha \leq n^{1-\delta}$  where  $\delta > 0$ .*

**Previous Work.** It has been shown by Dark and Konrad [18] that  $\Omega(n^2/\alpha^2)$  bits is necessary for  $\alpha$ MVC. They also gave a simple deterministic algorithm which uses  $O(n^2/\alpha^2 \cdot \log \alpha)$  bits of space, matching the lower bound up to logarithmic factors. Their algorithm arbitrarily

---

<sup>1</sup> This optimal space bound applies when the probability of success is at least  $1 - \frac{1}{\text{poly}(n)}$ .

partitions the vertex set into  $n/\alpha$  groups of size  $\alpha$  and uses counters, which introduce the logarithmic overhead, to maintain the number of edges between each of the  $\Theta(n^2/\alpha^2)$  pairs of vertex groups. The solution follows by computing a group-level minimum vertex cover, and then returning the vertices of the covering groups.

**Main Techniques.** We improve the approach of Dark and Konrad [18] by additionally computing a supporting random GREEDY matching and randomly partitioning the vertex set into  $n/\alpha$  groups, effectively using randomisation to reduce the space required. The random GREEDY matching returned is either large enough to imply a trivial solution for  $\alpha$ MVC (“easy” case) or implies sparseness properties of the residual subgraph induced by the unmatched vertices (“hard” case). To solve the “hard” cases, we use the sparseness properties and the random partitioning to argue that there are only  $O(1)$  many edges between each pair of vertex groups in the residual subgraph. Therefore, storing edge counters for each of the  $\Theta(n^2/\alpha^2)$  many pairs, as done by Dark and Konrad [18], now requires only  $O(n^2/\alpha^2)$  bits of space in total.

**Sampling Strategies.** The sparseness properties (of the residual subgraph) implied are reliant on the method of randomly sampling edges from the graph. Uniformly sampling from the edge set only implies sparseness properties sufficient for a small range of  $\alpha$  since it is skewed to sampling high degree vertices. On the other hand, non-uniform sampling – sampling from the neighbourhood of a random set of vertices, coined *neighbourhood edge sampling* by Assadi and Shah [11] – is less biased towards high degree vertices and implies the necessary sparseness properties for the full range of  $\alpha$ . Indeed, Assadi and Shah [11] also use the approach of computing a GREEDY matching on non-uniformly sampled edges to identify the “easy” and “hard” instances of  $\alpha$ MM. However, for  $\alpha$ MVC, our “easy” and “hard” instances differ from those of  $\alpha$ MM, so we require different guarantees. Furthermore, we use different techniques for solving the “hard” instances.

**Further Related Work.** Resolving the space complexity up to constant factors has also been achieved for non-graph problems in the general data streaming setting. For instance, Braverman, Katzman, Seidell, and Vorsanger [14] gave an upper bound for finding a constant factor approximation to the  $k$ -th frequency moment in constantly many passes that matches the lower bound of Woodruff and Zhang [40]. Price and Woodruff [37] showed a lower bound for any adaptive sparse recovery scheme that matches the upper bound of Indyk, Price, and Woodruff [21]. Graph problems in other streaming settings have also been studied. For example, the settings which allow multiple passes over the stream [31, 28, 6, 10, 32, 4, 8], have a random arrival order [31, 5, 12], or have highly structured deletions via a sliding window [16, 17, 13] have been considered. See the work by McGregor [33] for an excellent survey on graph streaming algorithms.

**Outline.** We begin in Section 2 with some important notation and tools which we will later use. In Section 3, we discuss the guarantees required from a random GREEDY matching for  $\alpha$ MVC. In Section 4, we present and analyse our algorithm that proves Theorem 1. Then, we conclude in Section 5.

## 2 Preliminaries

For any  $n$ -vertex graph  $G = (V, E)$ , let  $\mu(G)$  be the size of the maximum matching of the graph, let  $V^*(G)$  be a minimum vertex cover, and let  $\text{opt}(G)$  be its size. We will simply use  $\mu$ ,  $V^*$  or  $\text{opt}$  if the graph is clear from context. For any subset of edges  $F \subseteq E$ , we denote the set of their endpoints by  $V(F)$ . For any subgraph  $H$  of  $G$  and vertex  $v \in V$ , we use  $N_H(v)$  to denote the neighbourhood of  $v$  in  $H$ .

The graph  $G$  may be specified as a dynamic graph stream<sup>2</sup>  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$  such that  $\sigma_j = (i_j, \Delta_j)$  where  $i_j \in [m]$  for  $m = \binom{n}{2}$  and  $\Delta_j \in \{1, -1\}$  (insertions or deletions). Note that edges may only be deleted if they have previously been inserted. Additionally, the stream must produce a vector  $\mathbf{vec}(E) \in \{0, 1\}^m$  that defines the edge set  $E$ , i.e., the  $i^{\text{th}}$  entry of the vector indicates the presence of the edge indexed by  $i \in [m]$ .

In our work, we will rely on limited independence hash functions to reduce the space complexity of our algorithm. Roughly speaking, a hash function sampled from a family of  $k$ -wise independent hash functions behaves like a totally random function when considering at most  $k$  elements. For simplicity, when we mention a  $k$ -wise independent hash function, we will mean a hash function sampled from a family of  $k$ -wise independent hash functions. We use the following standard result for  $k$ -wise independent hash functions.

► **Proposition 2** ([34]). *For all integers  $n, m, k \geq 2$ , there is a family of  $k$ -wise independent hash functions  $\mathcal{H} = \{h : [n] \rightarrow [m]\}$  such that sampling and storing a function  $h \in \mathcal{H}$  takes  $O(k \cdot (\log n + \log m))$  bits of space.*

We shall also use the following concentration result on an extension of Chernoff-Hoeffding bounds for  $k$ -wise independent hash functions.

► **Proposition 3** ([38]). *Suppose  $h$  is a  $k$ -wise independent hash function and  $X_1, \dots, X_m$  are  $m$  random variables in  $\{0, 1\}$  where  $X_i = 1$  iff  $h(i) = 1$ . Let  $X := \sum_{i=1}^m X_i$ . Then, for any  $\varepsilon > 0$ ,*

$$\Pr(|X - \mathbb{E}[X]| \geq \varepsilon \cdot \mathbb{E}[X]) \leq \exp\left(-\min\left\{\frac{k}{2}, \frac{\varepsilon^2}{4 + 2\varepsilon} \cdot \mathbb{E}[X]\right\}\right).$$

Finally, we will use the following sketching tool for dynamic graph streams to test the size of the neighbourhood of a subset of vertices.

► **Proposition 4** ([11]). *Let  $a \geq b \geq 2$  be known integers. Consider a  $n$ -vertex graph  $G = (V, E)$  specified in a dynamic stream and let  $S \subseteq V$  be a known set. Then, given a set  $T \subseteq V$  of size at most  $a$  at the end of the stream, there exists a randomised algorithm that returns “Yes” if  $|N_G(S) \setminus T| \geq b$  or “No” if  $|N_G(S) \setminus T| \leq \frac{1}{2} \cdot b$ , uses  $O(\frac{a}{b} \cdot \log^3 n)$  bits of space, and succeeds with probability at least  $1 - n^{-3}$ . We denote one such algorithm as  $\mathcal{ALG}_{NT}(S; a, b)$ .*

## 3 Sampling Strategies for Random Greedy Matchings

In this section, we discuss and present the tool that we use to either find a large matching or show that the residual subgraph induced by the unmatched vertices is sparse.

<sup>2</sup> A dynamic graph stream is a special case of the strict turnstile data streaming model [35] where we consider only bit-vectors which represent the edges of a graph.

This approach was also used in Assadi and Shah’s recent work for  $\alpha$ MM [11] to identify “easy” and “hard” instances of the problem. For  $\alpha$ MVC, the “easy” case is finding a large enough matching to imply that we can trivially return the entire vertex set to solve the problem. The “hard” case is when we get a sparse residual subgraph, which is where our main savings in space come from. Identifying these cases can be accomplished by computing a GREEDY matching on randomly sampled edges of the graph.

Uniformly sampling as many edges as possible from a  $n$ -vertex graph (using  $L_0$ -sampling) without exceeding  $O(n^2/\alpha^2)$  bits of space followed by computing a GREEDY matching implies sparseness properties based on an already known maximum degree bound of the residual subgraph induced by the unmatched vertices [3, 29, 20]. Intuitively, uniform sampling is skewed towards sampling edges incident to high degree vertices. Hence, a GREEDY matching either matches these high degree vertices or matches many of its neighbours (decreasing their residual degree), and regardless of the size of the matching found, this gives a  $\text{poly}(\alpha)$  max degree bound in the residual graph. Furthermore, we can show that this also bounds the average degree (even when a small matching is found) since a worst-case instance<sup>3</sup> practically has all vertices in the residual subgraph with max degree. This degree bound, however, is only sufficient for solving  $\alpha$ MVC for any  $\alpha \ll n^{\frac{1}{3.5}}$ .

Non-uniformly sampling the edges using neighbourhood edge sampling followed by GREEDY, as done by Assadi and Shah [11], proves to give better sparseness properties, and thus a better average degree bound<sup>4</sup>. The benefit of neighbourhood edge sampling is that it biases away from sampling high degree vertices. Furthermore, when a small GREEDY matching is found, the implication is that the residual subgraph is sparse. Therefore, the average degree bound is sufficient for solving the “hard” case of  $\alpha$ MVC for the full range of  $\alpha$ .

As previously mentioned, Assadi and Shah’s algorithm called Match-or-Sparsify [11], does exactly this, although its guarantees are not sufficient for our purposes. Hence, we first discuss their algorithm, and then explain the alterations we make.

**Match-or-Sparsify.** For some parameter  $\beta \leq n$ , Assadi and Shah’s Match-or-Sparsify $_{\beta}$  algorithm non-uniformly samples edges using space  $O(\beta^2/\alpha^3)$  bits, and then computes a GREEDY matching from them. They give an intricate analysis to show that their algorithm either finds a large matching of size at least  $\beta/8\alpha$  or implies that the residual subgraph has at most  $20 \cdot \beta \cdot \log^4 n$  edges [11, Lemma 16]. Unlike uniform sampling, the residual properties (sufficiently) only hold when the matching is small – a key property exploited in their analysis. Additionally, in order for the guarantees to hold, they rely on the assumption that  $\beta \geq \alpha^2 \cdot n^{\delta}$ . Informally, when  $\beta$  is set as the size of the maximum matching  $\mu$ , Match-or-Sparsify $_{\mu}$  finds a large matching in “easy” graph cases and a sparse residual subgraph in the “hard” graph cases. However,  $\mu$  is not known, so they find a setting of  $\beta$  close to  $\mu$  by running Match-or-Sparsify $_{\beta}$  in parallel with  $\beta$  as all powers of 2 between 1 and  $n$ .

**Our Alterations.** The first thing to note is that the “easy” and “hard” instances for  $\alpha$ MM and  $\alpha$ MVC are not the same. Consider Match-or-Sparsify $_{\beta}$  when a large GREEDY matching is found. Since at least one endpoint of each matching edge must be in a vertex cover, it implies that  $\text{opt} \geq \frac{\beta}{8\alpha}$ . However, returning a solution to  $\alpha$ MVC at this stage can only be

<sup>3</sup> Consider a graph with a large clique on  $\Theta(n/\alpha)$  vertices where most the edges are sampled from, and many smaller cliques which assert the guaranteed max degree bounds.

<sup>4</sup> Having an average degree bound is more difficult to work with, but in this case, the bound on the average degree is much smaller than the bound on the max degree in the uniform case.

of size at most  $\Theta(\beta)$ , which would not be a trivial solution (the entire vertex set) with  $\beta \ll n$ . Furthermore, we have no guaranteed sparseness properties since the matching found is large. Hence, instead of needing  $\beta \approx \mu$ , which requires  $\log n$  many runs to find, we only need a single run of **Match-or-Sparsify** <sub>$n$</sub>  (with the parameter  $\beta$  fixed as  $n$ ). Secondly, their assumption that  $\beta \geq \alpha^2 \cdot n^\delta$  implies that  $\alpha \leq n^{\frac{1-\delta}{2}}$ , but we require it to hold for any  $\alpha \leq n^{1-\delta}$ . Since we have an additional  $\alpha$  factor of space (see [18]), we can increase the number of non-uniformly sampled edges to use  $O(n^2/\alpha^2)$  bits instead, which allows us to remove the assumption. Finally, the increase in the number of samples also allows us to increase the sparseness guarantees of the residual subgraph by an  $\alpha$  factor. Therefore, this altered **Match-or-Sparsify** <sub>$n$</sub>  algorithm, denoted by  $\mathcal{ALG}_{MS}$ , gives us the following lemma (the full proof is given in the arXiv version for completeness).

► **Lemma 5.** *There is a linear sketch for dynamic graph streams that, given any graph  $G = (V, E)$  specified via  $\mathbf{vec}(E)$ , uses  $O(n^2/\alpha^2)$  bits of space and with high probability outputs a matching  $M_{\text{easy}}$  that satisfies at least one of the following conditions for any  $\alpha \leq n^{1-\delta}$  and  $\delta > 0$ :*

- **Match-case:** *The matching  $M_{\text{easy}}$  has at least  $\frac{n}{8\alpha}$  edges;*
- **Sparsify-case:** *The induced subgraph of  $G$  on vertices not matched by  $M_{\text{easy}}$ , denoted by  $G_R$ , has at most  $20 \cdot \frac{n}{\alpha} \cdot \log^4 n$  edges.*

## 4 Main Result

In this section, we give a dynamic graph streaming algorithm for  $\alpha$ MVC for any  $n$ -vertex graph which implies our main result:

► **Theorem 1.** *There exists a randomised dynamic graph streaming algorithm for  $\alpha$ MVC that succeeds with high probability and uses  $O(n^2/\alpha^2)$  bits of space for any  $\alpha \leq n^{1-\delta}$  where  $\delta > 0$ .*

Before proceeding, we give the following standard assumption (with reason) which simplifies what we need to prove.

► **Assumption 6.** *A randomised dynamic graph streaming  $\Theta(\alpha)$ -approximation algorithm that uses  $O(n^2/\alpha^2)$  bits of space and succeeds on graphs where  $\text{opt} \geq \frac{n}{\alpha \cdot \log^2 n}$  is sufficient to prove Theorem 1.*

**Reason.** Let  $\mathcal{A}$  be an algorithm that returns a  $(c \cdot \alpha)$ -approximation using  $O(n^2/\alpha^2)$  bits of space. Run  $\mathcal{A}$  with parameter  $\alpha/c$  to get an  $\alpha$ -approximation which similarly uses  $O(n^2/\alpha^2)$  bits.

Then, since we can run  $\Theta(1)$  many algorithms which use  $O(n^2/\alpha^2)$  bits of space in parallel without asymptotically increasing the space, we run an additional algorithm which detects and outputs a solution for graphs with small  $\text{opt}$ .

**Algorithm for small  $\text{opt}$ .** We use the well-known algorithm for finding an exact minimum vertex cover in dynamic graph streams given the promise that  $\text{opt} \leq k$  with  $k = \frac{n}{\alpha \cdot \log^2 n}$  [15]. If  $\text{opt} < k$ , then we get an optimal solution; otherwise, we get a set of vertices of size  $k$  which are not necessarily a solution. Thus, we can detect this case by the size of the returned vertex cover being smaller than  $k$ . The space taken by the algorithm is  $O(k^2 \cdot \log^4 n) = O(n^2/\alpha^2)$  bits and it works for all  $\alpha = \omega(1)$  (for  $\alpha = \Theta(1)$  we can store the entire graph). ◀



■ **Algorithm 1** Optimal Dynamic Vertex Cover.

**Input:** A dynamic graph stream  $\sigma$  for a  $n$ -vertex graph  $G = (V, E)$ , a small constant  $\delta > 0$ , and a positive integer  $\alpha \leq n^{1-\delta}$

**Output:** A vertex cover  $V_C$  of  $G$

**Pre-processing:**

- 1: Initialise  $\mathbf{M}$  to be an instance of  $\mathcal{ALG}_{MS}$  (Lemma 5)
- 2: Randomly partition  $V$  into groups  $V_1, V_2, \dots, V_{\frac{n}{\alpha}}$  having size in  $[\alpha/2, 2\alpha]$
- 3: For each group  $V_i$ , initialise  $\mathbf{N}_i$  to be an instance of  $\mathcal{ALG}_{NT}(V_i; a, b)$  (Proposition 4) with  $a = n/\alpha$  and  $b = n^{\delta/2}$
- 4: Set  $c = 15/\delta$

**Processing the stream:**

- 5: Update  $\mathbf{M}$  and each  $\mathbf{N}_i$  using  $\sigma$
- 6: For every pair of groups  $V_i$  and  $V_j$ , store a counter  $C_{i,j}$  for the number of edges between them modulo  $c$
- 7: For every group  $V_i$ , store a counter  $C_i$  for the number of internal edges

**Post-processing:**

- 8: Let  $M_{\text{easy}}$  be the matching returned by  $\mathbf{M}$
- 9: **if**  $M_{\text{easy}}$  has at least  $\frac{n}{8 \cdot \alpha}$  edges **then return**  $V$
- 10: Let  $V_C$  be the union of all groups  $V_i$  containing a vertex of  $M_{\text{easy}}$  or with  $C_i > 0$
- 11: Add to  $V_C$  all remaining vertex groups  $V_i$  where  $\mathbf{N}_i$  returns “Yes” when  $T = V(M_{\text{easy}})$
- 12: Consider the multi-graph  $G'$  obtained by contracting the vertices of each remaining vertex group  $V_i$  into a single vertex  $v_i$  where  $C_{i,j}$  represents the multiplicity modulo  $c$  of each edge  $(v_i, v_j)$  in  $G'$
- 13: Greedily compute a vertex cover  $V'_C$  of  $G'$
- 14: For all  $v_i \in V'_C$ , add vertex group  $V_i$  to  $V_C$
- 15: **return**  $V_C$

**Algorithm Description.** Let  $G = (V, E)$  be specified by a dynamic graph stream,  $\delta > 0$ , and  $\alpha \leq n^{1-\delta}$  be the inputs to Algorithm 1. The algorithm, in its pre-processing step, partitions  $V$  into  $n/\alpha$  groups using a  $(10 \cdot \log n)$ -wise independent hash function (when the space allows, i.e., for small  $\alpha$ , we do this using a uniform random permutation instead), and we later show that all their sizes lie between  $\alpha/2$  and  $2\alpha$  with high probability. During the stream, it maintains counters modulo some constant for the number of edges between each pair of groups and (standard) counters for the number of internal edges of each group. In parallel, it computes a random matching  $M_{\text{easy}}$  using an instance of  $\mathcal{ALG}_{MS}$  (Lemma 5) and maintains residual neighbourhood size testers for each vertex group using instances of  $\mathcal{ALG}_{NT}$  (Proposition 4). In the post-processing step, if the matching is of size at least  $\frac{n}{8 \cdot \alpha}$ , then the entire vertex set is returned. Otherwise, the vertex groups containing any vertex of the matching or any internal edges are entirely picked in the solution – we call these *simple vertex groups*. Next, the remaining vertex groups  $V_i$  whose residual neighbourhood is large,  $|N_{G_R}(V_i)| = |N_G(V_i) \setminus V(M_{\text{easy}})| \geq n^{\delta/2}$  where  $G_R = G[V \setminus V(M_{\text{easy}})]$ , are added to the solution – we call these *residual vertex groups*. Finally, among the leftover *clean vertex groups*, the algorithm uses the counters modulo some constant to perform a group-level vertex cover, and then further adds the covering groups to the solution before returning it.

► **Definition 7** (Simple Vertex Groups). *We say that a vertex group  $V_i$  is simple if any of its vertices are matched by  $M_{\text{easy}}$  or it has at least one internal edge, i.e.,  $|V_i \cap V(M_{\text{easy}})| > 0$  or  $C_i > 0$ .*

► **Definition 8** (Residual Vertex Groups). *We say that a vertex group  $V_i$  is residual if it is not simple and has a large residual neighbourhood, i.e.,  $|N_{G_R}(V_i)| \geq n^{\delta/2}$ .*

► **Definition 9** (Clean Vertex Groups). *We say that a vertex group  $V_i$  is clean if it is not simple or residual, i.e.,  $|V_i \cap V(M_{\text{easy}})| = 0$ ,  $C_i = 0$  and  $|N_{G_R}(V_i)| < n^{\delta/2}$ .*

Note that throughout the subsequent analysis of Algorithm 1, all results succeed with high probability. Hence, at any point, we can do a simple union bound to show that they all hold with high probability. As such, we condition on this event here to avoid explicitly doing so during the analysis.

Let  $G$  be the input graph of the algorithm. We begin the analysis with the following observation: If  $G$  contains a matching of size at least  $\frac{n}{8\alpha}$ , then  $V$  is a valid  $(8\alpha)$ -approximation of a minimum vertex cover  $V^*$  since at least one endpoint of a matching edge must be in a valid vertex cover. Therefore, if the condition of Algorithm 1 is satisfied, the algorithm terminates and the solution is a valid  $\Theta(\alpha)$ -approximation (“easy” graph instances). Otherwise, the algorithm progresses with  $|M_{\text{easy}}| < \frac{n}{8\alpha}$ , i.e., the sparsify-case of Lemma 5 (“hard” graph instances). This implies that the residual subgraph  $G_R$  is sparse with at most  $20 \cdot \frac{n}{\alpha} \cdot \log^4 n$  many edges. As such, we need to prove that we also get a  $\Theta(\alpha)$ -approximation in the sparsify-case.

We highlight here that the algorithm adds vertex groups to the solution for various reasons, which are determined by whether it is a simple, residual, or clean vertex group (see Definitions 7–9). Hence, we proceed with the analysis of the sparsify-case by considering these different types of vertex groups separately.

**Simple Vertex Groups.** Let  $\mathcal{I}_s$  be the index set of the simple vertex groups. We argue that there are not too many of these, so we can add all of them to the solution.

► **Claim 10.** The number of simple vertex groups  $|\mathcal{I}_s|$  is at most  $2 \cdot \text{opt}(G)$ .

*Proof.* Each edge of the matching  $M_{\text{easy}}$  can cause up to two vertex groups to be classified as simple; however, they must have at least one vertex of  $V^*$  since at least one endpoint of every matching edge must be in  $V^*$ . Therefore, for every two groups classified as simple in this way, there is at least one vertex of  $V^*$  in their union. On the other hand, a group could also be classified as simple if it contains an internal edge, where one of its endpoints must be in  $V^*$ . Hence, for each group classified as simple in this way, there is at least one vertex of  $V^*$  in it. Then, it follows that the number of simple vertex groups must be at most  $2 \cdot |V^*| = 2 \cdot \text{opt}$ . ◁

**Residual Vertex Groups.** Let  $\mathcal{I}_r$  be the index set of the residual vertex groups. Recall that any residual vertex group must have at least  $n^{\delta/2}$  many residual neighbours. We note, however, that due to the guarantees of the neighbourhood size tester algorithm  $\mathcal{ALG}_{NT}$  (see Proposition 4), there are some misclassifications, so some residual vertex groups are also of size between  $\frac{1}{2} \cdot n^{\delta/2}$  and  $n^{\delta/2}$ . This will not be an issue, and moving forward, when we mention residual vertex groups, we assume that this includes the misclassifications. Now, we argue that there are not too many residual vertex groups, so we can add them all to the solution.

► **Claim 11.** The number of residual vertex groups  $|\mathcal{I}_r|$  is at most  $\text{opt}(G)$  with high probability.



Proof. We have that  $|V(M_{\text{easy}})|$  is at most  $\frac{n}{4\alpha}$  and  $G_R$  has at most  $20 \cdot \frac{n}{\alpha} \cdot \log^4 n$  many edges. As such,  $G_R$  has  $n - |V(M_{\text{easy}})| \geq \frac{n}{2}$  vertices, and the average degree of a vertex in  $G_R$  is at most  $20 \cdot \frac{n}{\alpha} \cdot \log^4 n \cdot \frac{2}{n} = \frac{40 \log^4 n}{\alpha}$ . Since each non-simple vertex group  $V_i$  is fully contained in  $G_R$  and has at most  $2\alpha$  vertices, we have that  $\mathbb{E}[|N_{G_R}(V_i)|] \leq \frac{40 \log^4 n}{\alpha} \cdot 2\alpha = 80 \log^4 n$ . Then, it follows by Markov's inequality that

$$\begin{aligned} \Pr(V_i \text{ is residual} \mid V_i \text{ is non-simple}) &\leq \Pr\left(|N_{G_R}(V_i)| \geq \frac{1}{2} \cdot n^{\delta/2}\right) \\ &\leq \frac{2 \cdot 80 \log^4 n}{n^{\delta/2}} \leq \frac{\log^5 n}{n^{\delta/2}}. \end{aligned} \tag{1}$$

Let  $X_i$  be the indicator random variable that a non-simple vertex group  $V_i$  is a residual vertex group, then  $R = \sum_{i \in [\frac{n}{\alpha}] \setminus \mathcal{I}_s} X_i$  is the number of residual vertex groups. By Equation (1), we have the following:

$$\mathbb{E}[R] = \sum_{i \in [\frac{n}{\alpha}] \setminus \mathcal{I}_s} \Pr(X_i) \leq \sum_{i \in [\frac{n}{\alpha}]} \frac{\log^5 n}{n^{\delta/2}} = \frac{n \cdot \log^5 n}{\alpha \cdot n^{\delta/2}}.$$

Finally, since  $\text{opt}(G) \geq \frac{n}{\alpha \cdot \log^2 n}$  (Assumption 6), a further application of Markov's inequality implies the result:

$$\Pr(|\mathcal{I}_r| > \text{opt}) \leq \Pr\left(R > \frac{n}{\alpha \cdot \log^2 n}\right) \leq \frac{n \cdot \log^5 n}{\alpha \cdot n^{\delta/2}} \cdot \frac{\alpha \log^2 n}{n} \leq n^{-\delta/4}.$$

Note that we can easily increase the success probability by running the algorithm in parallel  $40/\delta$  times and detecting failures when the number of residual groups is more than  $n/\alpha \cdot \log^2 n$ . Then, with probability at least  $1 - n^{-10}$ , one of the runs will succeed. This only increases the space of the algorithm by a constant factor since  $40/\delta = \Theta(1)$ .  $\triangleleft$

**Clean Vertex Groups.** Let  $\mathcal{I}_c$  be the index set of the clean vertex groups and let  $\mathcal{I}_c^+$  be the ones added to the solution, which also corresponds to the group-level vertex cover  $V'_C$  in Algorithm 1.

Before analysing the group-level vertex cover, we note that the relevant counters are stored modulo  $c$ . This means that if the number of edges between clean vertex groups is some multiple of  $c$ , the corresponding counter would be 0 and the group-level vertex cover would be incorrect. Hence, we want the number of edges between clean vertex groups to be less than  $c$  with high probability.

$\triangleright$  **Claim 12.** For all pairs of clean vertex groups  $V_i$  and  $V_j$ , with high probability,

$$|N_G(V_i) \cap V_j| < c.$$

Proof. We prove a slightly generalised statement which implies what we need. We show that there are less than  $c$  edges of  $G_R$  between any clean vertex group  $V_i$  and any other vertex group  $V_j$ . This implies what we need since, by definition, all edges between clean vertex groups are in  $G_R$ .

Consider the random partitioning of  $V$  using an at least  $(3 \cdot c)$ -wise independent hash function (the algorithm uses  $(10 \cdot \log n)$ -wise independence). A residual neighbour of the clean vertex group  $v \in N_{G_R}(V_i)$  uniformly belongs to any of the other vertex groups. Since there are  $\frac{n}{\alpha} - 1$  of these (including  $V_j$ , but not including  $V_i$ ), the probability that  $v \in V_j$  is at most  $\frac{2\alpha}{n}$ .

## 53:10 Space Optimal Vertex Cover in Dynamic Streams

Now, since clean vertex groups are non-residual,  $|N_{G_R}(V_i)| \leq n^{\delta/2}$ , and for a fixed  $V_i$  and  $V_j$ , we have that

$$\Pr(|N_{G_R}(V_i) \cap V_j| \geq c) \leq \binom{n^{\delta/2}}{c} \cdot \left(\frac{2\alpha}{n}\right)^c \leq \left(\frac{2\alpha}{n^{1-\delta/2}}\right)^{15/\delta} \leq n^{-7}$$

where we have used  $\alpha \leq n^{1-\delta}$  and  $c = 15/\delta$  in the final inequalities. Then, the result holds with probability at least  $1 - n^{-5}$  by a union bound over all pairs of vertex groups.

Note that for small  $\alpha$  we will partition  $V$  into groups of size exactly  $\alpha$  with a uniform random permutation due to concentration and space reasons (see Claim 15), but the above arguments also hold in this case.  $\triangleleft$

With Claim 12, we can assume that all the counters between clean vertex groups count exactly the number of edges with high probability, that is, the modulo has no effect on the correctness of the algorithm. Thus, the setting is now identical to that of Dark and Konrad's algorithm [18], and we follow a similar argument as they did to analyse the group-level vertex cover and the corresponding subset of clean vertex groups added.

$\triangleright$  **Claim 13.** The number of clean vertex groups added  $|\mathcal{I}_c^+|$  is at most  $2 \cdot \text{opt}(G)$ .

*Proof.* Consider the subgraph  $H = G[\cup_{i \in \mathcal{I}_c} V_i]$  induced by the clean vertex groups. Observe that since  $H$  is an induced subgraph of  $G$ ,  $\text{opt}(H) \leq \text{opt}(G)$ . Then, since the vertex contractions to obtain the multi-graph  $G'$  from  $H$  cannot increase the size of its minimum vertex cover, we have that  $\text{opt}(G') \leq \text{opt}(H)$ . Finally, since we greedily compute the group-level vertex cover  $V'_C$ , it is a 2-approximation and we have that  $|\mathcal{I}_c^+| = |V'_C| \leq 2 \cdot \text{opt}(G') \leq 2 \cdot \text{opt}(G)$ .  $\triangleleft$

By combining the analysis of the simple, residual, and clean vertex groups, we prove the approximation factor of the algorithm.

$\blacktriangleright$  **Lemma 14.** *Algorithm 1 returns a valid  $\Theta(\alpha)$ -approximation of a minimum vertex cover for any input graph  $G$  with  $\text{opt} \geq \frac{n}{\alpha \cdot \log^2 n}$ .*

*Proof.* We first show that the solution  $V_C$  is indeed a valid vertex cover, then we prove that it is a  $\Theta(\alpha)$ -approximation.

**Validity.** For the sake of finding a contradiction, let  $e \in E$  be an edge which is not covered by  $V_C$ . Observe that any non-clean vertex group  $V_i$  is added to  $V_C$ ; thus, all edges with at least one endpoint in any of these vertex groups are covered. So, we have that  $e$  must be in  $G[\cup_{i \in \mathcal{I}_c} V_i]$ , the subgraph induced by the clean vertex groups.

Let  $i, j \in \mathcal{I}_c$  be such that  $e$  has endpoints in the clean vertex groups  $V_i$  and  $V_j$ , implying that there is an edge between their corresponding contracted vertices  $v_i$  and  $v_j$  in the multi-graph  $G'$ . It follows that one of  $v_i$  or  $v_j$  must be in the computed group-level vertex cover  $V'_C$ , so all vertices of either  $V_i$  or  $V_j$ , including at least one endpoint of  $e$ , are added to  $V_C$ . However, this means that  $e$  is covered by  $V_C$ , a contradiction.

**Approximation.** Observe that the solution  $V_C$  is comprised of a (disjoint) union of all simple vertex groups, all residual vertex groups, and a subset of clean vertex groups. Recall that  $\mathcal{I}_s$ ,  $\mathcal{I}_r$  and  $\mathcal{I}_c^+$  are the corresponding index sets of these groups.

By Claims 10, 11, and 13, we have that  $|\mathcal{I}_s| + |\mathcal{I}_r| + |\mathcal{I}_c^+| \leq 5 \cdot \text{opt}$ . Finally, since the size of each vertex group is at most  $2\alpha$ , we can bound the size of the solution as follows:

$$|V_C| = \sum_{i \in \mathcal{I}_s \cup \mathcal{I}_r \cup \mathcal{I}_c^+} |V_i| \leq 2\alpha \cdot (|\mathcal{I}_s| + |\mathcal{I}_r| + |\mathcal{I}_c^+|) \leq 10\alpha \cdot \text{opt}. \quad \blacktriangleleft$$

It remains to show that the algorithm can be implemented using  $O(n^2/\alpha^2)$  bits of space. Algorithm 1 randomly partitions  $V$ , maintains several instances of  $\mathcal{ALG}_{MS}$  (Lemma 5) and  $\mathcal{ALG}_{NT}$  (Proposition 4), and stores various counters. To show the space usage of the algorithm, we first consider each of these components separately.

▷ **Claim 15.** The partitioning of  $V$  into  $\frac{n}{\alpha}$  vertex groups of size in the range  $[\alpha/2, 2\alpha]$  uses  $O(n^2/\alpha^2)$  bits of space and succeeds with high probability.

*Proof.* We show that for small  $\alpha < \log^2 n$ , i.e., when we have sufficient space, we can achieve this with a uniform random permutation, and for large  $\alpha \geq \log^2 n$ , we use a  $(10 \cdot \log n)$ -wise independent hash function.

**Small  $\alpha$ .** For any  $\alpha < \log^2 n$ , we can randomly permute the vertices using  $O(n \log n) = O(n^2/\alpha^2)$  random bits to create a uniform random partitioning of  $V$  into  $\frac{n}{\alpha}$  groups of size  $\alpha$ .

**Large  $\alpha$ .** For any  $\alpha \geq \log^2 n$ , we can partition  $V$  using a  $(10 \cdot \log n)$ -wise independent hash function  $h : [n] \rightarrow [\frac{n}{\alpha}]$  which uses  $O(\log^2 n) = O(n^2/\alpha^2)$  bits by Proposition 2. We bound the size of the groups as follows: Consider any group  $V_j$  ( $j \in [\frac{n}{\alpha}]$ ) and let  $X_i$  be the random variable that is 1 if vertex  $i$  is hashed to  $V_j$ , i.e.,  $h(i) = j$ . Let  $X = \sum_i X_i$  represent the number of vertices in group  $V_j$ . We have  $\mathbb{E}[X] = n \cdot (\alpha/n) = \alpha$ . Using Proposition 3 with  $\varepsilon = 0.1$ ,

$$\Pr(|X - \mathbb{E}[X]| \geq \varepsilon \cdot \mathbb{E}[X]) \leq \exp(-5 \log n) \leq n^{-5}.$$

A union bound over all groups implies that with probability at least  $1 - n^{-4}$ , all groups have size between  $0.9\alpha$  and  $1.1\alpha$ . ◁

▷ **Claim 16.** The instances of  $\mathcal{ALG}_{MS}$  and  $\mathcal{ALG}_{NT}$ , and the counters use  $O(n^2/\alpha^2)$  bits of space.

*Proof.* We use one instance of  $\mathcal{ALG}_{MS}$  (Lemma 5) which takes space  $O(n^2/\alpha^2)$  bits. We use  $n/\alpha$  instances of  $\mathcal{ALG}_{NT}$  (Proposition 4) with parameters  $a = n/\alpha$  and  $b = n^{\delta/2}$  each of which take space  $O(\frac{a}{b} \log^3 n) = O((n/\alpha) \cdot (\log^3 n/n^{\delta/2})) = o(n/\alpha)$  bits. This implies that the total space used by  $n/\alpha$  instances is  $O(n^2/\alpha^2)$  bits. We maintain counters modulo a constant  $c = 15/\delta$  for the number of edges between every pair of vertex groups. Each takes  $O(1)$  bits of space, and since there are  $O(n^2/\alpha^2)$  many of these counters, this totals  $O(n^2/\alpha^2)$  bits of space. We also maintain counters for the number of internal edges for each group which requires  $O(\log n) = o(n/\alpha)$  bits of space each. Since there are  $\frac{n}{\alpha}$  many groups, this totals  $O(n^2/\alpha^2)$  bits of space. ◁

Hence, by Claims 15 and 16, we have shown that the components of Algorithm 1 use  $O(n^2/\alpha^2)$  bits in total. We still, however, need to consider the format of the output. When  $\alpha$  gets large enough, the space is only  $o(n)$ , whereas simply storing the output – the vertices of a solution – could require  $\Theta(n)$  bits of space. We solve this by showing that we can implicitly store the solution when there is limited space.

▷ **Claim 17.** The output of Algorithm 1 can be maintained using  $O(n^2/\alpha^2)$  bits of space.

*Proof.* For  $\alpha < \log^2 n$ , we can maintain the vertices of the solution explicitly. For  $\alpha \geq \log^2 n$ , we rely on the hash function  $h$  used to partition  $V$  (see Claim 15). Recall that vertices are added to the solution at a group level, so we can simply maintain a bit vector of length  $\frac{n}{\alpha}$  representing the groups added to the solution. Then, the output consists of  $h$  and the bit vector which is sufficient for checking if a vertex belongs to the solution and uses  $O(\log^2 n + \frac{n}{\alpha}) = O(n^2/\alpha^2)$  bits of space. ◁

We have now shown that Algorithm 1 can be implemented using  $O(n^2/\alpha^2)$  bits of space. Therefore, combined with Lemma 14 and Assumption 6, we have proven our main result, Theorem 1.

## 5 Conclusion

In this paper, we have resolved the space complexity of  $\alpha$ MVC for the full range of  $\alpha$ . We have provided a randomised algorithm which asymptotically matches the lower bound [18] up to constant factors, showing that  $\Theta(n^2/\alpha^2)$  is necessary and sufficient for this problem.

The previous best algorithm for  $\alpha$ MVC was a deterministic one using  $O(n^2/\alpha^2 \cdot \log \alpha)$  bits of space [18]. We have shown that we can remove the logarithmic overhead using randomness. Can we, however, remove this logarithmic factor using deterministic techniques or otherwise prove a deterministic lower bound which shows that it is necessary?

Our work continues the direction set by the results on connectivity [1, 36] and matchings [18, 11]; we resolve the space complexity (up to constant factors) of another problem in the dynamic graph streaming setting. However, other problems still remain open. Hence, can we achieve this for other dynamic graph streaming problems such as dominating set [26] and spectral sparsification [24]?

---

## References

- 1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.
- 2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.
- 3 KookJin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *International Conference on Machine Learning*, pages 2237–2246. PMLR, 2015.
- 4 Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 708–742. SIAM, 2022. doi:10.1137/1.9781611977073.32.
- 5 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.
- 6 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 265–276. ACM, 2019. doi:10.1145/3313276.3316361.
- 7 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for  $(\delta + 1)$  vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786. SIAM, 2019.

- 8 Sepehr Assadi, Arun Jambulapati, Yuja Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.
- 9 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.ch93.
- 10 Sepehr Assadi, Gillat Kol, Raghuvansh R. Saxena, and Huacheng Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16–19, 2020*, pages 354–364. IEEE, 2020. doi:10.1109/FOCS46700.2020.00041.
- 11 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.9.
- 12 Aaron Bernstein. Improved bounds for matching in random-order streams. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8–11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 12:1–12:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.12.
- 13 Leyla Biabani, Mark de Berg, and Morteza Monemizadeh. Maximum-weight matching in sliding windows and beyond. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6–8, 2021, Fukuoka, Japan*, volume 212 of *LIPICs*, pages 73:1–73:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ISAAC.2021.73.
- 14 Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using  $o(n^{1-2/k})$  bits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.
- 15 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016*, pages 1326–1344. SIAM, 2016. doi:10.1137/1.9781611974331.ch92.
- 16 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013 – 21st Annual European Symposium, Sophia Antipolis, France, September 2–4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013. doi:10.1007/978-3-642-40450-4\_29.
- 17 Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Christopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4–6, 2014, Barcelona, Spain*, volume 28 of *LIPICs*, pages 96–104. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.APPROX-RANDOM.2014.96.

- 18 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPICs*, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.30.
- 19 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In Josep Diaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 531–543. Springer, 2004. doi:10.1007/978-3-540-27836-8\_46.
- 20 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 – August 2, 2019*, pages 491–500. ACM, 2019. doi:10.1145/3293611.3331603.
- 21 Piotr Indyk, Eric Price, and David P Woodruff. On the power of adaptivity in sparse recovery. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 285–294. IEEE, 2011.
- 22 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.
- 23 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66.
- 24 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1814–1833. SIAM, 2020. doi:10.1137/1.9781611975994.111.
- 25 Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P. Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486. IEEE Computer Society, 2017.
- 26 Sanjeev Khanna and Christian Konrad. Optimal bounds for dominating set in graph streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 93:1–93:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.93.
- 27 Christian Konrad. Maximum matching in turnstile streams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015. doi:10.1007/978-3-662-48350-3\_70.
- 28 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPICs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.74.
- 29 Christian Konrad. Mis in the congested clique model in  $o(\log \log \delta)$  rounds. *arXiv preprint*, 2018. arXiv:1802.07647.



- 30 Christian Konrad. Frequent elements with witnesses in data streams. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 83–95, 2021.
- 31 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0\_20.
- 32 Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPICs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.APPROX/RANDOM.2021.19.
- 33 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.
- 34 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- 35 S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2), 2005. doi:10.1561/04000000002.
- 36 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1844–1860. SIAM, 2019.
- 37 Eric Price and David P Woodruff. Lower bounds for adaptive sparse recovery. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 652–663. SIAM, 2013.
- 38 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, 1995.
- 39 Xiaoming Sun and David P Woodruff. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- 40 David P Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 941–960, 2012.