# Vertex and Hyperedge Connectivity in Dynamic Graph Streams

Sudipto Guha* Andrew McGregor† David Tench‡

**Abstract**

A growing body of work addresses the challenge of processing dynamic graph streams: a graph is defined by a sequence of edge insertions and deletions and the goal is to construct synopses and compute properties of the graph while using only limited memory. Linear sketches have proved to be a powerful technique in this model and can also be used to minimize communication in distributed graph processing.

We present the first linear sketches for estimating vertex connectivity and constructing hypergraph sparsifiers. Vertex connectivity exhibits markedly different combinatorial structure than edge connectivity and appears to be harder to estimate in the dynamic graph stream model. Our hypergraph result generalizes the work of Ahn et al. (PODS 2012) on graph sparsification and has the added benefit of significantly simplifying the previous results. One of the main ideas is related to the problem of reconstructing subgraphs that satisfy a specific sparsity property. We introduce a more general notion of graph degeneracy and extend the graph reconstruction result of Becker et al. (IPDPS 2011).

## 1 Introduction

Massive graphs arise in many applications. Popular examples include the web-graph, social networks, and biological networks but, more generally, graphs are a natural abstraction whenever we have information about both a set of basic entities and relationships between these entities. Unfortunately, it is not possible to use existing algorithms to process many of these graphs; many of these graphs are too large to be stored in main memory and are constantly changing. Rather, there is a growing need to design new algorithms for even basic graph problems in the relevant computational models.

In this paper, we consider algorithms in the dynamic data stream and linear sketching models. In the dynamic data stream model, a sequence of edge insertions and deletions defines an input graph and the goal is to solve a specific problem on this graph given only one-way access to the input sequence and limited working memory. While insert-only graph streaming has been an active area of research for over a decade, it is only relatively recently algorithms have been found that handle insertions and deletions [2–4, 16, 19, 20, 24]. The main technique used in these algorithms is *linear sketching* where a random linear projection of the input graph is maintained as the graph is updated. To be useful, we need to be able to a) store the projection of the graph in small space and b) solve the problem of interest given only the projection of the graph. While linear sketching is a classic technique for solving statistical problems in the data stream model, it was long

*University of Pennsylvania. Supported by NSF Awards CCF-1117216. sudipto@seas.upenn.edu

†University of Massachusetts Amherst. Supported by NSF Awards CCF-0953754, IIS-1251110, CCF-1320719, and a Google Research Award. mcgregor@cs.umass.edu.

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

thought unlikely to be useful in the context of combinatorial problems on graphs. Not only do linear sketches allow us to process edge deletions (a deletion can just be viewed as a "negative" insertion) but the linearity of the resulting data structures enables a rich set of algorithmic operations to be performed after the sketch has been generated. Linear sketches are also a useful technique for reducing communication when processing distributed graphs. For a recent survey of graph streaming and sketching see [25].

## 1.1 Our Contributions and Related Work

We present sketch-based dynamic graph algorithms for three basic graph problems: computing vertex connectivity, graph reconstruction, and hypergraph sparsification. All our algorithms run in (low) polynomial time, typically linear in the number of edges. However, our primary focus is on space complexity, as is the convention in much of the data streams literature. In what follows, let $n$ denote the number of vertices in the graph.

**Vertex Connectivity.** To date, the main success story for graph sketching has been about edge connectivity, i.e., estimating how many edges need to be removed to disconnect the graph, and estimating the size of cuts. In this paper we present the first dynamic graph stream algorithms for vertex connectivity, i.e., estimating how many *vertices* need to be removed to disconnect the graph. While it can be shown that edge connectivity is an upper bound for vertex connectivity, the vertex connectivity of a graph can be much smaller. Furthermore, the combinatorial structure relevant to both quantities is very different. For example, edge-connectivity is transitive[1] whereas vertex-connectivity is not. A celebrated result by Karger [21] bounds the number of near minimum cuts whereas no analogous bound is known for vertex removal. Feige et al. [14] discuss issues that arise specific to vertex connectivity in the context of approximation algorithms and embeddings.

In Section 3, we present two sketch-based algorithms for vertex connectivity. The first algorithm uses $O(kn \operatorname{polylog} n)$ space and constructs a data structure such that, at the end of the stream, it is possible to test whether the removal of a queried set of at most $k$ vertices would disconnect the graph. We prove that this algorithm is optimal in terms of its space use. The second algorithm estimates the vertex connectivity up to a $(1+\epsilon)$ factor using $O(\epsilon^{-1}kn \operatorname{polylog} n)$ space where $k$ is an upper bound on the vertex connectivity.

No stream algorithms were previously known that supported both edge insertions and deletions. Existing approaches either use $\Omega(n^2)$ space [28] or only handle insertions [13]. With only insertions, Eppstein et al. [13] proved that $O(kn \operatorname{polylog} n)$ space was sufficient. Their algorithm drops an inserted edge $\{u, v\}$ iff there already exists $k$ vertex-disjoint paths between $u$ and $v$ amongst the edges stored thus far. Such an algorithm fails in the presence of edge deletions since some of the vertex disjoint paths that existed when an edge was ignored need not exist if edges are subsequently deleted.

**Graph Reconstruction.** Our next result relates to reconstructing graphs rather than estimating properties of the graph. Becker et al. [5] show that is possible to reconstruct a $d$-degenerate graph given an $O(d \operatorname{polylog} n)$ size sketch of each row of the adjacency matrix of the graph. In Section 4, we define the $d$-cut-degeneracy and show that the strictly larger class of graphs that satisfy this property can also be reconstructed given an $O(d \operatorname{polylog} n)$-size sketch of each row. Moreover, even if the graph is not $d$-cut-degenerate we show that we can find all edges with a certain connectivity property. This will be an integral part of our algorithm for hypergraph sparsification. For this purpose, we also prove the first dynamic graph stream algorithms for hypergraph connectivity in this section. We also extend the vertex connectivity results to hypergraphs.

[1] If it takes at least $k$ edge deletions to disconnect $u$ and $v$ and it takes at least $k$ edge deletions to disconnect $v$ and $w$, then it takes at least $k$ edge deletions to disconnect $u$ and $w$.

**Hypergraph Sparsification.** Hypergraph sparsification is a natural extension of graph sparsification. Given a hypergraph, the goal is to find a sparse weighted subgraph such that the weight of every cut in the subgraph is within a $(1+\epsilon)$ factor of the weight of the corresponding cut in the original hypergraph. Estimating hypergraph cuts has applications in video object segmentation [17], network security analysis [30], load balancing in parallel computing [8], and modelling communication in parallel sparse-martix vector multiplication [7].

Kogan and Krauthgamer [23] recently presented the first stream algorithm for hypergraph sparsification in the insert-only model. In Section 5, we present the first algorithm that supports both edge insertions and deletions. The algorithm uses $O(n \operatorname{polylog} n)$ space assuming that size of the hyperedges is bounded by a constant. This result is part of a growing body of work on processing hypergraphs in the data stream model [12, 23, 26, 27, 29]. There are numerous challenges in extending previous work on graph sparsification [3, 4, 16, 19, 20] to hypergraph sparsification and we discuss these in Section 5. In the process of overcoming these challenges, we also identify a simpler approach for graph sparsification in the data stream model.

## 2 Models and Preliminaries

**Graphs Preliminaries.** A hypergraph is specified by a set of vertices $V = \{v_1, \ldots, v_n\}$ and a set of subsets of $V$ called hyperedges. In this paper we assume all hyperedges have cardinality at most $r$ for some constant $r$. The special case when all hyperedges have cardinality exactly two corresponds to the standard definition of a graph. All graphs and hypergraphs discussed in this paper will be undirected. It will be convenient to define the following notation: Let $\delta_G(S)$ be the set of hyperedges that cross the cut $(S, V \setminus S)$ in the hypergraph $G$ where we say a hyperedge $e$ crosses $(S, V \setminus S)$ if $e \cap S \neq \emptyset$ and $e \cap (V \setminus S) \neq \emptyset$. For any hyperedge $e$, define $\lambda_e(G)$ to be the minimum cardinality of a cut that includes $e$. A *spanning graph* $H = (V, E)$ of a hypergraph $G = (V, E)$ is a subgraph such that $|\delta_H(S)| \geq \min(1, |\delta_G(S)|)$ for every $S \subset V$.

**Linear Sketches and Applications.** All the algorithms presented in this paper use linear sketches.

**Definition 1** (Linear Sketches)**.** *A* linear measurement *of a hypergraph on $n$ vertices is defined by a set of coefficients $\{c_e : e \in \mathcal{P}_r(V)\}$ where $\mathcal{P}_r(V)$ is the set of all subsets of $V$ of size at most $r$. Given a hypergraph $G = (V, E)$, the evaluation of this measurement is defined as $\sum_{e \in E} c_e$. A* sketch *is a collection of (non-adaptive) linear measurements. The cardinality of this collection is referred to as the* size *of the sketch. We will assume that the magnitude of the coefficients $c_e$ is $\operatorname{poly}(n)$. We say a linear measurement is* local *for node $v$ if the measurement only depends on hyper-edges incident to $v$, i.e., $c_e = 0$ for all hyper-edges that do not include $v$. We say a sketch is* vertex-based *if every linear measurement is local to some node.*

Linear sketches have long been used in the context of data stream models because it is possible to maintain a sketch of the stream incrementally. Specifically, if the next stream update is an insertion or deletion of an edge, we can update the sketch by simply adding or subtracting the appropriate set of coefficients. Sketches are also useful in distributed settings. In particular, the model considered by Becker et al. [5] was as follows: suppose there are $n+1$ players $P_1, \ldots, P_n$ and $Q$. The input for player $P_i$ is the set of (hyper-)edges that include the $i$th vertex of a graph $G$. Player $Q$ wants to compute something about this graph such as determining whether $G$ connected. To enable this, each of the players $P_1, \ldots, P_n$ simultaneously sends a message about their input to $Q$ such that the set of these $n$ messages contains sufficient information to complete $Q$'s computation. In the case of randomized protocols, we assume that all players have access to public random bits. The goal is to minimize the maximum length of the $n$ messages that are sent to $Q$. If a vertex-based sketch exists for the problem under consideration, then for each linear measurement, there is a single player that can evaluate this message and send it to $Q$.

# 3 Vertex Connectivity

A natural approach to determining vertex connectivity could be to try to mimic the algorithm of Cheriyan et al. [11]. They showed that the union of $k$ disjoint "scan first search trees" (a generalization of breadth-first search trees) can be used to determine if a graph is $k$ vertex connected. A similar approach worked in data stream model for the case of edge-connectivity (which we discuss in further detail in the next section) but in that case the trees to be constructed could be arbitrary. Unfortunately, we can show (see appendix) that any algorithm for constructing a scan-first search tree in the data stream model requires $\Omega(n^2)$ space even when there are no edge deletions.

To avoid this issue, we take a different approach based on finding arbitrary spanning trees for the induced graph on a random subset of vertices.[2] We will use the following result for finding these spanning trees.

**Theorem 2** (Ahn et al. [2])**.** *For a graph on $n$ vertices, there exists a vertex-based sketch of size $O(n \operatorname{polylog} n)$ from which we can construct a spanning forest with high probability.*

Note that in this section we restrict our attention to graphs rather than hypergraphs. However, in the next section we will explain how the vertex connectivity results extend to hypergraphs.

## 3.1 Warm-Up: Supporting Vertex Connectivity Queries

For $i = 1, 2, \ldots, R := 16 \cdot k^2 \ln n$, let $G_i$ be a graph formed by deleting each vertex in $G$ with probability $1 - 1/k$. Let $T_i$ be an arbitrary spanning forest of $G_i$ and define $H = T_1 \cup T_2 \cup \ldots \cup T_R$.

**Lemma 3.** *Let $S$ be an arbitrary collection of at most $k$ vertices. With high probability, $H \setminus S$ is connected iff $G \setminus S$ is connected.*

*Proof.* First we note that $H$ has the same set of vertices as $G$ with high probability. This follows because the probability a given vertex is not in $H$ is $(1 - 1/k)^R \leq \exp(-16 \cdot k \cdot \ln n) = n^{-16k}$ and hence by an application of the union bound, all vertices in $G$ are also in $H$ with probability at least $1 - n^{-(16k-1)}$. Then since $H$ is a subgraph of $G$, then $G \setminus S$ disconnected implies $H \setminus S$ disconnected. It remains to prove that $G \setminus S$ connected implies $H \setminus S$ connected.

Assume $G \setminus S$ is connected. Consider an arbitrary pair of vertices $s, t \notin S$ and let $s = v_0 \to v_1 \to v_2 \to \ldots \to v_\ell = t$ be a path between $s$ and $t$ in $G \setminus S$. Then note that there is a path between $v_i$ and $v_{i+1}$ in $H \setminus S$ if there exists $G_i$ such that $G_i \cap S = \emptyset$ and $v_i, v_{i+1} \in H \setminus S$. This follows because if $\{v_i, v_{i+1}\} \in G_i$ and $G_i \cap S = \emptyset$ then $T_i \setminus S$ either contains $\{v_i, v_j\}$ or a path between between $v_i$ and $v_j$. Hence,

$$\mathbb{P}\left[v_i \text{ and } v_{i+1} \text{ are connected in } T_i \setminus S\right] \geq 1/k^2(1 - 1/k)^k$$

and therefore

$$\mathbb{P}\left[v_i \text{ and } v_{i+1} \text{ are disconnected in } T_i \setminus S \text{ for all } i \in [R]\right] \quad \leq \quad (1 - 1/k^2(1 - 1/k)^k)^R \leq 1/n^4 \, .$$

Taking the union bound over all $\ell < n$ pairs $\{v_i, v_{i+1}\}$, we conclude that $s$ and $t$ are connected in $H \setminus S$ with probability at least $1 - 1/n^3$. By applying the union bound again, with probability at least $1 - 1/n^2$, $s$ is connected in $H \setminus S$ to all other vertices. □

[2] We note that the idea of subsampling vertices was recently explored by Censor-Hillel et al. [9, 10]. They showed that if each vertex of a $k$-vertex-connected graph is subsampled with probability $p = \Omega(\sqrt{\log n/k})$ then the resulting graph has vertex connectivity $\Omega(kp^2)$. We do not make use of this result in our work as it does not lead to an approximation factor better than $\sqrt{k}$.

Our algorithm constructs a spanning forest for each of $G_1, \ldots, G_R$ using the algorithm referenced in Theorem 2. Note that since each $G_i$ has $O(n/k)$ vertices with high probability, we can construct these $R$ trees in $R \times O(n/k \operatorname{polylog} n) = O(nk \operatorname{polylog} n)$ space. This gives us the following theorem.

**Theorem 4.** *There is a sketch-based dynamic graph algorithm that uses $O(kn \operatorname{polylog} n)$ space to test whether a set of vertices $S$ of size at most $k$ disconnects the graph. The query set $S$ is specified at the end of the stream.*

We next prove that the above query algorithm is space-optimal.

**Theorem 5.** *Any dynamic graph algorithm that allows us to test, with probability at least $3/4$, whether a queried set of at most $k$ vertices disconnects the graph requires $\Omega(kn)$ space.*

*Proof.* The proof is by a reduction from the communication problem of indexing [1]. Suppose Alice has a binary string $x \in \{0,1\}^{(k+1)\times n}$ indexed by $[k+1] \times [n]$ and Bob wants to compute $x_{i,j}$ for some index $(i,j) \in [k+1] \times [n]$ that is unknown to Alice. This requires $\Omega(nk)$ bits to be communicated from Alice to Bob if Bob is to be successful with probability at least $3/4$. Consider the protocol where the players create a bipartite graph on vertices $L \cup R$ where $L = \{l_1, \ldots, l_{k+1}\}$ and $R = \{r_1, \ldots, r_n\}$. Alice adds edges $\{l_i, r_j\}$ for all pairs $(i,j)$ such that $x_{i,j} = 1$. Alice runs the algorithm and sends the state to Bob. Bob adds edges $\{r_\ell, r_{\ell'}\}$ for all $\ell, \ell' \neq j$ and deletes all vertices in $L$ except $l_i$. Now $r_j$ is connected to the rest of the graph iff the $x_{i,j} = 1$. □

### 3.2 $k$-vertex connectivity

For $i = 1, 2, \ldots, R := 160 \cdot k^2 \epsilon^{-1} \ln n$, let $G_i$ be a graph formed by deleting each vertex in $G$ with probability $1 - 1/k$. As before, let $T_i$ be an arbitrary spanning forest of $G_i$ and define $H = T_1 \cup T_2 \cup \ldots \cup T_R$.

**Theorem 6.** *Let $S$ be a subset of $V$ of size $k$. Consider any pair of vertices $u, v \in V \setminus S$ such that there are at least $(1+\epsilon)k$ vertex-disjoint paths between $u$ and $v$ in $G$. Then,*

$$\mathbb{P}\left[u \text{ and } v \text{ are connected in } G_S\right] \geq 1 - 4/n^{10k}$$

*where $G_S = \cup_{i \in U(S)} G_i$ and $U(S) = \{i : G_i \cap S = \emptyset\}$ is the set of sampled graphs with no vertices in $S$.*

*Proof.* We first argue that $|U(S)|$ is large with high probability. Then $\mathbb{E}\left[|U(S)|\right] = (1-1/k)^k R \geq R/4$. By an application of the Chernoff bound:

$$\mathbb{P}\left[|U(S)| \leq 1/2 \times R/4\right] \leq e^{-1/4 \times R/4 \times 1/3} < 1/n^{10k} \ .$$

In the rest of the proof we condition on event $|U(S)| \geq r := R/8$.

Note that there are $t \geq \epsilon k$ vertex-disjoint paths between $u$ and $v$ in $G \setminus S$. Call these paths $P_1, \ldots, P_t$. For each $P_i$, let $a_i$ be the edge incident to $u$, let $c_i$ be the edge incident to $v$, and let $B_i$ be the remaining edges in $P_i$. Note that $a_i$ and $c_i$ need not be distinct and $B_i$ could be empty.

**Claim.** *The followings three probabilities are each larger than $1 - 1/n^{10k}$:*

$$\mathbb{P}\left[a_i \in G_S \text{ for at least } 3t/4 \text{ values of } i\right] \ , \ \mathbb{P}\left[B_i \subseteq G_S \text{ for at least } 3t/4 \text{ values of } i\right]$$

$$\mathbb{P}\left[c_i \in G_S \text{ for at least } 3t/4 \text{ values of } i\right] \ .$$

*Proof of Claim.* Each edge in $B_i$ is not present in $G_S$ with probability $(1-1/k^2)^r$. Hence, by the union bound, $\mathbb{P}[B_i \not\subseteq G_S] \leq |B_i|(1-1/k^2)^r$. Also by the union bound,

$$\mathbb{P}[B_i \not\subseteq G_S \text{ for more than } t/4 \text{ values of } i] < \binom{t}{t/4}(n(1-1/k^2)^r)^{t/4} < e^{t\ln 2+(\ln n - r/k^2)t/4} < 1/n^{10k} .$$

The proofs for $a_i$ and $c_i$ are entirely symmetric so we just consider $a_i$. Consider the set $U'(S) = U(S) \cap \{j : u \in G_j\}$. Note that for $j \in U'(S)$ we have $\mathbb{P}[a_i \in G_j] = 1/k$ and by the union bound,

$$\mathbb{P}\left[a_i \notin \cup_{j\in U'(S)} G_j \text{ for at least } t/4 \text{ values of } i\right] \leq \binom{t}{t/4}(1-1/k)^{|U'(S)|t/4} \leq 2^t \exp\left(\frac{-|U'(S)|t}{(4k)}\right) .$$

Let $E$ be the event that $|U'(S)| \leq |U(S)|/(2k)$. Then, by an application of the Chernoff bound:

$$\begin{aligned}
& \mathbb{P}[a_i \notin G_S \text{ for at least } t/4 \text{ values of } i] \\
\leq\ & \mathbb{P}[E] + \mathbb{P}\left[a_i \notin \cup_{j\in U'(S)} G_j \text{ for at least } t/4 \text{ values of } i \mid \neg E\right] \\
\leq\ & \exp(-1/4 \times |U(S)|/k \times 1/3) + \mathbb{P}\left[a_i \notin \cup_{j\in U'(S)} G_j \text{ for at least } t/4 \text{ values of } i \mid \neg E\right] \\
\leq\ & \exp(-1/4 \times r/k \times 1/3) + 2^t \exp(-r/(2k) \times t/(4k)) < 1/n^{10k} .
\end{aligned}$$

□

It follows from the claim that there exists $i$ such that $P_i \in G_S$ (and therefore $u$ and $v$ are connected in $G_S$) with probability at least $1 - 3/n^{10k}$. The conditioning on $|U(S)| \geq r$ decreases this by another $1/n^{10k}$. □

**Corollary 7.** *If $G$ is $(1+\epsilon)k$-vertex-connected then $H$ is $k$-vertex-connected with high probability. If $H$ is $k$-vertex connected then $G$ is $k$-vertex connected.*

*Proof.* The first part of the corollary follows from Theorem 6 by applying the union bound over all $O(n^k)$ subsets of size at most $k$ and $O(n^2)$ choices of $u$ and $v$. Note that $u$ and $v$ connected in $G_S$ implies $u$ and $v$ are connected in $H$ since $H$ includes a spanning forest of $G_S$. The second part is implied by the fact $H$ is a subgraph of $G$. □

As in the previous section, our algorithm is simply to construct $H$ be using the algorithm referenced in Theorem 2 to construct $T_1, \ldots, T_R$. We can then run any vertex connectivity algorithm on $H$ in post-processing. Since each $G_i$ has $O(n/k)$ vertices with high probability, we can construct these $R$ trees in $R \times O(n/k \cdot \operatorname{polylog} n) = O(nk\epsilon^{-1} \operatorname{polylog} n)$ space. This gives us the following theorem.

**Theorem 8.** *There is a sketch-based dynamic graph algorithm that uses $O(kn\epsilon^{-1} \operatorname{polylog} n)$ space to distinguish $(1+\epsilon)k$-vertex connected graphs from $k$-connected graphs.*

# 4 Reconstructing Cut-Degenerate Hypergraphs

We next present sketches for reconstructing cut-degenerate hypergraphs. Recall that a hypergraph is $d$-degenerate if all induced subgraphs have a vertex of degree at most $d$. Cut-degeneracy is defined as follows.

**Definition 9.** *A hypergraph is* $d$-cut-degenerate *if every induced subgraph has a cut of size at most $d$.*

The following lemma establishes that this is a strictly weaker property than $d$-degeneracy.

**Lemma 10.** *Any hypergraph that is $d$-degenerate is also $d$-cut-degenerate. There exists graphs that are $d$-cut-degenerate but not $d$-degenerate.*

*Proof.* Since the degree of a vertex $v$ is exactly the size of the cut $(\{v\}, V \setminus \{v\})$ it is immediate that $d$-degeneracy implies $d$-cut-degeneracy. For an example that $d$-cut-degenerate does not imply it is $d$-degenerate consider the graph $G$ on eight vertices $\{v_1, v_2, v_3, v_4, u_1, u_2, u_3, u_4\}$ with edges $\{v_i, v_j\}, \{u_i, u_j\}$ for all $i, j$ except $i = 1, j = 4$ and edges $\{v_1, u_1\}$ and $\{v_4, u_4\}$. Then $G$ has minimum degree 3 and is therefore not 2-degenerate while it is 2-cut-degenerate. □

Becker et al. [5] show that is possible to reconstruct a $d$-degenerate graph in the simultaneous communication model using a $O(d \operatorname{polylog} n)$ bit message from each player. We will show that it is also possible to reconstruct any $d$-cut-degenerate graph using $O(d \operatorname{polylog} n)$ bit messages. Even if the graph is not cut-degenerate, we show that is possible to reconstruct all edges with a certain connectivity property. We will subsequently use this fact in Section 5.

### 4.1 $k$-Skeletons for Hypergraphs

We first review the existing results on constructing $k$-skeletons [2] that we will need for our new results. In doing so, we generalize the previous work to the case of hypergraphs. In particular, this leads to the first dynamic graph algorithm for determining hypergraph connectivity.

**Definition 11** ($k$-skeleton)**.** *Given a hypergraph $H = (V, E)$, a subgraph $H' = (V, E')$ is a* $k$-skeleton *of $H$ if for any $S \subset V$, $|\delta_{H'}(S)| \geq \min(|\delta_H(S)|, k)$.*

In particular, any spanning graph is a 1-skeleton and it can be shown that $F_1 \cup F_2 \cup \ldots \cup F_k$ is a $k$-skeleton [2] of $G$ if $F_i$ is a spanning graph of $G \setminus (\cup_{j=1}^{i-1} F_j)$. The next lemma establishes that given an arbitrary $k$-skeleton of a graph we can exactly determine the set of edges with $\lambda_e(G) \leq k - 1$.

**Lemma 12.** *Let $H$ be a $k$-skeleton of $G$ then $\lambda_e(H) \leq k - 1$ iff $\lambda_e(G) \leq k - 1$.*

*Proof.* Since $H$ is a subgraph $\lambda_e(H) \leq \lambda_e(G)$ and hence $\lambda_e(G) \leq k - 1$ implies $\lambda_e(H) \leq k - 1$. Using the fact that $H$ is a $k$-skeleton $\lambda_e(H) \geq \min(k, \lambda_e(G))$ and hence, if $\lambda_e(H) \leq k - 1$ it must be that $\lambda_e(G) \leq k - 1$. □

**Constructing Spanning Graphs.** For each vertex $v_i \in V$, define the vector $\mathbf{a}^i \in \{-1, 0, 1, 2, \ldots, r-1\}^d$ where $d = \sum_{i=2}^{r} \binom{n}{i}$ is the number of possible hyperedges of size at most $r$:

$$\mathbf{a}^i_e = \begin{cases} |e| - 1 & \text{if } i = \min e \text{ and } e \in E \\ -1 & \text{if } i \in e \setminus \min e \text{ and } e \in E \\ 0 & \text{otherwise} \end{cases}$$

where $e$ ranges over all subsets of $V$ of size between 2 and $r$ and $\min e$ denotes the smallest ID of a node in $e$. Observe that these vectors have the property that for any subset of vertices $\{v_i\}_{i \in S}$, the non-zero entries of $\sum_{i \in S} \mathbf{a}^i$ correspond exactly to $\delta(S)$. This follows because the only subsets of

$$\{|e| - 1, \underbrace{-1, -1, \ldots, -1}_{|e|-1}\}$$

that sum to zero are the empty set and the entire set. Hence, the $e$-th coordinate of $\sum_{i \in S} \mathbf{a}^i$ is zero iff either $e \notin E$ or $e \subset S$ or $e \subset V \setminus S$.

The rest of algorithm proceeds exactly as in the case of (non-hyper) graphs [2] and a reader that is very familiar with the previous work should feel free to skip the remainder of Section 4.1. We construct the sketches $M\mathbf{a}^1, \ldots, M\mathbf{a}^n$ where $M$ is chosen according to a distribution over matrices $\mathbb{R}^{k \times d}$ where $k = \text{polylog}(d)$. The distribution has the property that for any $\mathbf{a} \in \mathbb{R}^d$, it is possible to determine the index of a non-zero entry of $\mathbf{a}$ given $M\mathbf{a}$ with probability $1 - 1/\text{poly}(n)$. Such as distribution is known to exist by a result of Jowhari et al. [18]. Given $M\mathbf{a}^1, \ldots, M\mathbf{a}^n$ we can find an edge across an arbitrary cut $(S, V \setminus S)$. To do this, we compute $\sum_{i \in S} M\mathbf{a}^i = M(\sum_{i \in S} \mathbf{a}^i)$. We can then determine the index of a non-zero entry of $\sum_{i \in S} \mathbf{a}^i$ which corresponds to an element of $\delta(S)$ as required. It may appear that to test connectivity we need to test all $2^{n-1} - 1$ possible cuts. Since the failure probability for each cut is only inverse polynomial in $n$ this would be problematic. However, it is possible to be more efficient and only test $O(n)$ cuts. See Ahn et al. [2] for details.

**Theorem 13** (Spanning Graph Sketches)**.** *There exists a vertex-based sketch $\mathcal{A}$ of size $O(n \operatorname{polylog} n)$ such that we can find a spanning graph of a hypergraph $G$ from $\mathcal{A}(G)$ with high probability.*

Note the above theorem can be substituted for Theorem 2 and the resulting algorithms for vertex connectivity go through for hypergraphs unchanged.

**Constructing $k$-skeletons.** As mentioned above, it suffices to find $F_1, \ldots, F_k$ such that $F_i$ is a spanning graph of $G \setminus (\cup_{j=1}^{i-1} F_j)$. Do to this we use $k$ independent spanning graph sketches $\mathcal{A}^1(G), \mathcal{A}^2(G), \ldots, \mathcal{A}^k(G)$ as described in the previous section. We may construct $F_1$ from $\mathcal{A}^1(G)$ because this is the functionality of a spanning graph sketch. Then, assuming we have already constructed $F_1, \ldots, F_{i-1}$ we can construct $F_i$ from:

$$\mathcal{A}^i(G - F_1 - F_2 \ldots - F_{i-1}) = \mathcal{A}^i(G) - \sum_{j=1}^{i-1} \mathcal{A}^i(F_j) \,.$$

**Theorem 14** ($k$-Skeleton Sketches)**.** *There exists a vertex-based sketch $\mathcal{B}$ of size $O(kn \operatorname{polylog} n)$ such that we can find of a $k$-skeleton a hypergraph $G$ from $\mathcal{B}(G)$ with high probability.*

## 4.2 Beyond $k$-Skeletons and Reconstructing Cut-Degenerate Hypergraphs

One might be tempted as ask whether it was necessary to use $k$ independent spanning graph sketches $\mathcal{A}^1, \ldots, \mathcal{A}^k$ rather that reuse a single sketch $\mathcal{A}$. If each application of the sketch $\mathcal{A}$ fails to return a spanning graph with probability $\delta$, one might hope to use the union bound to argue that the probability that $\mathcal{A}$ fails on any of the inputs $G, G - F_1, G - F_1 - F_2, \ldots, G - F_1 - \ldots - F_{k-1}$ is at most $k\delta$. *But this would not be a valid application of the union bound!* The union bound states that for any *fixed* set of $t$ events $B_1, \ldots, B_t$, we have $\mathbb{P}[B_1 \cup \ldots \cup B_t] \leq \sum_i \mathbb{P}[B_i]$. The issue is that the events in the above example are not fixed, i.e., they can not be specified a priori, since spanning graph $F_i$ is determined by the randomness in the sketch.[3] We belabor this point because, while the union bound was not applicable in the above case, we will need it to prove our next result in a situation that is only subtly different and yet the union bound *is* valid.

[3] Another way to see that using the same sketch cannot work is that if it were possible to repeatedly remove each spanning graph from the sketch of the original graph, we would be able to reconstruct the entire graph using only a sketch of size $O(n \operatorname{polylog} n)$. Clearly this is not possible because it requires at $\Omega(n^2)$ bits to specify an arbitrary graph on $n$ vertices.

### 4.2.1 Finding the light edges

Given a graph $G = (V, E)$ and a postive integer $k$, recursively define

$$E_i = \{e \in E : \lambda_e(G \setminus \bigcup_{j=1}^{i-1} E_i) \leq k\}$$

and denote the union of these sets as:

$$\text{light}_k(G) = \bigcup_{i \geq 1} E_i \ .$$

Note that if $G$ is $d$ cut-degenerate then $\text{light}_d(G) = E$. Furthermore, there is at most $n$ values of $i$ such that $E_i$ is non-empty since removing each non-empty set $E_i$ from the graph increases the number of connected components.

Suppose $\mathcal{B}(G)$ is a sketch that returns an arbitrary $(k+1)$-skeleton of $G$ with failure probability $\delta = 1/\operatorname{poly}(n)$. Then, since $E_1, E_2, \ldots, E_n$ are sets defined solely by the input graph (and not any randomness in a sketch) we can specify the fixed events

$$B_i = \text{“We fail to return a } (k+1)\text{-skeleton sketch of } G - E_1 - \ldots - E_i \text{ given } \mathcal{B}(G - E_1 - \ldots - E_i)\text{”}$$

and therefore use the union bound to establish that the probability that we find a $(k+1)$-skeleton of each of the relevant graphs with failure probability at most $n\delta = 1/\operatorname{poly}(n)$.

We can therefore find the sets $E_1, E_2, \ldots, E_n$ as follows. Let $S_i$ be an arbitrary $(k+1)$ skeleton of $G - E_1 - \ldots E_{i-1}$. Assuming we have already determined $E_1, \ldots, E_{i-1}$, we can find $S_i$ using:

$$\mathcal{B}(G - E_1 - E_2 \ldots - E_{i-1}) = \mathcal{B}(G) - \sum_{j=1}^{i-1} \mathcal{B}(E_j) \ .$$

Then, by appealing to Lemma 12, we know that we can then uniquely determine $E_i$ given $S_i$.

**Theorem 15.** *There exists a vertex-based sketch of size $O(kn \operatorname{polylog} n)$ from which $\text{light}_k(G)$ can be reconstructed for any hypergraph $G$. In the case of a $k$-cut-degenerate graph, this is the entire graph.*

### 4.2.2 What are the light edges?

In this section, we restrict our attention to graphs rather than hypergraphs and show that the set of edges in $\text{light}_k(G)$ can be defined in terms of the notion of *strong connectivity* introduced by Benczúr and Karger [6].

**Lemma 16.** $\text{light}_k(G) = \{e : k_e \leq k\}$ *where $k_{\{u,v\}}$ is the maximum $k$ such that there is a set $S \subset V$ including $u$ and $v$ such that the induced graph on $S$ is $k$-edge-connected.*

*Proof.* Define $t_e$ to be the minimum value of $k$ such that $e \in \text{light}_k(G)$. We prove that $t_e = k_e$ and the result follows. To show $k_e \geq t_e$ suppose $t_e = t$ and then note that $e$ survives when we recursively remove edges with edge connectivity $t-1$. But the remaining components in this graph are at least $(t-1)+1 = t$ connected so $k_e \geq t$. To show that $k_e \leq t_e$, suppose $k_e = k$. Then there exists a vertex induced subgraph $H$ containing $e$ that is $k$-connected. But when we recursively remove edges with edge connectivity at most $k-1$ then no edge in $H$ can be removed. Hence, $t_e > (k-1)$ and so $t_e \geq k$. □

# 5 Hypergraph Sparsification

In this final section, we present a vertex-based sketch for constructing a sparsifier of a hypergraph. This yields the first dynamic graph stream algorithm for constructing a sparsifier of a hypergraph. As an added bonus, our approach gives an algorithm and analysis that is significantly simpler than previous work on the specific case of graph sparsification [3, 16].

**Definition 17** (Hypergraph Sparsifier)**.** *A weighted subgraph $H = (V, E', w)$ of a hypergraph $G = (V, E)$ is a* sparsfier *if for all $S \subset V$, $\sum_{e \in \delta_H(S)} w(e) = (1 \pm \epsilon)|\delta_G(S)|$.*

Previous approaches to sparsification in the dynamic stream model relied on work by Fung et al. [15]. To construct a *graph* sparsifier, they showed that it was sufficient to independently sample every edge in the graph with probability $O(\epsilon^{-2}\lambda_e^{-1} \log n)$. Using their work required coopting their machinery and modifying it appropriately (e.g., replacing Chernoff arguments with careful Martingale arguments). Another downside to the previous approach is that the Fung et al. result does not seem to extend to the case of hypergraphs.[4]

Using our new-found ability (see the previous section) to find the entire set of edges that are not $k$-strong, we present an algorithm that a) has a simpler, and almost self-contained, analysis and b) extends to hypergraphs. Our approach is closer in spirit to Benczúr and Karger's original work on sparsification [6] which in turn is based on the following result by Karger [22]: if we sample each edge with probability $p \geq p^* = c\epsilon^{-2}\lambda^{-1} \log n$ where $\lambda$ is the cardinality of the minimum cut and $c \geq 0$ is some constant, and weight the sampled edges by $1/p$ then the resulting graph is a sparsifier with high probability.

The idea behind our algorithm is as follows. For a hypergraph $G$, if we remove the hyperedges $\text{light}_k(G)$ where $k = 2c\epsilon^{-2} \log n$, then every connected component in the remaining hypergraph has minimum cut of size greater than $2c\epsilon^{-2} \log n$. Hence, for each of these components $p^* \leq 1/2$. Therefore, the graph formed by sampling the hyperedges in $G \setminus \text{light}_k(G)$ with probability 1/2 (and doubling the weight of sampled hyperedges) and adding the set of hyperedges in $\text{light}_k(G)$ with unit weights is a sparsifier of $G$. We then repeat this process until there are no hyperedges left to sample.

**Algorithm.**

1. Generate a series of graphs $G_0, G_1, G_2 \ldots$ where $G_i$ is formed by deleting each hyperedge in $G_{i-1}$ independently with probability 1/2 and $G_0 = G$.
2. For $i = 0, 1, 2, \ldots, \ell = 3 \log n$:
   (a) Let $F_i = \text{light}_k(H_i)$ where $k = O(\epsilon^{-2}(\log n + r))$ where $H_i = G_i \setminus (F_0 \cup F_1 \cup F_2 \cup \ldots \cup F_{i-1})$
3. Return $\bigcup_{i=0}^{\ell} 2^i \cdot F_i$ where $2^i \cdot F_i$ is the set of hyperedges in $F_i$ where each is given weight $2^i$.

**Analysis.** The following lemma uses an argument due to Karger [21] combined with a hypergraph cut counting result by Kogan and Krauthgamer [23].

**Lemma 18.** *$2H_{i+1} \cup F_i$ is a $(1 + \epsilon)$-sparsifier for $H_i$.*

[4] For the reader familiar with Fung et al. [15], the issue is finding a suitable definition of cut-projection for hypergraphs and then proving a bound on the number of distinct cut-projections.

*Proof.* It suffices to prove that $2H_{i+1}$ is a $(1+\epsilon)$-sparsifier for $H_i \setminus F_i$. Furthermore, it suffices to consider each connected component of $H_i \setminus F_i$ separately.

Let $C$ be an arbitrary connected component of $H_i \setminus F_i$ and note that $C$ has a minimum cut of size at least $k$. Let $C'$ be the graph formed by deleting each hyperedge in $C$ with probability $1/2$. Consider a cut of size $t$ in $C$ and let $X$ be the number of hyperedges in this cut that are in $C'$. Then $\mathbb{E}[X] = t/2$ and by an application of the Chernoff bound, $\mathbb{P}[|X - t/2| \geq \epsilon t/2] \leq 2\exp(-\epsilon^2 t/6)$.

The number of cuts of size at most $t$ is $\exp(O(rt/k + t/k \cdot \log n))$ by appealing to a result by Kogan and Krauthgamer [23]. By an application of the union bound, the probability that there exists a cut of size $t$ such that the number of hyperedges in corresponding cut in $C'$ is not $(1 \pm \epsilon)t/2$ is at most

$$2\exp(-\epsilon^2 t/6) \cdot \exp(O(rt/k + t/k \cdot \log n)) .$$

This probability is less than $1/n^{10}$ if $k \geq c\epsilon^{-2}(\log n + r)$ for some sufficiently large constant $c$. Hence, taking the union bound over all $t \geq k$ ensures that with probability at least $1/n^8$, for every cut in $C$, the fraction of edges in the corresponding cut in $C'$ is $(1 \pm \epsilon)/2$. □

**Theorem 19.** *$\bigcup_{i=0}^{\ell} 2^i \cdot F_i$ is a $(1+\epsilon)^\ell$-sparsifier of $G$ where $\ell = 3\log n$.*

*Proof.* The theorem follows by repeatedly applying Lemma 18. Specifically,

1. $F_{\ell-1}$ is a $(1+\epsilon)$ sparsifier for $H_{\ell-1}$ since $H_\ell$ is the empty graph with high probability.
2. $2H_{\ell-1} \cup F_{\ell-2}$ is a $(1+\epsilon)$-sparsifier for $H_{\ell-2}$ and so $2F_{\ell-1} \cup F_{\ell-2}$ is a $(1+\epsilon)^2$-sparsifier for $H_{\ell-2}$
3. $2H_{\ell-2} \cup F_{\ell-3}$ is a $(1+\epsilon)$-sparsifier for $H_{\ell-3}$ and so $4F_{\ell-1} \cup 2F_{\ell-2} \cup F_{\ell-3}$ is a $(1+\epsilon)^3$-sparsifier for $H_{\ell-3}$

We continue in this way until we deduce $\bigcup_{i=0}^{\ell} 2^i \cdot F_i$ is a $(1+\epsilon)^\ell$-sparsifier for $H_0 = G_0$. □

By re-parameterizing $\epsilon \leftarrow \epsilon/(2\ell)$ and using the sketches from Section 4, we establish the next theorem.

**Theorem 20.** *There exists a vertex-based sketch of size $O(\epsilon^{-2} n \operatorname{polylog} n)$ from which we can construct a $(1+\epsilon)$ hypergraph sparsifier.*

**Acknowledgements.** We thank Jennifer Chayes for prompting us to investigate hypergraph connectivity.

# References

[1] F. M. Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 157(2):139–159, 1996.

[2] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 459–467, 2012.

[3] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 5–14, 2012.

[4] K. J. Ahn, S. Guha, and A. McGregor. Spectral sparsification in dynamic graph streams. In *APPROX*, pages 1–10, 2013.

[5] F. Becker, M. Matamala, N. Nisse, I. Rapaport, K. Suchan, and I. Todinca. Adding a referee to an interconnection network: What can(not) be computed in one round. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011*, pages 508–514, 2011.

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

[7] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

[8] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distrib. Comput.*, 69(8):711–724, 2009.

[9] K. Censor-Hillel, M. Ghaffari, G. Giakkoupis, B. Haeupler, and F. Kuhn. Tight bounds on vertex connectivity under vertex sampling. In *ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, 2015.

[10] K. Censor-Hillel, M. Ghaffari, and F. Kuhn. A new perspective on vertex connectivity. In *Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 546–561, 2014.

[11] J. Cheriyan, M. Y. Kao, and R. Thurimella. Scan-first search and sparse certificates: an improved parallel algorithm for k-vertex connectivity. *SIAM Journal on Computing*, 22(1):157–174, 1993.

[12] Y. Emek and A. Rosén. Semi-streaming set cover - (extended abstract). In *ICALP*, pages 453–464, 2014.

[13] 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.

[14] U. Feige, M. Hajiyaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separators. *Proc. of STOC*, 2005.

[15] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80, 2011.

[16] A. Goel, M. Kapralov, and I. Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012.

[17] Y. Huang, Q. Liu, and D. N. Metaxas. Video object segmentation by hypergraph cut. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 1738–1745, 2009.

[18] 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.

[19] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. In *FOCS*, 2014.

[20] M. Kapralov and D. P. Woodruff. Spanners and sparsifiers in dynamic streams. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 272–281, 2014.

[21] D. R. Karger. Random sampling in cut, flow, and network design problems. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 648–657, New York, NY, USA, 1994. ACM.

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

[23] D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *6th Innovations in Theoretical Computer Science*, 2015.

[24] K. Kutzkov and R. Pagh. Triangle counting in dynamic graph streams. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 306–318, 2014.

[25] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

[26] J. Radhakrishnan and S. Shannigrahi. Streaming algorithms for 2-coloring uniform hypergraphs. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, pages 667–678, 2011.

[27] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, pages 697–708, 2009.

[28] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126, 2007.

[29] H. Sun. Counting hypergraphs in data streams. *CoRR*, abs/1304.7456, 2013.

[30] Y. Yamaguchi, A. Ogawa, A. Takeda, and S. Iwata. Cyber security analysis of power networks by hypergraph cut algorithms. 2014.

# A Scan-First Trees

A scan first search tree (SFST) of a graph [11] is defined as follows: The tree is initially empty, all vertices except the root (chosen arbitrarily) are *unmarked*, and all vertices are *unscanned*. At each step we *scan* an marked but unscanned vertex. For each vertex $x$ that is being scanned, all edges from $x$ to unmarked neighbors of $x$ are added to the tree and the unmarked neighbors are marked. This continues until no marked but unscanned vertices remain.

**Theorem 21.** *Any data stream algorithm that constructs a SFST with probability at least* $3/4$ *requires* $\Omega(n^2)$ *space.*

*Proof.* The proof is by a reduction from the communication problem of indexing [1]. Suppose Alice has a binary string $x \in \{0,1\}^{n^2}$ indexed by $[n]\times[n]$ and Bob wants to compute $x_{i,j}$ for some index $(i,j) \in [n]\times[n]$ that is unknown to Alice. This requires $\Omega(n^2)$ bits to be communicated from Alice to Bob if Bob is to learn $x_{i,j}$ with probability at least $3/4$. Suppose we have a data stream algorithm for constructing an SFST. Alice creates a graph on nodes $T \cup U \cup V \cup W$ where $T = \{t_1, \ldots, t_n\}, U = \{u_1, \ldots, u_n\}, V = \{v_1, \ldots, v_n\}$, and $W = \{w_1, \ldots, w_n\}$. She adds edges $\{t_k, u_\ell\}$ and $\{v_\ell, t_k\}$ for each $\ell, k$ such that $x_{\ell,k} = 1$. Alice runs the scan-first search algorithm and sends the contents of her memory to Bob. Bob adds the edge $\{u_i, v_i\}$. Note that any SFST includes all neighbors of $u_i$ or $v_i$. In particular, $x_{i,j} = 1$ iff at least one of $\{t_j, u_i\}$ or $\{v_i, w_j\}$ is present in the SFST constructed. Hence, the algorithm must have used $\Omega(n^2)$ space. □