Analyzing Graph Structure via Linear Measurements

Kook Jin Ahn* Sudipto Guha* Andrew McGregor†

Abstract

We initiate the study of *graph sketching*, i.e., algorithms that use a limited number of linear measurements of a graph to determine the properties of the graph. While a graph on n nodes is essentially $O(n^2)$-dimensional, we show the existence of a distribution over random projections into d-dimensional "sketch" space ($d \ll n^2$) such that several relevant properties of the original graph can be inferred from the sketch with high probability. Specifically, we show that:

1. $d = O(n \cdot \text{polylog}\, n)$ suffices to evaluate properties including connectivity, k-connectivity, bipartiteness, and to return any constant approximation of the weight of the minimum spanning tree.
2. $d = O(n^{1+\gamma})$ suffices to compute graph sparsifiers, the exact MST, and approximate the maximum weighted matchings if we permit $O(1/\gamma)$-round adaptive sketches, i.e., a sequence of projections where each projection may be chosen dependent on the outcome of earlier sketches.

Our results have two main applications, both of which have the potential to give rise to fruitful lines of further research. First, our results can be thought of as giving the first *compressed-sensing style algorithms for graph data*. Secondly, our work initiates the study of *dynamic graph streams*. There is already extensive literature on processing massive graphs in the data-stream model. However, the existing work focuses on graphs defined by a sequence of inserted edges and does not consider edge deletions. We think this is a curious omission given the existing work on both dynamic graphs in the non-streaming setting and dynamic geometric streaming. Our results include the first dynamic graph semi-streaming algorithms for connectivity, spanning trees, sparsification, and matching problems.

*University of Pennsylvania. Supported by NSF Awards CCF-0644119 and IIS-0713267 and a gift from Google. {kookjin,sudipto}@seas.upenn.edu

†University of Massachusetts Amherst. Supported by NSF CAREER Award CCF-0953754. mcgregor@cs.umass.edu.

1 Introduction

In this paper we initiate the study of *graph sketching*, i.e., algorithms that use a limited number of linear measurements of a representation of a graph to determine the properties of the graph. While a graph on n nodes is essentially $O(n^2)$-dimensional, we show the existence of a distribution over random projections into d-dimensional "sketch" space ($d \ll n^2$) such that the sketch contains sufficient information to determine whether G satisfies a given property, e.g., whether the graph is connected, with high probability. The study of random linear projections arises in many contexts including dimensionality reduction results such as the Johnson-Lindenstrauss lemma [27], reconstructing sparse signals in compressed sensing [13], and estimating aggregate statistics in data streams [35]. Given the important role of graphs in modeling data interactions and relationships, it is perhaps surprising that there is not such a well-developed study of random linear projections for graph data.

We show that linear sketches exist for a variety of interesting graph properties including connectivity, bipartiteness, and the weight of the minimum spanning tree. Our algorithms run in near-linear time and the sketch matrices can be generated from small random seeds. In addition to algorithms that operate with precomputed projections, we also consider *adaptive sketching* where the linear measurements are performed in a limited number of rounds and the linear measurements performed in a round may depend on the outcomes of measurements in previous rounds. In the data stream setting this corresponds to multi-pass algorithms. While non-adaptivity is important for applications such as network monitoring, in other applications a limited degree of adaptation is accepted, e.g., in adaptive compressed sensing [23, 26] and 2-stage group testing [6, 22].

Dynamic Graph Steams. An important contribution of this paper is to initiate the study of processing *dynamic graph streams*. Motivated by the need to process massive graphs such as the web-graph and social networks, a rich body of work has developed that addresses the challenges of processing graphs in the data-stream model. In this model, an algorithm is presented with a stream of m edges on n nodes and the goal is to compute properties of the resulting graph given only sequential access to the stream and limited memory. The majority of work considers the *semi-streaming* model in which the algorithm is permitted $O(n \operatorname{polylog} n)$ memory [17, 35]. Recent results include algorithms for constructing graph sparsifiers [1, 30], spanners [14, 18], and matchings [2, 3, 16, 32, 39]. This includes both single-pass algorithms and algorithms that take multiple pass over the data. See McGregor [33] for an overview.

A curious omission in the existing work are algorithms that can support graphs where edges are both inserted and deleted.[1] This is an important consideration since many interesting graphs evolve with time, e.g., hyperlinks can be removed and tiresome friends can be de-friended. We note that dynamic geometric stream problems are well-studied [19, 25] and, in the non-streaming setting, there is considerable work on data structures for dynamic graph that can be quickly updated and queried, see e.g., results for connectivity [7, 24, 37, 38].[2]

[1] We note one exception by Cormode and Muthukrishnan [10]. However, while their paper uses multi-graphs as a motivation, the specific problems considered therein are restricted to the properties of the vector representing the degrees of the vertices rather than more general graph structure.

[2] The reader might also recall work on the "sparsification" technique for dynamic graphs algorithm introduced by Eppstein et al. [15] that may appear relevant to small-space dynamic streaming (as it did in insert-only graph streaming [18]). However, sparsification in this context relates to the construction of auxiliary sparse certificates that can be quickly updated. In general, the dynamic algorithms must still store all the data so that sparse certificates can

Our sketch algorithms are immediately applicable to dynamic graph streams and give the first results of this kind. Our single pass algorithms also apply in the distributed stream model [12] where the stream is partitioned over multiple locations and communication between the sites should be minimized. This follows because the linearity of the sketches ensures that by adding together the sketches of the partial streams, we get the sketch of the entire stream.

Compressed Sensing. Compressed sensing studies the problem of reconstructing a sparse signal $\mathbf{x}$ from a small number of linear measurements [13]. However, an extension called *functional compressed sensing* has been proposed, see [36]. Here the goal is to compute some function of $f(\mathbf{x})$ with only a few linear measurements. If f is the set of heavy-hitters then this reduces to the original (approximate) reconstruction problem. Our results can also be considered in terms of functional compressed sensing where $\mathbf{x}$ represents graph data and f is, e.g., the weight of the minimum spanning tree or a boolean function that evaluates to 1 if the graph is connected.

1.1 Our Results and Roadmap.

We start in Section 2 by discussing our approach and attempt to convey some of the inherent challenges. We also remind the reader of recent results in ℓ_p sampling that will be used as a primitive in our algorithms. In the following five sections we present our results. We group these results according to the degree of adaptivity that is required, or equivalently, the number of passes used by the resulting stream algorithms.

1. *Non-adaptive:* We show that $O(n \cdot \operatorname{polylog} n)$ non-adaptive linear measurements are sufficient to determine whether a graph is connected and identify a spanning forest. This yields a single-pass semi-streaming algorithm for dynamic connectivity. By performing local per-edge transformations, we use our connectivity result as the basis for single-pass semi-streaming algorithms for testing bipartiteness and estimating the weight of the minimum spanning tree. We also show that $O(k \cdot n \cdot \operatorname{polylog} n)$ non-adaptive measurements (implying a single-pass streaming algorithm using $O(k \cdot n \cdot \operatorname{polylog} n)$ space) are sufficient to determine whether the graph is k-edge connected. These results are presented in Section 3.

2. *Adaptive:* In our final set of algorithmic results we show that $O(n^{1+\gamma})$ linear measurements suffices to compute graph sparsifiers, the exact MST, and any constant approximation to the maximum weighted matchings if we permit $O(1/\gamma)$-round adaptive sketches. This yields $O(\log n/\log\log n)$-pass semi-streaming algorithms or a constant-pass algorithm if we are permitted slightly more space. These results are presented in Sections 4, 5, and 6 respectively.

Subsequent Developments. In subsequent work, we have improved our sparsification result and can show that sparsification is possible in the single-pass, semi-streaming model [4].

2 Computation in Sketch Space

In this section, we briefly discuss some of the ideas underlying our algorithmic results. In doing so, we also hope to convey some of the inherent challenges that we need to overcome. We will also summarize some existing results that we will require.

be reconstructed in the event of a deletion.

The basic philosophy that pervades many of our results is that, rather than processing the large input in its original form, we would rather first sketch the input and then perform the computation in the smaller sketch space. Simpler forms of the idea appear in earlier work such as Gilbert et al. [21] where the authors considered the problem of histogram construction. But in the context of graph computation, a richer set of algorithmic ideas can be explored. However, there are subtle issues regarding the operations that can be legitimately performed in sketch space. These arise because of the randomness of the sketches and relate to the "adaptivity" of the operations we wish to perform. We illustrate a couple of the issues using the example of using linear sketches for ℓ_p sampling. This is also an example that will play an important role in our algorithms.

Combing Sketches for ℓ_p Sampling. ℓ_p-sampling is a problem that has recently enjoyed considerable attention [11, 19, 20, 28, 34]. Consider a turnstile stream $S = \langle s_1, \ldots, s_t \rangle$ where each $s_i \in (u_i, \Delta_i) \in [n] \times \mathbb{R}$ and the aggregate vector $\mathbf{x} \in \mathbb{R}^n$ defined by this stream, i.e., $x_i = \sum_{j:u_j=i} \Delta_i$.

Definition 1 (ℓ_p Sampling)**.** *An (ϵ, δ) ℓ_p-sampler for $\mathbf{x} \neq 0$ returns $\perp$ with probability at most δ and otherwise returns some $i \in [n]$ with probability in the range:*

$$\left[\frac{(1-\epsilon)|x_i|^p}{\ell_p^p(\mathbf{x})}, \frac{(1+\epsilon)|x_i|^p}{\ell_p^p(\mathbf{x})}\right] ,$$

where $\ell_p(\mathbf{x}) = \left(\sum_{i\in[n]} |x_i|^p\right)^{1/p}$ is the p-norm of $\mathbf{x}$.

The next lemma, due to Jowhari et al. [28], summarizes the state-of-the-art result for $p \in \{0, 1\}$.

Lemma 2.1. *There exists linear sketch-based algorithms that perform ℓ_p sampling using:*

1. *$O(\log^2 n \log \delta^{-1})$ space for $p = 0$. Note we may set $\epsilon = 0$ in this case.*
2. *$O(\epsilon^{-1} \log \epsilon^{-1} \log^2 n \log \delta^{-1})$ space for $p = 1$.*

A simple application to graph sketching is to apply sketches for ℓ_0-sampling to the characteristic vector $\mathbf{a}^v \in \{0, 1\}^n$ of the neighborhood of a node v, i.e., $\mathbf{a}^v[u] = 1$ iff u is a neighbor of v. Call this sketch $\mathcal{S}(\mathbf{a}^v)$. By querying $\mathcal{S}(\mathbf{a}^v)$, we can identify a random neighbor of v. Since $\mathcal{S}$ is linear, if any elements of $\mathbf{a}^v$ change, we can easily update $\mathcal{S}(\mathbf{a}^v)$. It might seem that we can then query the sketch again to return a new random neighbor of v. This is true in a sense but there are the following important caveats:

1. *Loss of Independence:* The first issue is the simple observation that if we repeatedly update and query $\mathcal{S}(\mathbf{a}^v)$, the random neighbors returned will not be independent.

2. *Sketch Updates must be Non-Adaptive:* The second issue is more insidious. Suppose we query $\mathcal{S}(\mathbf{a}^v)$ and are returned node u. Can we somehow get the sketch $\mathcal{S}(\mathbf{a}^v)$ to yield an additional neighbor of v? One idea would be to update $\mathbf{a}^v$ by removing u from the neighborhood of v and then query the updated sketch of $\mathbf{a}^v$. However, this clearly does not work because otherwise we could repeat the process multiple times and return all neighbors of v from the $O(\log^2 n \log \delta^{-1})$ size sketch! The issue is that we may not adaptively update the data being sketched based on the sketch itself. This caveat is especially important when processing graphs since many graph algorithms are adaptive in non-trivial ways.

Algorithm SPANNING-FOREST

1. Sketch $\mathbf{a}_1, \ldots, \mathbf{a}_n$ using the sketch matrices $\mathcal{S}_1, \ldots, \mathcal{S}_t$ where $t = O(\log n)$.
2. Initialize the set of supernodes as $\hat{V} = V$.
3. For $r \in [t]$,
   (a) For each $s \in \hat{V}$, try to sample an inter-supernode edge using the sketch $\sum_{v_i \in s} \mathcal{S}_r(a_i)$
   (b) Update $\hat{V}$ by collapsing connected supernodes.
4. Assert that G has $|\hat{V}|$ connected components and that any maximal acyclic sub-graph of the set of sampled edges is a spanning forest.

Figure 1: The SPANNING-FOREST Algorithm

However, there are various useful things that you *can* do in sketch space. For example, suppose we have used the same random bits to construct sketches $\mathcal{S}(\mathbf{a}^u)$, $\mathcal{S}(\mathbf{a}^v)$ of the neighborhoods of nodes u and v. If our graph algorithm calls for merging u and v to create a new node w, we automatically have the ability to sample a random neighbor of w in the resulting multigraph. This follows since $\mathcal{S}(\mathbf{a}^w) = \mathcal{S}(\mathbf{a}^u) + \mathcal{S}(\mathbf{a}^v)$ by linearity. We will see how such as idea can be very useful in the next couple of sections.

3 Connectivity and Applications

3.1 Connectivity

We present a single-pass, semi-streaming algorithm that computes the number of connected components of a dynamic graph. Our algorithm is based on constructing sketches for ℓ_0-sampling the rows of a matrix that we now define.

Definition 2. *Given an unweighted graph $G = (V, E)$, define the $n \times \binom{n}{2}$ matrix A_G with entry $(i, (j, k)) \in [n] \times \binom{[n]}{2}$ defined by*

$$a_{i,(j,k)} = \begin{cases} 1 & \text{if } i = j \text{ and } (v_j, v_k) \in E \\ -1 & \text{if } i = k \text{ and } (v_j, v_k) \in E \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{a}_1, \ldots, \mathbf{a}_n$ be the rows of A_G where $\mathbf{a}_i$ corresponds to node v_i. The following preliminary lemma follows immediately from the above definition.

Lemma 3.1. *Let $E_S = E(S, V \setminus S)$ be the set of edges across the cut $(S, V \setminus S)$. Then, $|E_S| = \ell_0(\mathbf{x})$ where $\mathbf{x} = \sum_{v_i \in S} \mathbf{a}_i$. Furthermore, $\mathbf{x} \in \{-1, 0, 1\}^{\binom{n}{2}}$ with $|\mathbf{x}_{(j,k)}| = 1$ iff $(v_j, v_k) \in E_S$.*

Our algorithm is based on the following simple $O(\log n)$ stage process. In the first stage, we find any incident edge on each node. We then collapse each of the resulting connected components into a "supernode". In each subsequent stage we find an edge from each supernode to another supernode if one exists. If the graph has $cc(G)$ connected components, the difference between the number of supernodes and $cc(G)$ halves with each stage and therefore after $O(\log n)$ stages the graph has

collapsed into $cc(G)$ supernodes and will not collapse further. This is a trivial algorithm! The challenge is to emulate the algorithm space-efficiently in a single pass over a dynamic graph stream.

Sketch-Based Algorithm. To emulate the basic algorithm efficiently, we will construct a number of sketches of each $\mathbf{a}_i$ that will facilitate the ℓ_0-sampling. Let the sketch matrices be $\mathcal{S}_1, \ldots, \mathcal{S}_t$ where $t = O(\log n)$. Constructing a sketch of each node, using each sketch matrix requires $O(nt \log^2 n)$ for constant δ. For our algorithm it will suffice to set the ℓ_0-sampling parameters as $\epsilon = 0$ and $\delta = 1/100$.

To sample an edge incident on v_i, we can use the sketch $\mathcal{S}_1(\mathbf{a}_i)$. Appealing to Lemma 3.1, this succeeds with probability $1-\delta$. At the next step, to sample an edge from the super-node $s = \{v_i, v_j\}$ we can use the sketch $\mathcal{S}_2(\mathbf{a}_i) + \mathcal{S}_2(\mathbf{a}_j)$ to find such an edge with probability at least $1 - \delta$. This follows by the linearity of the sketches and by appealing again to Lemma 3.1. We emphasize that using $\mathcal{S}_1(\mathbf{a}_i) + \mathcal{S}_1(\mathbf{a}_j)$ rather than $\mathcal{S}_2(\mathbf{a}_i) + \mathcal{S}_2(\mathbf{a}_j)$ will *not* work for two reasons: a) it would mean that we has used $\mathcal{S}_1$ in the first stage to determine our use of $\mathcal{S}_1$ in the second stage and such adaptivity is not permissible in general and b) the probabilities from Lemma 3.1 would not be independent. In short, we need to use a new sketching matrix $\mathcal{S}_i$ in each round. See Fig. 1 for the full algorithm.

To correctness of the algorithm follows because in each stage we expect to decrease $|\hat{V}| - cc(G)$ by at least a third by appealing to the linearity of expectation. Hence, it suffices to set $t = O(\log n)$.

Theorem 3.1. *There exists a single-pass, $O(n \cdot \log^3 n)$-space algorithm for dynamic connectivity. The algorithm returns a spanning forest of the graph.*

3.2 k-Edge-Connectivity

We next present a single-pass algorithm for k-edge-connectivity. This algorithm builds upon ideas in the previous section and exploits the "updatability" of the sketches used in ℓ_0-sampling. The starting point for the algorithm is the following simple k phase algorithm:

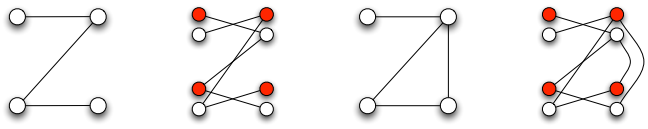
1. For $i = 1$ to k: Let F_i be a spanning forest of $(V, E \setminus \bigcup_{j=1}^{i-1} F_j)$
2. Then $(V, F_1 \cup F_2 \cup \ldots \cup F_k)$ is k-edge-connected iff $G = (V, E)$ is at least k-edge-connected.

The correctness of this algorithm is simple to show.

Lemma 3.2. *Given a graph $G = (V, E)$, for $i \in [k]$, let F_i to be a spanning forest of $(V, E \setminus \bigcup_{j=1}^{i-1} F_j)$. $(V, F_1 \cup F_2 \cup \ldots \cup F_k)$ is k-edge-connected iff $G = (V, E)$ is at least k-edge-connected.*

Proof. Let $E' = F_1 \cup \ldots \cup F_k$. Consider a cut $(S, V \setminus S)$ and let $E_S \subset E$ be the set of edges that cross this cut. Similarly, let E'_S be the set of edges among E' that cross this cut. Clearly $|E_S| \geq |E'_S|$ since $E' \subset E$. Hence, it suffices to prove that $|E'_S| \geq k$ if G is k-edge-connected. Suppose there exists i such that $F_i \cap E_S = \emptyset$ (otherwise we are done by the edge-disjointness of the k spanning forests.) Then $E_S \cap (F_1 \cup F_2 \cup \ldots \cup F_{i-1}) = E_S$ and hence $|E'_S| = |E_S| \geq k$. □

It is relatively straight-forward to design a $O(k)$-pass algorithm using the spanning forest algorithm from the previous section. However, by exploiting the linearity of sketches, we show that it is possible to test k-edge connectivity in only one pass. The algorithm is as follows:



(a) G (b) $D(G)$ (c) G' (d) $D(G')$

Figure 2: The map D doubles the number of connected components iff the graph is bipartite.

1. Construct the sketches for k independent instantiations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ of the spanning forest algorithm.
2. For $i \in [k]$:
   (a) Use the sketches for $\mathcal{I}_i$ to find a spanning forest F_i of $(V, E \setminus F_1 \cup \ldots \cup F_{i-1})$
   (b) Update the sketches for $\mathcal{I}_{i+1}, \ldots, \mathcal{I}_k$ by deleting all edges in F_i

Because of instantiations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ are independent, we may update then as claimed.

Theorem 3.2. *There exists a one-pass, $O(k \cdot n \cdot \log^3 n)$-space algorithm that tests k-edge connectivity where the space depends linearly on k.*

3.3 Bipartiteness

Next, we reduce the bipartiteness problem to the problem of counting the number of connected components. The reduction is based on the following local mapping.

Definition 3. *For a graph $G = (V, E)$, let $D(G) = (V', E')$ be the graph constructed as follows. For each $v \in V$ construct $v_1, v_2 \in V'$ and for each edge $(u, v) \in E$, create two edges (u_1, v_2) and (u_2, v_1).*

See Figure 2 for an illustration of the action of D.

Lemma 3.3. *Let K be the number of connected components in G. Then, $D(G)$ has $2K$ connected components if and only if G is bipartite.*

Proof. Let $G_1, G_2, \cdots, G_K$ be the connected components in G. By the construction, $D(G)$ consists of $D(G_1)$, $D(G_2)$, $\cdots$, $D(G_K)$. If we sum the number of connected components in each $D(G_i)$, we have the total number of connected components in $D(G)$.

Suppose that there exists an odd cycle in G_i which contains u. The odd cycle corresponds to a path from u_1 to u_2 or vice versa in $D(G_i)$. And each path from u to v in G_i corresponds to a path from u_1 to v_1 or v_2 and a path from u_2 to the other copy of v. Since G_i is connected, there exists a path from u to any $v \in V$. So there is a path from u_1 to any vertex in $v \in V'$ and $D(G_i)$ is connected. On the other hand, if G_i is bipartite, $D(G_i)$ is not connected because any path from u_1 to u_2 in G' corresponds to an odd cycle in G.

If G is bipartite, every G_i is bipartite. Therefore, there are $2K$ connected components in $D(G)$. On the other hand, if G is not bipartite, there must be G_i that is not bipartite. So there is less than $2K$ connected components. □

Hence, by applying the SPANNING-FOREST algorithm on G and $D(G)$ we prove the following.

Theorem 3.3. *There exists a single-pass semi-streaming algorithm for dynamic bipartiteness.*

3.4 Approximate Minimum Spanning Trees

Next, we reduce the problem of estimating the weight of the minimum spanning tree to the problem of counting the number of connected components in graphs. The reduction uses an idea due to Chazelle et al. [8]. Consider a graph G with edge weights are in the range $[1, W]$ where $W = \text{poly}(n)$. We will assume that G is connected but our algorithm can be used to estimate the weight of the minimum weight spanning forest if G is unconnected. Let G_i be the subgraph of G consisting of all edges whose weight is at most $w_i = (1+\epsilon)^i$ and let $cc(H)$ denote the number of connected components of a graph H.

Lemma 3.4. *Let T be a minimum spanning tree of G and set $r = \lceil \log_{1+\epsilon} W \rceil$. Then*

$$w(T) \leq n - (1+\epsilon)^r + \sum_{i=0}^{r} \lambda_i cc(G_i) \leq (1+\epsilon) w(T)$$

where $\lambda_i = (1+\epsilon)^{i+1} - (1+\epsilon)^i$.

Proof. Consider the G' formed by rounding each edge weight up to the nearest power of $(1+\epsilon)$. Then it is clear that $w(T) \leq w(T') \leq (1+\epsilon)w(T)$ where T' is a minimum spanning tree of G'. It remains to compute $w(T')$ and we do this by considering the operation of Kruskal's algorithm on G'. Kruskal's algorithm will first add $n - cc(G_1)$ edges of weight 1, then $cc(G_1) - cc(G_2)$ edges of weight $(1+\epsilon)$ etc. The total weight of edges added will be

$$w(T') = (n - cc(G_1)) + \sum_{i=1}^{r-1} (1+\epsilon)^i (cc(G_i) - cc(G_{i+1}))$$

which simplifies to give the claimed quantity. □

Hence, we can estimate the weight of the minimum spanning tree using the connectivity algorithm.

Theorem 3.4. *There exists a single-pass, $O(\epsilon^{-1} \cdot n \cdot \log^3 n)$ space algorithm that $(1+\epsilon)$ approximates the weight of the minimum spanning tree of a dynamic graph.*

4 Exact Minimum Spanning Trees

Basic Algorithm. Our starting point is Boruvka's algorithm for finding the MST. This algorithm proceeds in $O(\log n)$ phases. In each phase, the minimum weight edge on each node is added and the resulting connected components are collapsed to form new nodes. This algorithm can be implemented in the dynamic stream setting in $O(\log^2 n)$ passes by emulating each phase in $O(\log n)$ passes of the dynamic graph stream. We emulate a phase as follows: In the first pass, we ℓ_0-sample an incident edge on each node without considering the weights. Suppose we sample an edge with weight w_v on node v. In the next pass, we again ℓ_0 sample incident edges but this time we ignore all edges of weight at least w_v on node v when we construct the sketch. Repeating this process $O(\log n)$ ensures that we succeed in finding the minimum weight edge incident on each node. Thus the algorithm takes $O(\log^2 n)$ passes as claimed. In the rest of this section, we transform the algorithm into a new algorithm that uses only $O(\log n)$ passes.

Reducing the Number of Passes. Our first step is to show that $O(\log n \log\log n)$ passes suffice. The algorithm is based on the observation that the number of nodes under consideration in the ith phase is at most $n/2^i$. Hence, during the ith phase we can afford to sample $t_i = 2^i$ incident edges without violating the semi-streaming space restriction. Therefore, the ith phase can be emulated in $O(\log_{t_i} n)$ passes which implies that the total number of passes is $\sum_{i=1}^{\log n} \log_{2^i} n = O(\log n \log\log n)$.

The next step is to reduce the number of phases. The basic idea is to not just find the lightest incident edge for each node, but to find the k lightest edges. It follows from the next lemma that this allows us to reduce the number of nodes by a factor k.

Lemma 4.1. *In a simple weighted graph $G = (V, E)$, if $E' \subset E$ contains the k lightest incident edges on each node, then we can identify the lightest edge in the cut $(S, V \setminus S)$ for any subset $S \subset V$ of size at most k.*

Proof. If $|S| \leq k$ then we know the lightest edge incident on each $v \in S$ in $E(S, V \setminus S)$ because v has at most $k - 1$ neighbors in S. □

Unfortunately, after the first phase the graph under consideration is a multi-graph and the above lemma does not apply directly. For example, the lightest k incident edges on v may all connect v to the same neighbor. However, we can rectify this situation by constructing $O(\log n)$ random partitions $P_1, \ldots, P_{O(\log n)}$ of the nodes where each partition is of size $2k$. For each node v and each partition $P = \{V_1, \ldots, V_{2k}\}$ we find the lightest edge from v to each V_i. Let E' be the set of edges collected.

Lemma 4.2. *With high probability, E' contains the k lightest edges to distinct neighbors.*

Proof. Let N_v be the k closest neighbors of v and consider $u \in N_v$. With high probability there exists a partition $P = \{V_1, \ldots, V_{2k}\}$ where u is the only element in $V_i \cap N_v$ for some i. Hence, we identify the lightest edge between u and v. □

The next theorem is proved by carefully combining the above idea with the $O(\log n \log\log n)$ pass algorithm.

Theorem 4.1. *There exists a $O(p)$-pass, $\tilde{O}(n^{1+1/p})$-space algorithm that finds the MST of a dynamic graph stream. In particular there is a semi-streaming algorithm that uses $O(\log n / \log\log n)$ passes.*

Proof. Suppose at some point we have $n^{1+1/p}/t$ remaining nodes. Then we can find the $\sqrt{t}$ closest neighbors of a node in $O(\log_{\sqrt{t}} n)$ passes as follows: construct $O(\log n)$ random partitions each of size $2\sqrt{t}$ and then use $O(\log_{\sqrt{t}} n)$ successive batches of $\sqrt{t}$ sketches for ℓ_0-sampling to find the lightest edges between each node and a node in each set of each partition. Let n_i be the number of nodes at the start of the ith phase and define $t_i = n^{1+1/p}/n_i$. Then $t_1 = n^{1/p}, t_2 = n^{3/(2p)}, t_3 = n^{9/(4p)} \ldots$ and hence $\sum_i \log_{t_i}(n) = O(p)$. □

5 Sparsification

In this section, we present a multi-pass, semi-streaming algorithm for graph sparsification in the dynamic setting. The concept of (cut) sparsification was introduced by Benczúr and Karger [5] and is defined as follows:

Algorithm SPARSIFIER

1. Let $G' = G$, $H = (V, \emptyset)$.
2. While $(E(G') \neq \emptyset)$ do
   (a) Let $|E(G')| = kn$. Sample $6\epsilon^{-2}tn \ln n$ edges from $E(G')$ using ℓ_1 sampling.
   (b) Assign each sampled edge e weight $\frac{\epsilon^2 k}{6t \ln n} f_e$ where f_e is the number of times it is sampled.
   (c) Let H' be the resulting graph and let $\{V'\}$ be its set of $(1+\epsilon)\frac{t}{k}$-strong connected components.
   (d) Set $E(G') \leftarrow E(G') \backslash \{(u,v) \in E(G') : u, v \in V'\}$ and $H \leftarrow H \cup \{(u,v) \in H' : u, v \in V'\}$.

Figure 3: The SPARSIFIER Algorithm

Definition 4. *A weighted graph $H = (V, E', w')$ is a* sparsifier *for a graph $G = (V, E, w)$ if for every cut, the cut value in H is within a $(1 \pm \epsilon)$ factor of the cut value in G.*

We will use the results of [5, 29] and start by recapping a key definition and two important lemmas.

Definition 5. *A node induced subgraph of G is a k-*strong connected component *if its minimum cut value is k. An edge e is k-*strong connected *if k is the maximum value such that there exists a k-strong connected component that contains e.*

Lemma 5.1 (Benczúr, Karger [5])**.** *If G has kn edges, there exists k-strong connected component.*

The next property is a restatement of Theorem 2.1 in [29]; the proof in [29] used uniform random sampling. In our setting, we use ℓ_1 sampling to allow deletions.

Lemma 5.2 (Karger [29])**.** *Let C be the minimum cut value of a graph G. If we sample edges from G using ℓ_1-sampling and give weights $\frac{6}{\epsilon^2 C} \ln n$ for sampled edges, we preserve every cut value up to $1 \pm \epsilon$ factor with high probability.*

Consider the algorithm SPARSIFIER in Figure 3. Using the above lemmas we can show that:

Lemma 5.3. *If the strong connectivity of an edge e is at least $(1 + 3\epsilon)k/t$, it is eliminated with high probability.*

Proof. Let e be an edge with strong connectivity $c_e \geq (1+2\epsilon)k/t$ and H_e be the c_e-strong connected component that contains e. By Lemma 5.2, every cut in H_e is preserved with high probability. So H_e is $(1-\epsilon)c_e$-connected in H' and e is eliminated with high probability. □

Lemma 5.4. *Let c_e be the strong connectivity of an edge e. If we sample each edge with probability $\epsilon^{-2}6t \ln n/k$, the strong connectivity of an edge e is estimated as $c_e + \epsilon k/t$ or less with high probability.*

Proof. Given a graph $G = (V, E)$, we can construct a cut $C_f(e)$ for every edge e such that if we approximate the cut $C_f(e)$ then we cannot overestimate c_e by more than an additive $\epsilon k/t$ term.

Consider G and its min-cut. If e crosses the mincut, $C_f(e)$ is the mincut. Otherwise consider the part to which e belongs and proceed recursively. Note that we are constructing a tree of cuts with a polynomial number of cuts.

The values of the cuts we find till we find $C_f(e)$ (including $C_f(e)$) are at most c_e since otherwise we find a node-induced subgraph whose minimum cut is larger than c_e and contains e. By definition of the strong connectivity, there is no such subgraph. From Chernoff bound [9], every cut value is preserved up to $\epsilon k/t$ additive error with high probability. Therefore, we estimate the strong connectivity of e by at most $c_e + \epsilon k/t$ with high probability. □

From Lemma 5.4, we only remove components with strong connectivity at least k/t. From Lemma 5.2, we preserve every cut in the removed components. When the algorithm finishes, each cut in the graph can be represented as a sum of cuts in these removed components. Therefore, we preserve the value of every cut within $1 \pm \epsilon$ factor with high probability. By Lemma 5.3, every $(1+\epsilon)k/t$-connected component is removed from G'. By Lemma 5.1, we have at most $(1+\epsilon)kn/t$ remaining edges. Thus, the algorithm terminates in $O(\log_t n)$ passes. Since we sample $O(\epsilon^{-2}tn \log n)$ edges for each pass, we obtain a sparsification of size $O(\epsilon^{-2}tn \log^2 n/\log t)$. Setting t appropriately, we obtain the following theorem:

Theorem 5.1. *There exists an algorithm using $\tilde{O}(\epsilon^{-2}n^{1+1/p})$ space that returns a graph sparsification of size $O(\epsilon^{-2}n\log^3 n/\log\log n)$ in $O(p)$ passes. This algorithm also also yields a semi-streaming algorithm that returns graph sparsification in $O(\log n/\log\log n)$ passes.*

6 A Discussion about Matchings

In this short section we discuss algorithms for finding maximal and approximately-maximum matchings in dynamic graph streams. Since both results rely heavier on techniques developed in other models and in other papers [2, 3, 31], we limit ourselves to summarizing the results and the main issues involved.

Maximal Matchings. To find a maximal matching we appeal to a result by Lattanzi et al. [31] for approximating matchings (with no deletions) in the MapReduce model. Their algorithm repeatedly samples a subset of $\eta = O(n^{1+1/p})$ edges in each round and finds a maximal matching among the sampled edges, for p rounds; the vertices in the matching are excluded in the subsequent rounds. The same algorithm can be implemented in the dynamic semi-streaming model using sketches for ℓ_0-sampling. In each pass, we can sample η edges uniformly at random and hence, each round of the algorithm corresponds to a single pass.

Maximum Matchings. It is shown in [3], that it is possible to $(1-\epsilon)$ approximate the *maximum* bipartite matching (in multiple passes) given the ability to find *maximal* matchings. To do this we can use the maximal matching algorithm described above. This leads to the following result.

Theorem 6.1. *There exists a $O(n^{1+1/p} \cdot \operatorname{poly}\epsilon^{-1})$ space, $O(p \cdot \epsilon^{-2} \cdot \log\epsilon^{-1})$-pass algorithm that returns a matching that is $(1-\epsilon)$-approximation for the bipartite maximum weighted matching in the dynamic graph stream model. The number of passes can be improved to $O(p \cdot \epsilon^{-2} \cdot \log\log\epsilon^{-1})$ in the unweighted case.*

The situation is more complicated for non-bipartite graphs. It is possible to $(1-\epsilon)$-approximatate the size of the maximum matching and construct such a matching in non-bipartite graphs in the insertion- only model [2]. However, those techniques used rely on a specific approach to constructing a maximal matching and do not apply immediately in the dynamic setting. However, one can instead use the sparsifier algorithm presented in Section 5 to get the following:

Theorem 6.2. *There exists a $O(n^{1+1/p} \cdot \operatorname{poly} \epsilon^{-1})$ space, $O(p^2 \cdot \operatorname{poly} \epsilon^{-1})$-pass algorithm that returns a $(1-\epsilon)$-approximation to the weight of the (non-bipartite) maximum weighted matching in the dynamic graph stream model. The number of passes can be improved to $O(p^2 \cdot \epsilon^{-1})$ in the unweighted case. To construct such a matching, there exists a $O(n^{1+1/p} \cdot \operatorname{poly} \epsilon^{-1})$ space, $O(p \cdot \log n \cdot \operatorname{poly} \epsilon^{-1})$-pass algorithm.*

We also note that our algorithm can be implemented in the MapReduce model as well. It would improve upon the 8-approximation algorithm (for the weighted case and the obvious factor 2 in the unweighted case) in [31] for the MapReduce model.

Acknowledgments. The authors would like to thank Piotr Indyk for some useful conversations conducted in the back of a taxi somewhere between Orchha and Lucknow. The third author would also like to thank any other passengers for their patience and understanding.

References

[1] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *ICALP (2)*, pages 328–338, 2009.

[2] K. J. Ahn and S. Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *Manuscript, available at http://arxiv.org/abs/1104.4058*, 2011.

[3] K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *ICALP (2)*, pages 526–538, 2011.

[4] K.-J. Ahn, S. Guha, and A. McGregor. Graph sketches: Sparsification, spanners, and subgraphs. In *Manuscript*, 2011.

[5] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[6] A. D. Bonis, L. Gasieniec, and U. Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.*, 34(5):1253–1270, 2005.

[7] T. M. Chan, M. Patrascu, and L. Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011.

[8] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[9] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[10] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.

[11] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

[12] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *PODS*, pages 77–86, 2010.

[13] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[14] M. Elkin. A near-optimal fully dynamic distributed algorithm for maintaining sparse spanners, 2006.

[15] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[16] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *CoRR*, abs/00907.0305, 2000.

[17] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005.

[18] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.

[19] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. In *Symposium on Computational Geometry*, pages 142–149, 2005.

[20] S. Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.

[21] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, pages 389–398, 2002.

[22] M. T. Goodrich and D. S. Hirschberg. Improved adaptive group testing algorithms with applications to multiple access channels and dead sensor diagnosis. *J. Comb. Optim.*, 15(1):95–121, 2008.

[23] J. Haupt, R. Castro, and R. Nowak. Distilled sensing: selective sampling for sparse signal recovery. *Journal of Machine Learning Research - Proceedings Track*, 5:216–223, 2009.

[24] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[25] P. Indyk. Algorithms for dynamic geometric problems over data streams. *Proc. STOC*, pages 373–380, 2004.

[26] P. Indyk, E. Price, and D. Woodruff. On the power of adaptivity in sparse recovery. In *FOCS*, 2011.

[27] W. B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert Space. *Contemporary Mathematics, Vol 26*, pages 189–206, May 1984.

[28] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58, 2011.

[29] D. R. Karger. Random sampling in cut, flow, and network design problems. In *STOC*, pages 648–657, 1994.

[30] J. A. Kelner and A. Levin. Spectral sparsification in the semi-streaming setting. In *STACS*, pages 440–451, 2011.

[31] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, pages 85–94, 2011.

[32] A. McGregor. Finding graph matchings in data streams. *APPROX-RANDOM*, pages 170–181, 2005.

[33] A. McGregor. Graph mining on streams. In *Encyclopedia of Database Systems*, pages 1271–1275, 2009.

[34] M. Monemizadeh and D. P. Woodruff. 1-pass relative-error ℓ_p-sampling with applications. In *SODA*, pages 1143–1160, 2010.

[35] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, 2006.

[36] S. Muthukrishnan. Some algorithmic problems and results in compressed sensing. In *Allerton Conference*, 2006.

[37] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350, 2000.

[38] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *STOC*, pages 112–119, 2005.

[39] M. Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, pages 1–20, 2010. 10.1007/s00453-010-9438-5.