Dynamic Graph Stream Algorithms in $o(n)$ Space

Zengfeng Huang[1] and Pan Peng*[2]

1 **University of New South Wales, Sydney, Australia**
`zengfeng.huang@unsw.edu.au`
2 **Department of Computer Science, TU Dortmund, Dortmund, Germany; and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China**
`pan.peng@tu-dortmund.de`

Abstract

In this paper we study graph problems in dynamic streaming model, where the input is defined by a sequence of edge insertions and deletions. As many natural problems require $\Omega(n)$ space, where n is the number of vertices, existing works mainly focused on designing $\tilde{O}(n)$ space algorithms. Although sublinear in the number of edges for dense graphs, it could still be too large for many applications (e.g. n is huge or the graph is sparse). In this work, we give single-pass algorithms beating this space barrier for two classes of problems. We present $o(n)$ space algorithms for estimating the number of connected components with additive error εn and $(1+\varepsilon)$-approximating the weight of minimum spanning tree. The latter improves previous $\tilde{O}(n)$ space algorithm given by Ahn et al. (SODA 2012) for connected graphs with bounded edge weights. We initiate the study of approximate graph property testing in the dynamic streaming model, where we want to distinguish graphs satisfying the property from graphs that are ε-far from having the property. We consider the problem of testing k-edge connectivity, k-vertex connectivity, cycle-freeness and bipartiteness (of planar graphs), for which, we provide algorithms using roughly $\tilde{O}(n^{1-\varepsilon})$ space, which is $o(n)$ for any constant ε. To complement our algorithms, we present $\Omega(n^{1-O(\varepsilon)})$ space lower bounds for these problems, which show that such a dependence on ε is necessary.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases dynamic graph streams, sketching, property testing, minimum spanning tree

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.18

1 Introduction

Graphs or networks are a natural way to describe structural information. For example, users of Facebook and the acquaintance relations among them form a social network, the proteins together with interactions between them define a biological network, and web-pages and hyperlinks give rise to a huge web graph. Due to the rapid development of information technology, many such graphs become extremely large, and are constantly changing, which poses great challenges for analyzing their structures. Over the last decade, the data stream model [34] has proven to be successful in dealing with *big data*. In this model, the algorithm should make only one pass (or a few passes) over the stream, and use sublinear working space. The time required to output the final answer and process each element is also important. There is a growing body of work studying graph problems over data streams. Graph streams

* Supported by ERC grant No. 307696.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 18; pp. 18:1–18:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

were first considered by Henzinger et al. [24], and later have been extensively studied in the *insertion-only* model (eg., [17, 18, 34]), where there is no edge deletion in the stream. Recently, starting from the seminal works of Ahn, Guha and McGregor [2, 3], the interest has shifted to the *dynamic streaming model*, where the edges can be both inserted and deleted (see eg., [28, 29, 1, 5, 9, 10, 7, 31, 6, 33, 15, 23]). In this setting, most algorithms designed are linear sketch-based, which is also an effective technique for processing distributed graphs. For more information about graph streaming algorithms see the recent survey by McGregor [32].

For graph streams, both insertion-only and dynamic, the research in the past has mostly focused on the *semi-streaming model*, in which the algorithms are allowed to use $\tilde{O}(n)$ space, where n is the number vertices in the graph. (For notational convenience, we will use $\tilde{O}(g)$ and $\tilde{\Omega}(g)$ to hide $\operatorname{poly}\log(g)$ factors.) The reason behind this is that even in the insertion-only model, many natural graph problems require $\Omega(n)$ space (e.g. testing if the graph is connected [18]). Note that the allowed space in semi-streaming model is sublinear in the input size as the number of edges of the graph might be as large as $\Omega(n^2)$. However, in many real applications n is huge and the input graph is already very sparse, an $\tilde{O}(n)$ algorithm might be even worse than just storing all the edges. From this perspective, one may naturally ask the question *which kind of problems can be solved with even less space, i.e., $o(n)$ space.*

To the best of our knowledge, very few results are known in this direction. Chitnis et al. [10] and Fafianie and Kratsch [16] introduced parameterized graph stream algorithms which may only use $o(n)$ space with some promise of the size of the solution. This parameterized setting has been further investigated in [9]. In addition, it has been shown that the size of the maximum matching can be approximated within constant factor in $\tilde{O}(n^{4/5})$ space for graphs with bounded arboricity [14, 9, 7].

In this paper, we study two classes of graph problems that admit single-pass $o(n)$ space algorithms in the dynamic streaming model. The first class contains the problems of estimating the number of connected components and the weight of minimum spanning tree (MST). We show that one can estimate the number of connected components within an *additive* error of εn with $o(n)$ space and post-processing time, for any small constant $\varepsilon > 0$. We also present an algorithm to $(1+\varepsilon)$-approximate the weight of MST with $o(n)$ space and post-processing time for connected graphs with bounded edge weights, which improves the best known algorithm with $\tilde{O}(n)$ space in the same setting given by Ahn et al. [2]. It is worthy noting that the problem of estimating the number of connected components within small *multiplicative* error requires $\Omega(n)$ space, as it is generally harder than the problem of (exactly) testing graph connectivity; and that estimating the weight of MST for graphs with arbitrarily large edge weights (e.g., $\Omega(\log n)$) requires $\Omega(n)$ space (see Theorem 12). Previously these two problems have been studied in the framework of *sublinear time algorithms* (see eg. [8, 39]).

The second class consists of problems that are relaxations of deciding graph properties. Given a huge graph, it is very useful to know whether the graph has some predetermined property, such as k-connectivity, bipartiteness, cycle-freeness and etc., which provide valuable information about the graph. However, besides the requirement of $\Omega(n)$ space, exactly testing of these properties sometimes is a too strong requirement for analyzing highly dynamic graphs, since the answer may change in the next second due to an insertion or deletion of a single edge. In this paper, we initiate the study of *approximate graph property testing* in the dynamic streaming model: we want to test whether a graph satisfies some property or one has to modify a small constant fraction of edges to make it have the property. This notion of approximation is adapted from the framework of *property testing* [21, 22, 36], and a large number of existing literatures have given efficient testing algorithms (called *testers*) for many properties under different query models (see surveys [20, 38]). We show that some

Table 1 Upper and lower bounds of streaming testers.

	Space $\tilde{O}$	Space lower bound Ω
Connectivity	$n^{1-\varepsilon}$	$n^{1-8\varepsilon}$
k-edge connectivity	$k^{1+\varepsilon} \cdot n^{1-\varepsilon}$	
k-vertex connectivity	$\frac{k^{1+\varepsilon/4}}{\varepsilon} \cdot n^{1-\varepsilon/4}$	
Cycle-freeness	$n^{1-\varepsilon+\varepsilon^2}$	$n^{1-8\varepsilon}$
Bipartiteness of planar graphs	$n^{1-\Omega(\varepsilon^2)}$	$n^{1-4\varepsilon}$

fundamental properties can be tested in both $o(n)$ space and post-processing time in the dynamic streaming model and we also present close lower bounds for these problems which hold even in the insertion-only model. We remark that McGregor [32] also suggested to study the (approximate) property testers in graph streaming model, and asked whether more space-efficient algorithms exist for these problems, and we thus give affirmative answer to this question.

1.1 Our results

Now we formally state our main results. Our results regarding estimating the number of connected components and the MST weight are as follows.

- **Estimating the number of connected components.** We present a dynamic streaming algorithm that estimates the number of connected components within additive error εn in $\tilde{O}(n^{1-\varepsilon+\varepsilon^{q+1}})$ space and post-processing time for any constant $q \geq 1$. We note that a lower bound of $\Omega(n^{1-O(\varepsilon)})$ for this problem follows from the work [41].
- **Estimating the weight of minimum spanning tree (MST).** In this problem, we want to estimate the weight of the MST of a graph with edge weights in the set $\{1, 2, \cdots, W\}$. We give a dynamic streaming algorithm that computes a $(1+\varepsilon)$-approximation of the MST weight and uses space and post-processing time $\tilde{O}(Wn^{1-\frac{\varepsilon}{W-1}+\frac{\varepsilon^t}{(W-1)^t}})$ for any constant $t \geq 1$. By an argument in [8], the result can be extended to non-integral weights, as long as the ratio between the largest and the smallest weight is bounded. A space lower bound of $\Omega(n^{1-\frac{4\varepsilon}{W-1}})$ is shown for this problem.

We also present approximate testing algorithms for a number of fundamental graph properties. Before stating the performance of these algorithms, we first introduce some definitions. Given a graph property Π, an m-edge graph G is called *ε-far from having* Π if one has to modify more than εm edges of G to get a graph G' satisfying Π. This distance definition is adapted from [36] and is most suitable for general graphs where neither edge density nor maximum degree is restricted. We call an algorithm a *(dynamic) streaming tester* for Π, if it makes a single-pass over a stream of edge insertions and deletions, with probability at least $2/3$, accepts any graph satisfying Π, and rejects any graph that is ε-far from having Π.

We give sketch-based streaming testers for properties of being connected, k-edge connected, k-vertex connected, cycle-freeness and bipartite (for planar graphs). The performance of our testers are summarized in Table 1. We stress that most of our testers have (asymptotically) the same post-processing time as the space they used except for testing k-edge connectivity when $k \geq \Omega(n^{\varepsilon/(1+\varepsilon)})$ and k-vertex connectivity when $k \geq \Omega(n^{\varepsilon/(4+\varepsilon)})$.

1.2 Our techniques

To estimate the number of connected components with small additive error εn, we note that it is sufficient to estimate the number $\mathrm{scc}(G)$ of connected components of small size (i.e., $O(1/\varepsilon)$), since the number of components of size larger than this is at most $O(\varepsilon n)$ (see also [8]). To estimate $\mathrm{scc}(G)$, the following vertex sampling framework is used: we sample a sufficient large set of vertices S by sampling each vertex in G with some probability p, and then use the statistics of the sampled connected components of the original graph to estimate $\mathrm{scc}(G)$. For any small connected component C in G, it is likely that all the vertices in C will be sampled out. Conditioned on this, we add $1/p^{|C|}$ to our final estimator, which is the reciprocal of the probability that C is entirely sampled out. Now the task is then to identify which subsets of S are connected components in the original graph. A trivial way is to check all subsets of S, which takes too much time. A more efficient way is to only check all the connected components in $G[S]$, since a sampled component of G must also form a component in $G[S]$. We carefully use a set of linear sketches to do this. More specifically, we first recover all connected components in $G[S]$ by invoking a sketch-based streaming algorithm given in [2], which only needs space near-linear in $|S|$. Then we use (different) linear sketches to check if any of these components is indeed a connected component of the original graph. We remark that the first set of linear sketches of a vertex v sketch its neighborhood information in $G[S]$, while the second set sketch its neighborhood information in G. Our $o(n)$ space streaming algorithm for $(1+\varepsilon)$-approximating the weight of MST follows via a connection between the number of connected components and the weight of MST established in [8].

To give testers for some graph property Π in dynamic streaming model, we start from the observation that if a graph G is far from having Π, then typically, there exist many small disjoint subgraphs, each of which is a witness that the graph G does not satisfy Π. (For example, if Π is connectivity, then there exists at least $\Omega(\varepsilon m)$ connected components of size at most $O(1/\varepsilon)$ in a graph that is ε-far from being connected.) This implies that by sampling a sufficient large set of vertices, with high probability, one of such subgraphs will be entirely sampled. Checking which vertices form a witness of the original graph can then be done by using the aforementioned framework. Different sketches will be used for testing different properties.

To prove lower bounds for our studied problems, we give reductions from *Boolen Hidden Hypermatching (BHH)* problem that was studied in [41]. Our reductions share similarity with the reduction in [41] to the cycle-counting problem and the reductions in [27, 30] to the approximate max-cut problem.

1.3 Related work

Ahn et al. [2] initiated the study of graph sketches, and gave dynamic semi-streaming algorithms for computing a spanning forest (which can be used to count the exact number of connected components), and $(1+\varepsilon)$-approximate the weight of MST. They also proposed algorithms to *exactly* testing of a set of properties, including testing connectivity, k-edge connectivity, and bipartiteness. Recently, Guha et al. gave dynamic streaming algorithms for exactly testing of k-vertex connectivity [23]. All these algorithms use $\tilde{O}(n)$ space ($\tilde{O}(kn)$ for k-connectivity). On the other hand, the randomized space lower bounds for these exact testing problems were known to be $\Omega(n)$ in the insertion-only model [17, 18]. Recently, Sun and Woodruff improved these lower bounds to $\Omega(n \log n)$ [40]. Verbin and Yu [41] proved a lower bound for cycle-counting, which implied a lower bound of $\Omega(n^{1-O(\varepsilon)})$ for estimating the number of components.

In the *random order* insertion-only model Kapralov et al. [26] gave a one pass streaming algorithm that estimates the maximum matching size with polylogarithmic approximation ratio in polylogarithmic space. Although sublinear in n, the model considered is very different from ours.

Sublinear time algorithms for estimating the number of connected component and the weight of MST were first given by Chazelle et al. [8]. Later these two problems have been further considered in geometric settings [11, 13, 19]. In particular, Frahling et al. studied the problem of $(1+\varepsilon)$-approximating the weight of MST in dynamic geometric data stream [19].

There has been a rich line of work on graph property testing in the query model (see surveys [38, 20]) and the goal there is to design fast algorithms that make as few queries as possible. The query models that are mostly related to ours are bounded degree model and general graph model. In particular, our definition of ε-far is adapted from the general graph model. Goldreich and Ron [22] initiated the study of property testers in bounded degree graph model, and gave testers for connectivity, k-edge connectivity, $2,3$-vertex connectivity, cycle-freeness, Eulerianity. Testing k-vertex connectivity in bounded degree graphs for arbitrary constant k was given in [42]. These testers have later been generalized to general graph model [36, 35]. Testing bipartiteness in planar graphs was studied in [12].

After having submitted the paper, we became aware that Hossein Jowhari [25] has independently studied the problem of estimating the number of connected components and provided similar results as ours, while he did not consider the streaming property testers considered here.

2 Preliminaries

2.1 Notations

We use $[n] = \{1, \cdots, n\}$ to denote the vertex set of the graph G defined by the stream, and let m denote the number of edges of G. For an undirected graph $G = ([n], E)$ and a vertex $i \in [n]$, we let $\Gamma(i)$ denote all the neighbors of i. For a set $C \subseteq [n]$, let $\Gamma(C)$ denote the set of vertices in $V \setminus C$ that have at least one neighbor in C, that is, $\Gamma(C) = \cup_{i \in C} \Gamma(i) \setminus C$. Let $E(C, V \setminus C)$ denote the set of edges crossing C and $V \setminus C$. We will use $G[C]$ to denote the subgraph induced by C.

For each vertex i, we define two vectors $\mathbf{\Delta}^i \in \{-1, 0, 1\}^{\binom{n}{2}}$ and $\mathbf{\Lambda}^i \in \{0, 1\}^n$ to encode the neighborhood information of i as follows:

$$\mathbf{\Delta}^i_{j,k} = \begin{cases} 1 & \text{if } i = j < k \text{ and } (j,k) \in E \\ -1 & \text{if } j < k = i \text{ and } (j,k) \in E \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{\Lambda}^i_j = \begin{cases} 1 & \text{if } j \in \Gamma(i) \text{ or } j = i \\ 0 & \text{otherwise} \end{cases}$$

By simple induction arguments, it is easy to prove that for any vertex set $C \subset V$, the nonzero entries in the vector $\mathbf{\Delta}^C := \sum_{i \in C} \mathbf{\Delta}^i$ corresponds to the edges between C and its complement $V \setminus C$. The nonzero entries in $\sum_{i \in C} \mathbf{\Lambda}^i$ corresponds exactly to vertices in $C \cup \Gamma(C)$.

2.2 Linear sketches

Linear sketch (or sketch for short) is a powerful tool widely used in the streaming model and other areas. Given a large vector $\mathbf{x} \in \mathbb{R}^n$, we want to construct a small sketch $\mathcal{L}(\mathbf{x})$, from which certain properties of $\mathbf{x}$ can be recovered. We call $\mathcal{L}$ a linear sketch if $\mathcal{L}(\mathbf{x}+\mathbf{y}) = \mathcal{L}(\mathbf{x}) + \mathcal{L}(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y}$, and this additive property make it trivial to implement linear sketches in the

dynamic streaming model. As in the previous works, we will use linear sketches as our main tool.

AGM sketch. We will use a dynamic streaming algorithm for constructing a spanning forest of a graph by Ahn, Guha and McGregor [2], which is summarized as follows.

▶ **Theorem 1** (AGM sketch [2]). *There exists a single-pass sketch-based dynamic streaming algorithm that uses $O(n \log^3 n)$ space, and recovers a spanning forest of the graph with probability 0.99. The recovery time of the algorithm is $\tilde{O}(n)$, and the update time is $\operatorname{poly}\log n$.*

AMS sketch. To check whether the input vector $\mathbf{x}$ is $\mathbf{0}$ or not, one can simply maintain a constant approximation of its *second frequency moment*, that is $F_2(\mathbf{x}) := \sum_i x_i^2$, which can be done in $O(\log n)$ space by using the classical AMS sketch that was introduced by Alon, Matias and Szegedy [4].

Exact k-sparse recovery. We call a vector k-sparse if $|\mathrm{supp}(\mathbf{x})| \leq k$. Given a non-zero vector $\mathbf{x} \in \mathbb{R}^n$, the goal here is to recover $\mathbf{x}$ if $\mathbf{x}$ is k-sparse, otherwise outputs **Fail**. We have the following result from [37].

▶ **Lemma 2** ([37]). *There exists an $O(k \log n \log_k \delta^{-1})$ space sketch-based algorithm that takes as input a non-zero vector $\boldsymbol{x} \in \mathbb{R}^n$, and with probability $1-\delta$, recovers $\boldsymbol{x}$ if $\boldsymbol{x}$ is k-sparse, otherwise outputs **Fail**. The update time is $O(\operatorname{poly}\log n)$ and the recovery time is $O(k \cdot \operatorname{poly}\log n)$.*

3 Estimating the number of connected components and MST weight

In this section, we present and analyze our algorithms for estimating the number of the connected components in a graph and $(1+\varepsilon)$-approximating the weight of the MST.

3.1 Estimating the number of connected components

Our first observation is that, to estimate the number of connected components within additive error εn, we can simply ignore all the large components (see also [8]). In particular, the number of components of size larger than $\Omega(1/\varepsilon)$ is at most $O(\varepsilon n)$. Thus it will be sufficient to estimate the number of components of small size, for which we have the following theorem.

▶ **Theorem 3.** *For any constant $t \geq 1$, there exists a one-pass dynamic streaming algorithm that uses $O(e^t n^{1-\varepsilon} \cdot \operatorname{poly}\log n)$ space and post-processing time to estimates the number of connected components of size at most $1/\varepsilon$ within an additive error $\varepsilon^t n$. The update time is $O(\operatorname{poly}\log n)$.*

By invoking Theorem 3 with parameter $\varepsilon' = (1-\varepsilon^q)\varepsilon$ and $t = (q+1)$, we get an estimator for the number of connected components of size smaller than $1/\varepsilon'$ within additive error at most $\varepsilon^{q+1} n$. Since the number of components of size at least $1/\varepsilon'$ is at most $\varepsilon' n = \varepsilon n - \varepsilon^{1+q} n$, the estimator also approximates the total number of connected components within additive error at most εn. The space of the algorithm is $\tilde{O}(e^{q+1} n^{1-\varepsilon+\varepsilon^{q+1}})$, and we have the following result.

▶ **Theorem 4.** *Let $q \geq 1$ be a constant. There exists a one-pass dynamic streaming algorithm that with constant success probability, estimates the number of connected components of a graph within an additive error εn in $O(e^{q+1} n^{1-\varepsilon+\varepsilon^{q+1}} \cdot \operatorname{poly}\log n)$ space and post-processing time.*

Algorithm 1 `EstimateNumSCC`

1: Sample each vertex with probability $p := (\varepsilon^{2t}n/16)^{-\varepsilon}$. If more than $16np$ vertices are sampled, then abort and output **Fail**. Let S denote the set of sampled vertices.
2: Maintain an AGM sketch of $G[S]$ using Theorem 1.
3: For each $v \in S$, maintain an AMS sketch $AMS(\mathbf{\Delta}^v)$, sketching the neighborhood of v in G.
4: **Post-Processing:**
5: Use the AGM sketch to recover a spanning forest F of the induced graph $G[S]$ using Theorem 1.
6: For each component $C \in F$, estimate $F_2(\mathbf{\Delta}^C)$ using the AMS sketch $AMS(\mathbf{\Delta}^C) = \sum_{v\in C} AMS(\mathbf{\Delta}^v)$, and set $X_C = 1$ if $F_2 = 0$, otherwise set $X_C = 0$. For each $1 \le \ell \le \frac{1}{\varepsilon}$, let $X_\ell := \sum_{C:|C|=\ell} X_C$.
7: Output $Y := \sum_{\ell \le \frac{1}{\varepsilon}} \frac{X_\ell}{p^\ell}$.

Now we give the proof of Theorem 3. Recall that the vectors $\mathbf{\Delta}^C$ encode the information of the number of edges between C and $V \setminus C$.

Proof of Theorem 3. Let $\mathrm{scc}(G)$ denote the number of connected components of size at most $1/\varepsilon$ in G. Our algorithm for estimating $\mathrm{scc}(G)$ is as follows. We first sample each vertex with probability $p := (\varepsilon^{2t}n/16)^{-\varepsilon}$. Let S be the set of sampled vertices. We then use the AGM sketch from Theorem 1 to maintain a spanning forest F of the subgraph induced by S. Then for each component C in F, we test whether C is actually a connected component in G by testing whether the vector $\mathbf{\Delta}^C := \sum_{v\in C} \mathbf{\Delta}^v$ is $\mathbf{0}$, which can be done by the AMS sketch. If $\mathbf{\Delta}^C = \mathbf{0}$, we set $X_C = 1$, otherwise set $X_C = 0$. Our estimator is then defined as $\sum_C \frac{X_C}{p^{|C|}}$, where C ranges over all components of F with size at most $\frac{1}{\varepsilon}$. See Algorithm 1 for the details.

Note that the algorithm samples at most $16np = O(\varepsilon^{-2t\varepsilon} \cdot n^{1-\varepsilon})$ vertices and we maintained an AGM sketch on $G[S]$ and an AMS sketch for each sampled vertex, which imply that the space complexity of the algorithm is $O(\varepsilon^{-2t\varepsilon} n^{1-\varepsilon} \cdot \mathrm{poly}\log n)$. By simple calculus, for any ε, it holds that $\varepsilon^{-2\varepsilon} \le e^{2/e} < e$, so the space is at most $\tilde{O}(e^t n^{1-\varepsilon})$. The post-processing time is near linear in the space, and the update time is $O(\mathrm{poly}\log n)$.

Now we prove the correctness of the above algorithm. First we note that the expected number of sampled vertices in Step (1) is np, and thus by Markov inequality, the probability that more than $16np$ vertices are sampled is at most $\frac{1}{16}$. Also note that with probability at least $1 - \frac{1}{16}$, the AGM sketch returns a true spanning forest of $G[S]$. In addition, since the number of components in F is at most n, we will query the AMS sketch at most n times. Thus if we set the error probability of the AMS sketch to be $\frac{1}{16n}$ (with an extra $\log n$ factor in space), then with probability at least $1 - \frac{1}{16}$, all invocations of AMS sketches for testing if $\mathbf{\Delta}^C = \mathbf{0}$ will give the correct answer. Conditioned on this event, X_ℓ defined in Step (6) is exactly the number of connected components B of size ℓ in G such that all vertices in B are sampled out, which is true since for any component $C \in F$, $F_2(\mathbf{\Delta}^C) = \mathbf{0}$ if and only if C is a connected component in G.

Let $B_1, \cdots, B_{\mathrm{scc}(G)}$ be the connected components of size at most $\frac{1}{\varepsilon}$ of G. For any integer $\ell \le \frac{1}{\varepsilon}$, let $\mathcal{B}_\ell$ denote the set of connected components of size ℓ in G, that is, $\mathcal{B}_\ell = \{B_i : 1 \le i \le \mathrm{scc}(G), |B_i| = \ell\}$. Let $b_\ell := |\mathcal{B}_\ell|$. Note that $\mathrm{scc}(G) = \sum_{\ell \le \frac{1}{\varepsilon}} b_\ell$. For any set B, let Z_B denote the indicator random variable that all the vertices in B have been sampled. Note that $\Pr[Z_B = 1] = p^{|B|}$. Now by the above argument, $X_\ell = \sum_{B\in\mathcal{B}_\ell} Z_B$,

and $\mathrm{E}[X_\ell] = b_\ell \cdot p^\ell$. Furthermore, we have $Y = \sum_{\ell \le \frac{1}{\varepsilon}} \frac{X_\ell}{p^\ell} = \sum_{\ell \le \frac{1}{\varepsilon}} \frac{\sum_{B \in \mathcal{B}_\ell} Z_B}{p^\ell}$, and thus $\mathrm{E}[Y] = \sum_{\ell \le \frac{1}{\varepsilon}} b_\ell = \mathrm{scc}(G)$.

Note that all Z_{B_i}'s are mutually independent for all i, so it holds that

$$\begin{aligned}\mathrm{Var}[Y] &= \sum_{\ell \le \frac{1}{\varepsilon}} \frac{\sum_{B \in \mathcal{B}_\ell} \mathrm{Var}[Z_B]}{p^{2\ell}} = \sum_{\ell \le \frac{1}{\varepsilon}} \frac{b_\ell (p^\ell - p^{2\ell})}{p^{2\ell}} \le \sum_{\ell \le \frac{1}{\varepsilon}} \frac{b_\ell}{p^\ell} \\ &\le \frac{\sum_{\ell \le \frac{1}{\varepsilon}} b_\ell}{p^{1/\varepsilon}} = \frac{\mathrm{scc}(G)}{p^{1/\varepsilon}} \le \frac{n}{p^{1/\varepsilon}} = \varepsilon^{2t} n^2/16, \qquad (1)\end{aligned}$$

where we use the fact that $\mathrm{scc}(G) \le n$, and $p = (\varepsilon^{2t} n/16)^{-\varepsilon}$. Then by Chebyshev's inequality,

$$\Pr[|Y - \mathrm{scc}(G)| \ge \varepsilon^t n] = \Pr[|Y - \mathrm{E}[Y]| \ge \varepsilon^t n] \le \frac{\mathrm{Var}[Y]}{\varepsilon^{2t} n^2} \le 1/16.$$

By the union bound, the algorithm will succeed with probability at least $\frac{2}{3}$. ◀

3.2 Approximating the weight of minimum spanning tree

We use the previous algorithm on estimating the number of connected components to approximate the weight of a minimum spanning tree of a weighted graph. Let $W \ge 2$ be an integer, G be a connected graph with integer edge weights from $[W] := \{1, \cdots, W\}$, and $c(\mathrm{MST})$ be the weight of an MST of G. For any $1 \le \ell \le W$, let $G^{(\ell)}$ denote the subgraph of G consisting of all edges of weight at most ℓ. Let $\mathrm{cc}^{(\ell)}$ denote the number of connected components of $G^{(\ell)}$. Chazelle et al. [8] give the following lemma relating the weight of MST to the number of connected components of $G^{(\ell)}$.

▶ **Lemma 5** ([8]). *It holds that* $c(MST) = n - W + \sum_{\ell=1}^{W-1} cc^{(\ell)}$.

For a connected graph with integer edge weights, the weight of any MST is at least $n-1$, so it is sufficient to estimate $cc^{(\ell)}$ within an additive error of $\varepsilon n/(W-1)$ for each ℓ. To do this, we can simply run $W-1$ parallel instances of Theorem 4, each of which sketches a subgraph $G^{(\ell)}$. Then the space of the algorithm will be $\tilde{O}(W n^{1-\frac{\varepsilon}{W-1}})$.

▶ **Theorem 6.** *Let $t \ge 1$ be any constant. There exists a single-pass dynamic streaming algorithm that uses space and post-processing time $O(e^t W n^{1-\frac{\varepsilon}{W-1}+\frac{\varepsilon^t}{(W-1)^t}} \operatorname{poly}\log n)$ to compute a $(1+\varepsilon)$-approximation of the weight of the MST.*

We remark that Ahn et al. [2] have given a dynamic streaming algorithm for this problem for any graph with maximum edge weight upper bounded by $O(\mathrm{poly}(n))$, and their algorithm uses space $O(n \cdot \operatorname{poly}\log n)$. Our algorithm uses $o(n)$ space for any connected graph with maximum edge weight bounded by $o(\log n)$ (for constant ε), which improves the algorithm of [2] in this setting. We also note that $\Omega(n)$ space is necessary for estimating the weight of MST for graphs with maximum edge weight at least $c \log n$ for constant ε and some large universal constant c (see Theorem 12). Finally, we remark that the algorithm can also be extended to the setting where non-integral weights are allowed (see [8] for more details).

4 Dynamic streaming testers

In this section, we give our streaming testers for k-edge connectivity and cycle-freeness. Due to space constraints, we present the testers for k-vertex connectivity, Eulerianity and planar graph bipartiteness in the full version of the paper.

Algorithm 2 TestConnectivity

1: Sample each vertex with probability $p := (\varepsilon n/10)^{-\varepsilon}$. If more than $16np$ vertices are sampled, abort and output **Fail**. Let S denote the set of sampled vertices.
2: For each $v \in S$, maintain an AMS sketch $AMS(\mathbf{\Delta}^v)$, sketching the neighborhood of v in G.
3: Maintain an AGM sketch of $G[S]$ using Theorem 1.
4: **Post-Processing:**
5: Use the above sketch to construct a spanning forest F of $G[S]$ as guaranteed by Theorem 1.
6: For each connected component $C \in F$, estimate $F_2(\mathbf{\Delta}^C)$ using the AMS sketch $AMS(\mathbf{\Delta}^C) = \sum_{v \in C} AMS(\mathbf{\Delta}^v)$. If the answer $\tilde{F}_2 = 0$, **Reject**.
7: **Accept**.

4.1 Testing k-edge connectivity

A graph is k-edge connected if the minimum cut of the graph has size at least k. We start from the simplest case, i.e., $k = 1$, which is equivalent to the problem of testing connectivity.

4.1.1 Connectivity

It is clear that if G is ε-far from being connected, one must add at least εm edges to make it connected, which implies that there are at least $\varepsilon m + 1$ connected components in G [22, 36]. Therefore, we can also solve this by estimating the number connected components by setting the error parameter appropriately, however, by a more careful analysis, we can improve this by a factor of $O(n^{O(\varepsilon)})$.

▶ **Theorem 7.** *There exists a dynamic streaming tester for* 1*-edge connectivity that runs in* $\tilde{O}(n^{1-\varepsilon})$ *post-processing time and space.*

Proof. First observe that one can simply reject the input graph if $m < n - 1$, since in this case, the graph is disconnected. Thus, in the following we assume $m \geq n - 1$ and our tester is described in Algorithm 2.

It is easy to see that Algorithm 2 only use $\tilde{O}(|S|)$ space, which is bounded by $\tilde{O}(np) = \tilde{O}(\varepsilon^{-\varepsilon} n^{1-\varepsilon}) = \tilde{O}(n^{1-\varepsilon})$. The post-processing time is nearly linear in the size of S, since the AGM algorithm needs $\tilde{O}(|S|)$ post-processing time, and we invoke at most $|S|$ AMS queries, each of which takes $\tilde{O}(1)$ time. The update time is $\operatorname{poly}\log n$.

For the correctness of the algorithm, we condition on the event that the number of sampled vertices is at most $16np$, which occurs with probability at least $1 - \frac{1}{16}$, and on the event that the spanning forest F is constructed correctly, which occurs with probability 0.99. By setting the error probability of the AMS sketch to be $1/n^2$ (with an extra $\log n$ factor in space), with probability 0.99, all the answers from AMS sketches are all correct, and we also condition on this.

If G is connected, then it will always be accepted, since for each $C \in F$, $\mathbf{\Delta}^C \neq \mathbf{0}$, and conditioned on the correctness of the AMS sketch, $\tilde{F}_2$ will never be 0. On the other hand, if the graph is ε-far from being connected, the number of connected components in G, denoted as $\operatorname{cc}(G)$, is at least $1 + \varepsilon m \geq \varepsilon n$. Let $B_1, \cdots, B_{\operatorname{cc}(G)}$ denote all connected components in G. Let $p_i = p^{|B_i|}$ for $1 \leq i \leq \operatorname{cc}(G)$. Using the inequality $1 - x \leq e^{-x}$ for all x, the probability that none of the components is entirely sampled out is $(1-p_1)\cdot(1-p_2)\cdot\cdots\cdot(1-p_{\operatorname{cc}(G)}) \leq e^{-\sum_i p_i}$. Then by the AM-GM inequality, this probability is at most

$$e^{-\operatorname{cc}(G)\cdot(\prod_i p_i)^{1/\operatorname{cc}(G)}} = e^{-\operatorname{cc}(G)\cdot p^{n/\operatorname{cc}(G)}} \leq e^{-\operatorname{cc}(G)\cdot p^{1/\varepsilon}} \leq e^{-\varepsilon n \cdot p^{1/\varepsilon}} \leq 1/16,$$

where we use the fact that $p = (\varepsilon n/10)^{-\varepsilon}$ and $\mathrm{cc}(G) \geq \varepsilon n$. So the probability that at least one of the components is sampled out is at least 15/16. Conditioned on this, $F_2(\boldsymbol{\Delta}^C) = 0$ for some component in $G[S]$ and the algorithm will output **Reject**. By union bound, our algorithm will succeed with probability $1 - \frac{1}{16} - 0.01 - 0.01 - \frac{1}{16} > 3/4$. ◀

4.1.2 k-edge connectivity: $k \geq 2$

By using a slightly more involved argument and replacing AMS sketches with $(k-1)$-sparse recovery sketches, we can generalize the above idea to testing k-edge connectivity for $k \geq 2$. We have the following theorem on testing k-edge connectivity. See the full version for the proof.

▶ **Theorem 8.** *Let $k \leq O(n^{\varepsilon/(1+\varepsilon)})$. There exists a single-pass dynamic streaming tester for k-edge connectivity with post-processing time and space $O(k^{1+\varepsilon} \cdot n^{1-\varepsilon} \cdot \mathrm{poly}\log n)$.*

4.2 Testing cycle-freeness

Now we consider the problem of testing cycle-freeness, which is equivalent to testing if the graph is a forest. Let $\mathrm{cc}(G)$ denote the number of connected components of the input graph G. Let $B_1, \cdots, B_{\mathrm{cc}(G)}$ be the connected components in G. Note that if G is cycle-free, then for each $i \leq \mathrm{cc}(G)$, $|E(B_i)| = |B_i| - 1$, and thus the total number of edges in G is

$$m = \sum_{i=1}^{\mathrm{cc}(G)} |E(B_i)| = \sum_{i=1}^{\mathrm{cc}(G)} (|B_i| - 1) = n - \mathrm{cc}(G),$$

that is, $\mathrm{cc}(G) = n - m$. If G is ε-far from being cycle-free, i.e., one has to delete more than εm edges to make it cycle-free, then $\mathrm{cc}(G) > n - m + \varepsilon m$. Therefore, to test cycle-freeness of a graph, it will be sufficient to approximate the number of connected components with additive error $\varepsilon m/2$. One may try to directly invoke Algorithm 1 with parameter $\varepsilon' = \frac{\varepsilon m}{2n}$. However, m could be much smaller than n and we do not know m in advance. We overcome this obstacle by a case analysis.

▶ **Theorem 9.** *There exists a single-pass dynamic streaming algorithm that tests cycle-freeness of a graph with space and post-processing time $O(n^{1-\varepsilon+\varepsilon^2} \cdot \mathrm{poly}\log n)$.*

Proof. Note that if $m > n - 1$, then the graph must contain at least one cycle, and thus we can safely reject the graph. In the following, we assume that $m \leq n - 1$. Our algorithm for testing cycle-freeness depends on the construction of AGM sketch, in which each vertex u maintains a linear sketch of $\boldsymbol{\Delta}^u$ (denoted as $\mathcal{A}(\boldsymbol{\Delta}^u)$). Each such sketch has size $\mathrm{poly}\log n$ and the property that $\mathcal{A}(\mathbf{0}) = \mathbf{0}$ (it consists of $O(\log n)$ l_0-samplers, see [2] for details). Our main idea is to maintain a sparse recovery sketch for the AGM sketch (i.e. a composition of sparse recovery sketch and AGM sketch). Now we describe our algorithm as follows.

Note that the space used by the algorithm is $\max\{\tilde{O}(np), k \cdot \mathrm{poly}\log n\} = \tilde{O}(n^{1-\varepsilon+\varepsilon^2})$, and the post-processing time is near linear in space. Now we prove the correctness of the algorithm. We define $G' \subseteq G$ to be a subgraph which consists of all the vertices of positive degree. Let $n' = |G'|$. Note that $m \geq n'/2$. If $n' \leq n^{1-\eta}$, then the vector $\boldsymbol{\Upsilon}$ is $\tilde{O}(n^{1-\eta})$-sparse, since for all isolated vertices u, we have $\mathcal{A}(\boldsymbol{\Delta}^u) = \mathbf{0}$, and thus we can recover the entire $\boldsymbol{\Upsilon}$ exactly. Then by Step (2) and Theorem 1, we can get the exact number of components of G'. Since the number of vertices of G' is $|Y|$, and $\lambda = m$ is the total number of edges, then the graph is cycle-free if and only if $\tilde{c}_1 = |Y| - \lambda$.

Algorithm 3 TestCycleFreeness

1: Maintain a count λ of the number of edges.
2: Let $\eta = \varepsilon/(1+\varepsilon+\varepsilon^2)$. Let $k = n^{1-\eta}\text{poly}\log n$. Maintain an exact k-sparse recovery sketch $\mathcal{S}$ of the vector $\boldsymbol{\Upsilon} := (\mathcal{A}(\boldsymbol{\Delta}^u))_{u\in V}$ using Lemma 2.
3: Run Algorithm 1 with parameter $p = (2^{2t}n^{1-\eta}/16)^{-\eta}$, while in step (6) of Algorithm 1, ignore all the isolated vertices that are sampled out (i.e., set $X_C = 0$ whenever $|C| = 1$).
4: **Post-Processing:**
5: Recover $\boldsymbol{\Upsilon}$ from $\mathcal{S}$.
6: **if** The recovery does not fail **then**
7: Use $\boldsymbol{\Upsilon}$ to construct a spanning forest on vertex set $Y := \{u : \mathcal{A}(\boldsymbol{\Delta}^u) \neq \mathbf{0}\}$ using Theorem 1. Let $\tilde{c}_1$ denote the number of connected components of this forest. If $\tilde{c}_1 = |Y| - \lambda$, **Accept**; otherwise, **Reject**.
8: **else**
9: Let $\tilde{c}_2$ be the resulting estimator of Algorithm 1 in Step 3. If $\tilde{c}_2 \leq n - (1-\varepsilon-\frac{\varepsilon^3}{4})\lambda$, **Accept**; otherwise, **Reject**.
10: **end if**

If $n' > n^{1-\eta}$, then by Theorem 3, $\tilde{c}_2$ is an estimator for the number of components in G' of size smaller than $1/\eta$ with additive error $\eta^t\sqrt{n'n^{1-\eta}}$. This follows by the upper bound $\eta^{2t}n^{1-\eta}n'/16$ of the variance of the estimator (which can be shown similarly to inequality (1) in Section 3) and the Chebyshev's inequality. Now note that the additive error is at most $\eta^t n' \leq \varepsilon^3 m/8$ for some constant t since $n' > n^{1-\eta}$ and $m \geq n'/2$. Let L be the number of components in G' of size larger than $1/\eta$, then $-\varepsilon^3 m/8 \leq \text{cc}(G') - \tilde{c}_2 \leq L + \varepsilon^3 m/8$ holds with high probability. Conditioned on this, Step (8) outputs the correct answer if $L + \varepsilon^3 m/8 + \varepsilon^3 m/8 = L + \varepsilon^3 m/4 < \varepsilon m$. Now if $L < \varepsilon m/4$, we are done. If $L \geq \varepsilon m/4$, then by our choice of η and the fact that $m \geq L \cdot (1/\eta - 1)$, $\varepsilon m \geq L + L\varepsilon^2 \geq L + \varepsilon^3 m/4$. This completes the proof of the theorem. ◀

5 Lower bounds

In this section we present lower bounds, which hold in the insertion-only model. Our proofs are based on the reductions to the *Boolean Hidden Hypermatching (BHH)* problem (See [41]), which are in the same spirit as the lower bound proof for the *Cycle Counting* problem in [41]. We first give the definition of the boolean hidden hypermatching problem.

▶ **Definition 10** (BHH_n^t)**.** In this problem, Alice gets a boolean vector $x \in \{0,1\}^n$, where $n = 2kt$ for some integer k. Bob gets a partition (or hypermatching) of the set $[n]$, $\{m_1, \cdots, m_{n/t}\}$, where the size of each m_i is t, and a vector $w \in \{0,1\}^{n/t}$. For convenience, we will also use the corresponding n-dimensional boolean indicator vector M_i to represent m_i, and let M be a $n/t \times n$ matrix, the i row of which is M_i. The promise of the input is either $Mx + w = \mathbf{1}$ or $Mx + w = \mathbf{0}$, where all the operations are modulo 2. The goal of the problem is to output 1 when $Mx + w = \mathbf{1}$, and output 0 otherwise.

▶ **Theorem 11** ([41])**.** *The randomized one-way communication complexity of BHH_n^t when $n = 2kt$ for some integer $k \geq 1$ is $\Omega(n^{1-1/t})$.*

Our lower bounds will be built upon the following basic construction.

Construction of $G(x, M)$. Given vector x and matrix M respectively, Alice and Bob construct a bipartite graph $G(x, M) = (U, V, E)$, where $U = \{u_1, \cdots, u_{2n}\}$ and $V =$

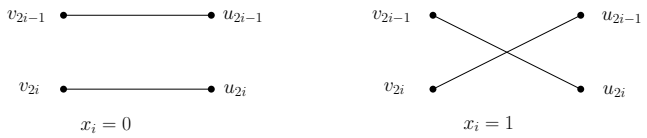

Figure 1 Parallel (left) and crossing (right) matching according to the value of x_i.

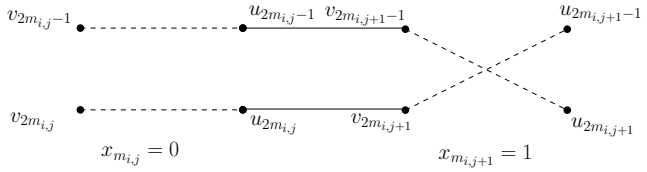

Figure 2 Bob connects $(u_{2m_{i,j}-1}, v_{2m_{i,j+1}-1})$ and $(u_{2m_{i,j}}, v_{2m_{i,j+1}})$ for each $j \in [t-1]$.

$\{v_1, \cdots, v_{2n}\}$, as follows. Given $x \in \{0,1\}^n$, Alice adds a perfect matching between U and V. For each $i \in [n]$, if $x_i = 0$, she adds two parallel edges (u_{2i-1}, v_{2i-1}) and (u_{2i}, v_{2i}); otherwise if $x_i = 1$, she adds two crossing edges (u_{2i-1}, v_{2i}) and (u_{2i}, v_{2i-1}) (see Figure 1).

Given M, Bob will do the following. For each $i \in [n/t]$ and the hyperedge $m_i \subset [n]$ (that corresponds to the ith row M_i), we use $m_{i,j} \in [n]$ to denote the jth element in m_i and we let $S_i := \{w | w = v_{2m_{i,j}-1}$ or $v_{2m_{i,j}}$ or $u_{2m_{i,j}-1}$ or $u_{2m_{i,j}}, j \in [t]\}$. For each $i \in [n/t]$ and $j \in [t-1]$, Bob adds two edges $(u_{2m_{i,j}-1}, v_{2m_{i,j+1}-1})$ and $(u_{2m_{i,j}}, v_{2m_{i,j+1}})$ (See Figure 2).

Observe that the edges added by Alice and Bob form two paths p_{2i-1}, p_{2i} over vertex set S_i, where p_{2i-1} starts from $v_{2m_{i,1}-1}$ and p_{2i} starts from $v_{2m_{i,1}-1}$ for each i. The entire graph $G(x, M)$ consists of $2n/t$ disjoint paths $\{p_1 \cdots, p_{2n/t}\}$.

Note that $G(x, M)$ has the following property. Based on the value of $(Mx)_i$, we have: 1) if $(Mx)_i = 0$, then p_{2i-1} is a path from $v_{2m_{i,1}-1}$ to $u_{2m_{i,t}-1}$ and p_{2i} is a path from $v_{2m_{i,1}}$ to $u_{2m_{i,t}}$; 2) if $(Mx)_i = 1$, then p_{2i-1} is a path from $v_{2m_{i,1}-1}$ to $u_{2m_{i,t}}$ and p_{2i} is a path from $v_{2m_{i,1}}$ to $u_{2m_{i,t}-1}$.

5.1 Minimum spanning tree

▶ **Theorem 12.** *In the insertion-only model, if all edges of the graph have weights in $[W]$, any algorithm that $(1 \pm \varepsilon)$-approximates the weight of the MST must use $\Omega(n^{1-\frac{4\varepsilon}{W-1}})$ bits of space.*

Proof. Given x and M, Alice and Bob first construct the graph $G(x, M)$ as describe above. Next Bob adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}-1})$ and $(u_{2m_{i,t}}, v_{2m_{i,1}})$ if $w_i = 0$; adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}})$ and $(u_{2m_{i,t}}, v_{2m_{i,1}-1})$ if $w_i = 1$. The weight of all the edges added so far is 1. Next Bob places edges $(v_{2m_{i,t}}, v_{2m_{i+1,1}})$ with weight 1 for $i = 1, \cdots, n/t - 1$ and edges $(v_{2m_{i,t}}, u_{2m_{i,t}})$ with weight W for each $i \in [n/t]$, so that the graph become connected. By similar argument as above, if $Mx + w = \mathbf{0}$, all the edges $(v_{2m_{i,t}}, u_{2m_{i,t}})$ must be picked in any minimum spanning tree, since each of these edges forms a cut, and thus the weight of any MST is $nW/t + 4n - n/t - 1 = 4n\varepsilon + 4n - 1$, where we set $t = (W-1)/4\varepsilon$. On the other hand, when $Mx + w = \mathbf{1}$, the weight of the MST is $4n - 1$, since in this case, the graph is already connected without those edges with weight W. So if the algorithm can compute an $(1+\varepsilon)$-approximation of the weight of the minimum spanning tree, it solves the BHH_n^t problem. This completes the proof. ◀

5.2 Testing connectivity

▶ **Theorem 13.** *In the insertion-only model, to distinguish whether a graph of $4n$ vertices is connected or $\frac{1}{8t+1}$-far from being connected, any algorithm must use $\Omega(n^{1-1/t})$ bits of space.*

Proof. Given x and M, Alice and Bob first construct the graph $G(x, M)$. Next Bob adds another set of edges based on vector w. If $w_i = 0$, he adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}-1})$ and $(u_{2m_{i,t}}, v_{2m_{i,1}})$; if $w_i = 1$, he adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}})$ and $(u_{2m_{i,t}}, v_{2m_{i,1}-1})$. So when $(Mx)_i + w_i = 0$, p_{2i-1} and p_{2i} become 2 disjoint cycles. On the other hand, when $(Mx)_i + w_i = 1$, p_{2i-1} and p_{2i} together form a larger cycle. Now Bob places $(v_{2m_{i,t}}, v_{2m_{i+1,1}})$ in E for $i = 1, \cdots, n/t - 1$ which connect p_{2i} with $p_{2(i+1)}$ for all $i \in [n/t - 1]$, i.e. all the paths in $G(x, M)$ with even indices become a connected component. The total number of edges is $8n + n/t$. When $Mx + w = \mathbf{0}$, the graph has $n/t + 1$ components which is $\frac{1}{8t+1}$-far from connected; when $Mx + w = \mathbf{1}$ the graph is connected. So if a streaming algorithm can distinguish whether a graph of size $4n$ is connected or $1/8t$-far from being connected, it solves BHH_n^t, since Alice can first run the algorithm on her part of the graph and send the memory to Bob, and then Bob continues to run the algorithm on his part and output the answer. Therefore, the communication lower bound of BHH_n^t implies a space lower bound of testing connectivity. ◀

5.3 Testing cycle-freeness

As in the proof of Theorem 13, given x and M, Alice and Bob first construct $G(x, M)$. Then, for $i \in [n/t]$, Bob adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}-1})$ if $w_i = 0$; adds $(u_{2m_{i,t}-1}, v_{2m_{i,1}})$ if $w_i = 1$. The total number of edges is less than $8n$. Through similar arguments, it is easy to verify that if if $Mx + w = \mathbf{0}$, the graph has exactly n/t cycles and n/t paths, which is $1/8t$-far from cycle-free. On the contrary, if $Mx + w = \mathbf{1}$, the graph has n/t paths and no cycle. So if an algorithm can distinguish whether a graph of size $4n$ is cycle-free or $1/8t$-far from cycle-free, it solves BHH_n^t.

▶ **Theorem 14.** *In the insertion-only model, any algorithm that can distinguish whether a graph of $4n$ vertices is cycle-free or $1/8t$-far from being cycle-free, must use $\Omega(n^{1-1/t})$ bits of space.*

5.4 Testing bipartiteness of planar graphs

Alice and Bob first construct the graph $G(x, M)$. Next, for each $i \in [n/t]$, Bob adds edges $(v_{2m_{i,1}-1}, \xi_1)$ and $(v_{2m_{i,1}}, \xi_2)$, where ξ_1, ξ_2 are new vertices. For $i \in [n/t]$, Bob also adds $(u_{2m_{i,t}-1}, \xi_1)$ and $(u_{2m_{i,t}}, \xi_2)$ if $w_i = 0$; adds $(u_{2m_{i,t}-1}, \xi_2)$ and $(u_{2m_{i,t}}, \xi_1)$ if $w_i = 1$. For this problem we assume t is odd. So by similar arguments, we can easily verify that, if $Mx + w = \mathbf{0}$, the graph contains $2n/t$ edge-disjoint cycles of length $2t + 1$, and if $Mx + w = \mathbf{1}$, the graph has no odd cycle, and thus bipartite. The graph constructed is planar and has $4n + 2$ vertices and $8n + 4n/t$ edges, so we have the following lower bound for testing bipartiteness.

▶ **Theorem 15.** *In the insertion-only model, any algorithm that can distinguish whether a planar graph of $4n + 2$ vertices is bipartite or $\frac{1}{4t+2}$-far from being bipartite, must use $\Omega(n^{1-1/t})$ bits space.*

References

1. Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, pages 6–11, 2015.
2. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.
3. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.
4. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.
5. Sepelir Assadi, Sanjeev Khanna, Yang Li, and Grigory 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. SIAM, 2016. URL: `http://dl.acm.org/citation.cfm?id=2884435.2884528`.
6. Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *ACM Symposium on Theory of Computing*, 2015.
7. Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. *ESA*, 2015.
8. Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.
9. Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to dynamic graph streams. *SODA*, 2016.
10. Rajesh Chitnis, Graham Cormode, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1234–1251. SIAM, 2015.
11. Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005.
12. Artur Czumaj, Morteza Monemizadeh, Krzysztof Onak, and Christian Sohler. Planar graphs: Random walks and bipartiteness testing. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 423–432. IEEE, 2011.
13. Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009.
14. Hossein Esfandiari, Mohammad T Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof 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*, pages 1217–1233. SIAM, 2015.
15. Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P Woodruff. Applications of uniform sampling: Densest subgraph and beyond. *arXiv preprint arXiv:1506.04505*, 2015.
16. Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *Mathematical Foundations of Computer Science 2014*, pages 275–286. Springer, 2014.

17 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

18 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.

19 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 142–149. ACM, 2005.

20 Oded Goldreich. Introduction to testing graph properties. In *Property testing*, pages 105–141. Springer, 2011.

21 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

22 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32:302–343, 2002.

23 Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, pages 241–247. ACM, 2015.

24 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, pages 107–118, 1998.

25 Hossein Jowhari. Estimating the number of connected components in graph streams. Personal communication.

26 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 734–751. SIAM, 2014.

27 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating max-cut. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1263–1282. SIAM, 2015.

28 Michael Kapralov, Yin Tat Lee, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 561–570. IEEE, 2014.

29 Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 272–281. ACM, 2014.

30 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 367–376. ACM, 2015.

31 Christian Konrad. Maximum matching in turnstile streams. *ESA*, 2015.

32 Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

33 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu. Densest subgraph in dynamic graph streams. *MFCS*, 2015.

34 S Muthukrishnan. Data streams: Algorithms and applications. *Theoretical Computer Science*, 1(2):117–236, 2005.

35 Yaron Orenstein and Dana Ron. Testing eulerianity and connectivity in directed sparse graphs. *Theoretical Computer Science*, 412(45):6390–6408, 2011.

36 Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.

37 Eric Price. Efficient sketches for the set query problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 41–56. SIAM, 2011.

38 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends® in Theoretical Computer Science*, 5(2):73–205, 2010.

39 Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4):1562–1588, 2011.

40 Xiaoming Sun and David P. Woodruff. Tight bounds for graph problems in insertion streams. In *The 18th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'2015)*, 2015.

41 Elad Verbin and Wei Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 11–25. SIAM, 2011.

42 Yuichi Yoshida and Hiro Ito. Property testing on k-vertex-connectivity of graphs. In *Automata, Languages and Programming*, pages 539–550. Springer, 2008.