Single Pass Spectral Sparsification in Dynamic Streams

2014.11.10

M. Kapralov, Y.T. Lee, C. Musco, C. Musco, A. Sidford
Massachusetts Institute of Technology

1-Pass Spectral Sparsification in Dynamic Streams

Overview

- In $\tilde{O}(n)$ space, maintain a graph compression from which we can always return a spectral sparsifier.

Main technique

- Use ℓ_2 heavy hitter sketches to sample by effective resistance in the streaming model.

1-Pass Spectral Sparsification in Dynamic Streams

Overview

- In $\tilde{O}(n)$ space, maintain a graph compression from which we can always return a spectral sparsifier.

Main technique

- Use ℓ_2 heavy hitter sketches to sample by effective resistance in the streaming model.

Outline

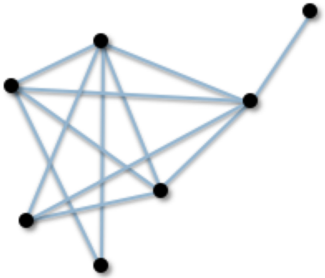
1. Graph Sparsification
2. Semi-Streaming Computational Model
3. Prior Work Review
4. Our Algorithm
   - Recover High Effective Resistance Edges
   - Sampling by Effective Resistance
   - Recursive Sparsification [Li, Miller, Peng '12]

Overview

Graph Sparsification

General Idea

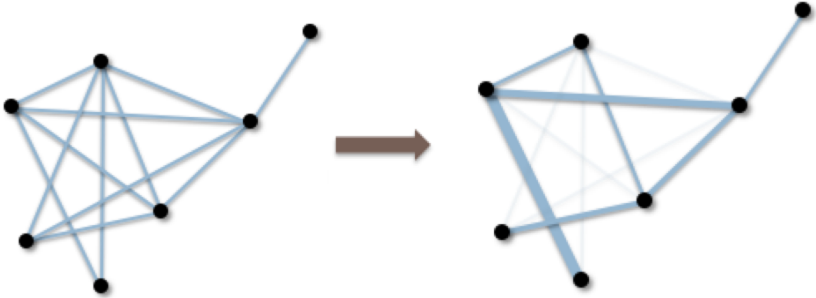
- Approximate a dense graph with a much sparser graph.
- Reduce $O(n^2)$ edges $\to$ $O(n \log n)$ edges



Graph Sparsification

General Idea

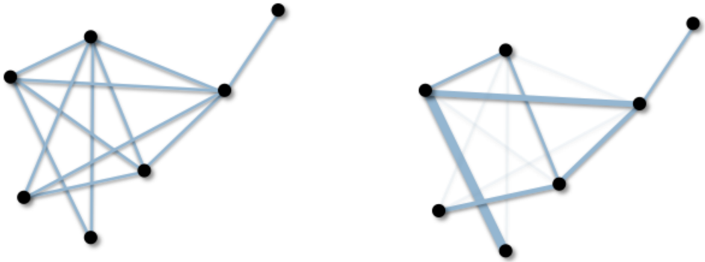
- Approximate a dense graph with a much sparser graph.
- Reduce $O(n^2)$ edges $\rightarrow$ $O(n \log n)$ edges



Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

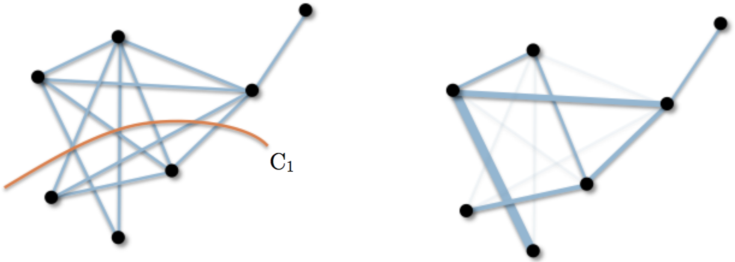


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

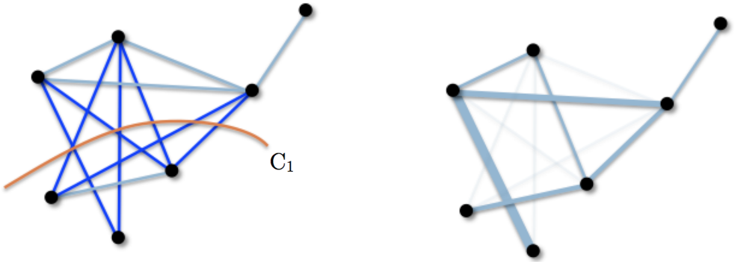


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger ’96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

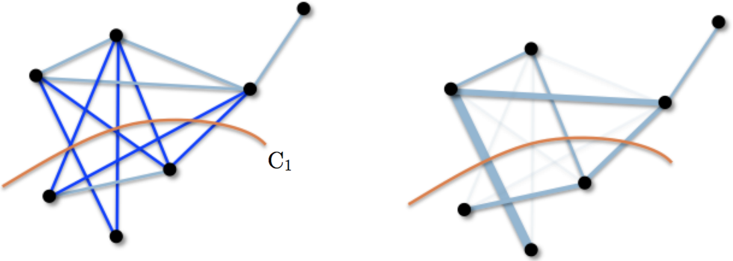


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

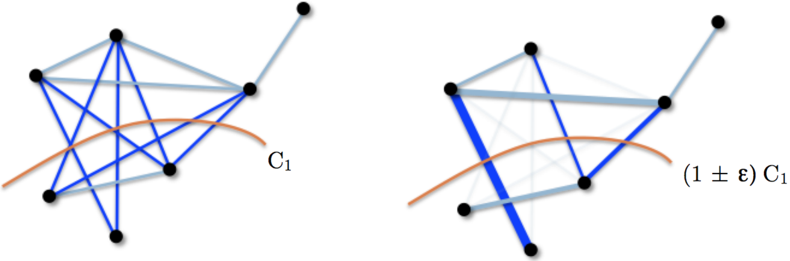


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

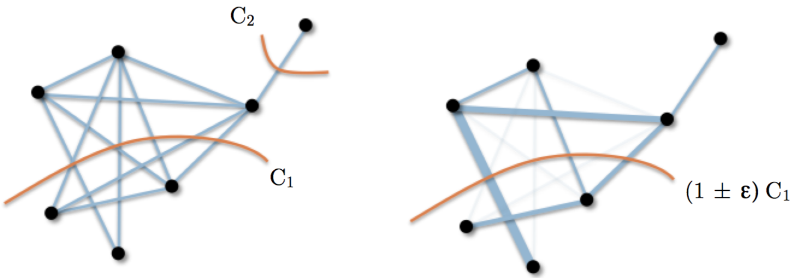


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

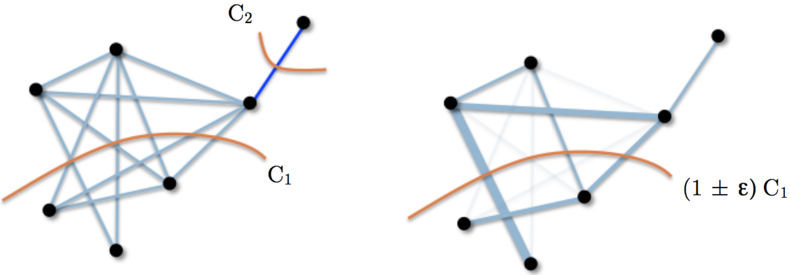


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

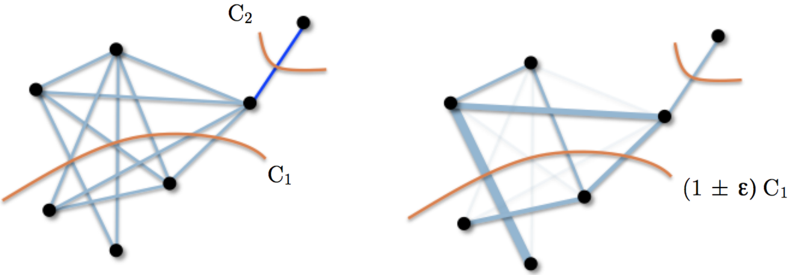


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

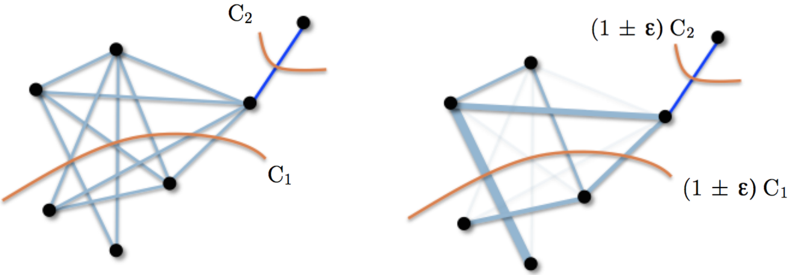


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Preserve *every* cut value to within $(1 \pm \varepsilon)$ factor

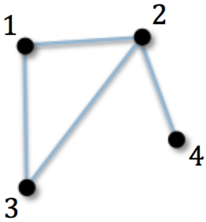


Applications: Minimum cut, sparsest cut, etc.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

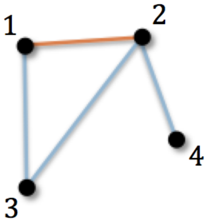


$$\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{B} \end{array}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



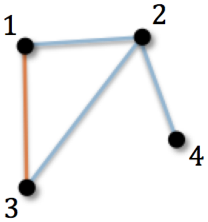
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	1	0	-1	0
e_{14}	0	0	0	0
e_{23}	0	1	-1	0
e_{24}	0	1	0	-1
e_{34}	0	0	0	0

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



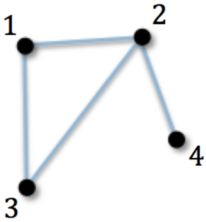
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	**1**	**0**	**-1**	**0**
e_{14}	0	0	0	0
e_{23}	0	1	-1	0
e_{24}	0	1	0	-1
e_{34}	0	0	0	0

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



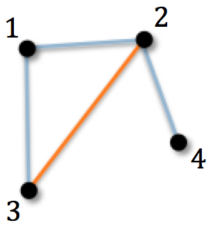
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	1	0	-1	0
e_{14}	**0**	**0**	**0**	**0**
e_{23}	0	1	-1	0
e_{24}	0	1	0	-1
e_{34}	0	0	0	0

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



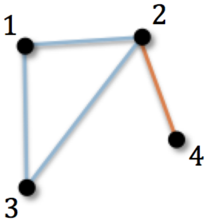
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	1	0	-1	0
e_{14}	0	0	0	0
e_{23}	**0**	**1**	**-1**	**0**
e_{24}	0	1	0	-1
e_{34}	0	0	0	0

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



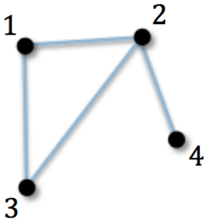
$$\mathbf{B} = \begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



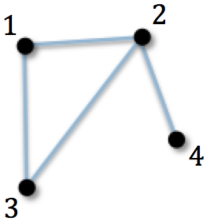
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	1	0	-1	0
e_{14}	0	0	0	0
e_{23}	0	1	-1	0
e_{24}	0	1	0	-1
e_{34}	**0**	**0**	**0**	**0**

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.



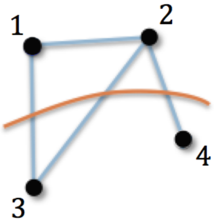
	v_1	v_2	v_3	v_4
e_{12}	1	-1	0	0
e_{13}	1	0	-1	0
e_{14}	0	0	0	0
e_{23}	0	1	-1	0
e_{24}	0	1	0	-1
e_{34}	0	0	0	0

B

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

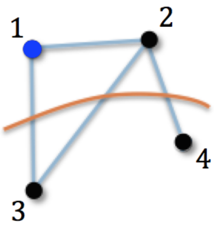


$$
\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array}
\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\
\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{B} \end{array}
\times
\begin{array}{c} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\ \mathbf{x} \end{array}
$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

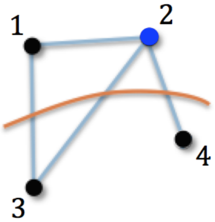


$$\underbrace{\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline e_{12} & 1 & -1 & 0 & 0 \\ e_{13} & 1 & 0 & -1 & 0 \\ e_{14} & 0 & 0 & 0 & 0 \\ e_{23} & 0 & 1 & -1 & 0 \\ e_{24} & 0 & 1 & 0 & -1 \\ e_{34} & 0 & 0 & 0 & 0 \end{array}}_{\mathbf{B}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

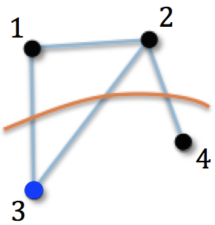


$$
\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array}
\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\
\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{B}} \end{array}
\times
\underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}}
$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

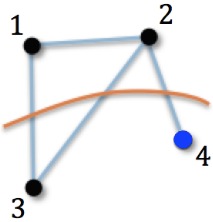


$$\underbrace{\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline e_{12} & 1 & -1 & 0 & 0 \\ e_{13} & 1 & 0 & -1 & 0 \\ e_{14} & 0 & 0 & 0 & 0 \\ e_{23} & 0 & 1 & -1 & 0 \\ e_{24} & 0 & 1 & 0 & -1 \\ e_{34} & 0 & 0 & 0 & 0 \end{array}}_{\mathbf{B}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

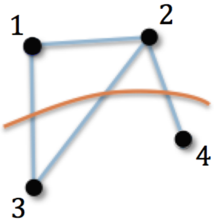


$$\underbrace{\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline e_{12} & 1 & -1 & 0 & 0 \\ e_{13} & 1 & 0 & -1 & 0 \\ e_{14} & 0 & 0 & 0 & 0 \\ e_{23} & 0 & 1 & -1 & 0 \\ e_{24} & 0 & 1 & 0 & -1 \\ e_{34} & 0 & 0 & 0 & 0 \end{array}}_{\mathbf{B}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

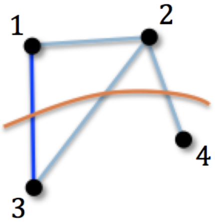


$$\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \overset{\begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array}}{\underset{\mathbf{B}}{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}} \times \underset{\mathbf{x}}{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

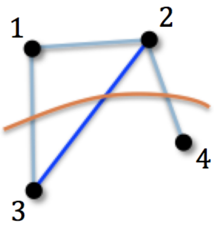


$$\underbrace{\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}}_{\mathbf{B}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

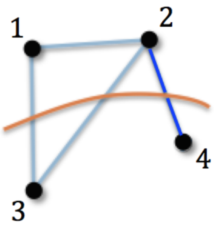


$$\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{B}}^{\begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0,1\}^n$ be an "indicator vector" for some cut.

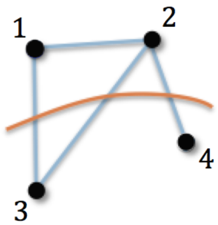


$$\begin{array}{c} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{B}} \end{array} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{x}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96)

- Let $\mathbf{B} \in \mathbb{R}^{\binom{n}{2} \times n}$ be the vertex-edge incidence matrix for a graph G.
- Let $\mathbf{x} \in \{0, 1\}^n$ be an "indicator vector" for some cut.



$\|\mathbf{Bx}\|_2^2 = \text{cut value}$

$$\begin{array}{c} e_{12} \\ e_{13} \\ e_{14} \\ e_{23} \\ e_{24} \\ e_{34} \end{array} \overset{\begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array}}{\underset{\mathbf{B}}{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}} \times \underset{\mathbf{x}}{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96) So, $\|\mathbf{Bx}\|_2^2 =$ cut value.

Goal

Find some $\tilde{\mathbf{B}}$ such that, for all $\mathbf{x} \in \{0,1\}^n$,

$$(1-\varepsilon)\|\mathbf{Bx}\|_2^2 \leq \|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \leq (1+\varepsilon)\|\mathbf{Bx}\|_2^2$$

- $\mathbf{x}^\top \tilde{\mathbf{B}}^\top \tilde{\mathbf{B}}\mathbf{x} \approx \mathbf{x}^\top \mathbf{B}^\top \mathbf{B}\mathbf{x}$.
- $\mathbf{L} = \mathbf{B}^\top \mathbf{B}$ is the *graph Laplacian*.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96) So, $\|\mathbf{B}\mathbf{x}\|_2^2 = $ cut value.

Goal

Find some $\tilde{\mathbf{B}}$ such that, for all $\mathbf{x} \in \{0,1\}^n$,

$$(1-\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2 \leq \|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \leq (1+\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2$$

- $\mathbf{x}^\top \tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} \mathbf{x} \approx \mathbf{x}^\top \mathbf{B}^\top \mathbf{B} \mathbf{x}$.
- $\mathbf{L} = \mathbf{B}^\top \mathbf{B}$ is the *graph Laplacian*.

Graph Sparsification

Cut Sparsification (Benczúr, Karger '96) So, $\|\mathbf{B}\mathbf{x}\|_2^2 = \text{cut value}$.

Goal

Find some $\tilde{\mathbf{B}}$ such that, for all $\mathbf{x} \in \{0,1\}^n$,

$$(1-\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2 \leq \|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \leq (1+\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2$$

- $\mathbf{x}^\top \tilde{\mathbf{B}}^\top \tilde{\mathbf{B}}\mathbf{x} \approx \mathbf{x}^\top \mathbf{B}^\top \mathbf{B}\mathbf{x}$.
- $\mathbf{L} = \mathbf{B}^\top \mathbf{B}$ is the *graph Laplacian*.

Graph Sparsification

Spectral Sparsification (Spielman, Teng '04)

Goal

Find some $\tilde{\mathbf{B}}$ such that, for all $\mathbf{x} \in$ ~~$\{0,1\}^n$~~ $\mathbb{R}^n$,

$$(1-\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2 \le \|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \le (1+\varepsilon)\|\mathbf{B}\mathbf{x}\|_2^2$$

Applications: Anything cut sparsifiers can do, Laplacian system solves, computing effective resistances, spectral clustering, calculating random walk properties, etc.

Graph Sparsification

Spectral Sparsification (Spielman, Teng '04)

Goal

Find some $\tilde{\mathbf{B}}$ such that, for all $\mathbf{x} \in \sout{\{0,1\}^n}$ $\mathbb{R}^n$,

$$(1-\varepsilon)\|\mathbf{Bx}\|_2^2 \leq \|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \leq (1+\varepsilon)\|\mathbf{Bx}\|_2^2$$

Applications: Anything cut sparsifiers can do, Laplacian system solves, computing effective resistances, spectral clustering, calculating random walk properties, etc.

Graph Sparsification

Again, recall that $\|\mathbf{y}\|_2^2 = \mathbf{y}^\top \mathbf{y}$.

All Equivalent:

$\|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \approx_\epsilon \|\mathbf{B}\mathbf{x}\|_2^2$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}} \mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L} \mathbf{x}$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}}^{-1} \mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}^{-1} \mathbf{x}$

Spectral matrix approximation also useful for approximate regression, constructing preconditioners, low rank approximation, RIP/compressed sensing, etc.

Graph Sparsification

Again, recall that $\|\mathbf{y}\|_2^2 = \mathbf{y}^\top \mathbf{y}$.

All Equivalent:

$\|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \approx_\epsilon \|\mathbf{B}\mathbf{x}\|_2^2$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}\mathbf{x}$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}}^{-1}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}^{-1}\mathbf{x}$

Spectral matrix approximation also useful for approximate regression, constructing preconditioners, low rank approximation, RIP/compressed sensing, etc.

Graph Sparsification

Again, recall that $\|\mathbf{y}\|_2^2 = \mathbf{y}^\top \mathbf{y}$.

All Equivalent:

$\|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \approx_\epsilon \|\mathbf{B}\mathbf{x}\|_2^2$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}} \mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L} \mathbf{x}$ $\quad$ $\mathbf{x}^\top \tilde{\mathbf{L}}^{-1} \mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}^{-1} \mathbf{x}$

Spectral matrix approximation also useful for approximate regression, constructing preconditioners, low rank approximation, RIP/compressed sensing, etc.

Graph Sparsification

Again, recall that $\|\mathbf{y}\|_2^2 = \mathbf{y}^\top \mathbf{y}$.

All Equivalent:

$\|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \approx_\epsilon \|\mathbf{B}\mathbf{x}\|_2^2 \qquad \mathbf{x}^\top \tilde{\mathbf{L}}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}\mathbf{x} \qquad \mathbf{x}^\top \tilde{\mathbf{L}}^{-1}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}^{-1}\mathbf{x}$

Spectral matrix approximation also useful for approximate regression, constructing preconditioners, low rank approximation, RIP/compressed sensing, etc.

Graph Sparsification

Again, recall that $\|\mathbf{y}\|_2^2 = \mathbf{y}^\top \mathbf{y}$.

All Equivalent:

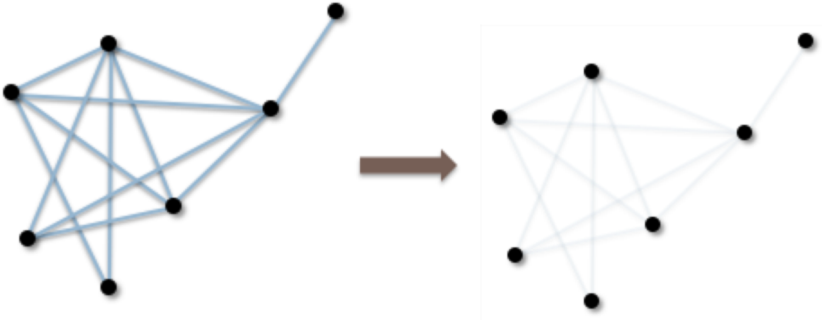
$$\|\tilde{\mathbf{B}}\mathbf{x}\|_2^2 \approx_\epsilon \|\mathbf{B}\mathbf{x}\|_2^2 \qquad \mathbf{x}^\top \tilde{\mathbf{L}}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}\mathbf{x} \qquad \mathbf{x}^\top \tilde{\mathbf{L}}^{-1}\mathbf{x} \approx_\epsilon \mathbf{x}^\top \mathbf{L}^{-1}\mathbf{x}$$

Spectral matrix approximation also useful for approximate regression, constructing preconditioners, low rank approximation, RIP/compressed sensing, etc.

Graph Sparsification

How are sparsifiers constructed?

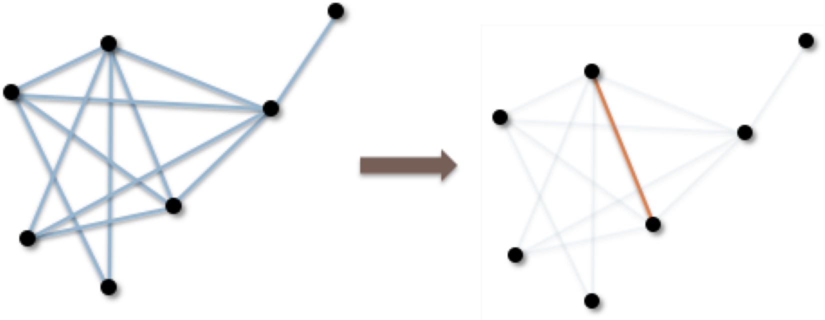
Randomly sample edges (i.e. rows from **B**):



Graph Sparsification

How are sparsifiers constructed?

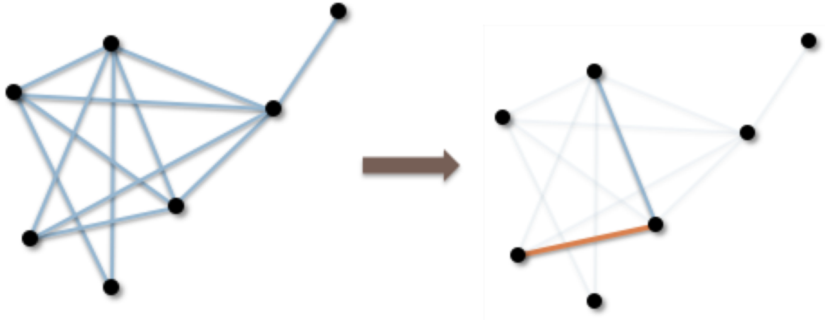
Randomly sample edges (i.e. rows from $\mathbf{B}$):



Graph Sparsification

How are sparsifiers constructed?

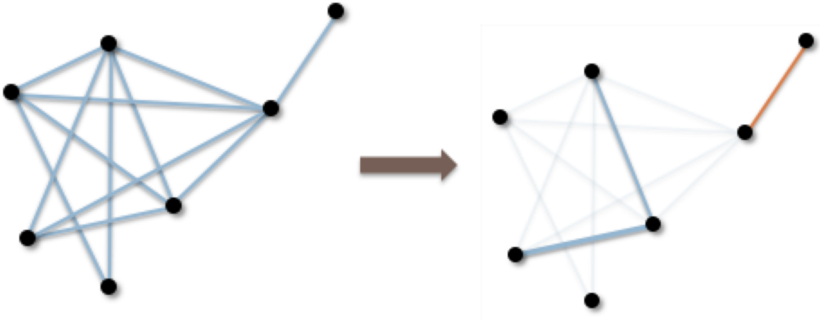
Randomly sample edges (i.e. rows from $\mathbf{B}$):



Graph Sparsification

How are sparsifiers constructed?

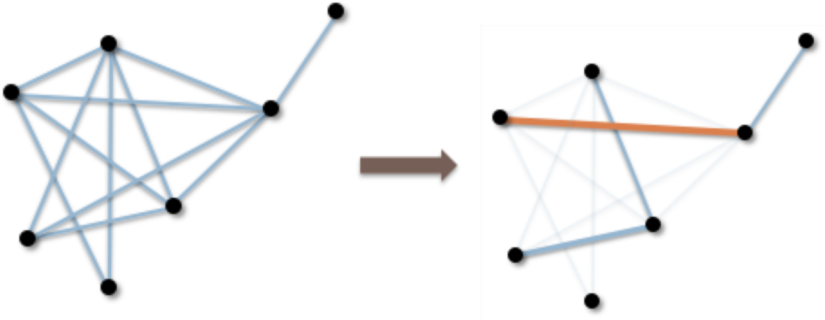
Randomly sample edges (i.e. rows from $\mathbf{B}$):



Graph Sparsification

How are sparsifiers constructed?

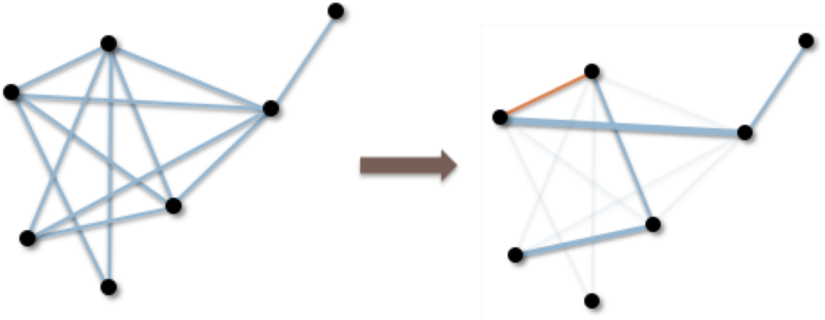
Randomly sample edges (i.e. rows from $\mathbf{B}$):



Graph Sparsification

How are sparsifiers constructed?

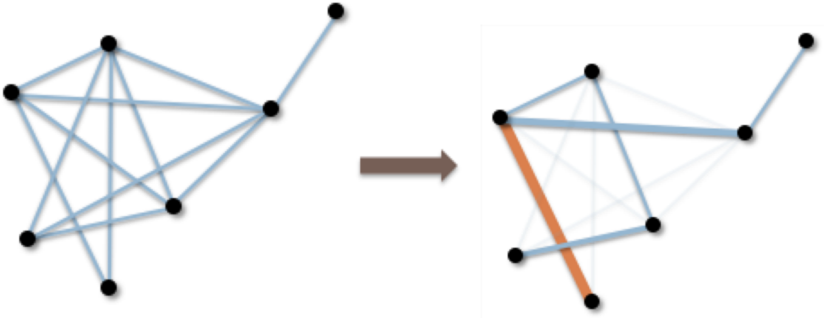
Randomly sample edges (i.e. rows from **B**):



Graph Sparsification

How are sparsifiers constructed?

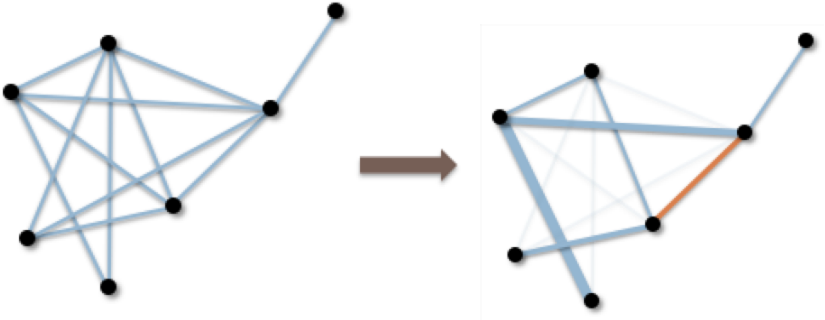
Randomly sample edges (i.e. rows from **B**):



Graph Sparsification

How are sparsifiers constructed?

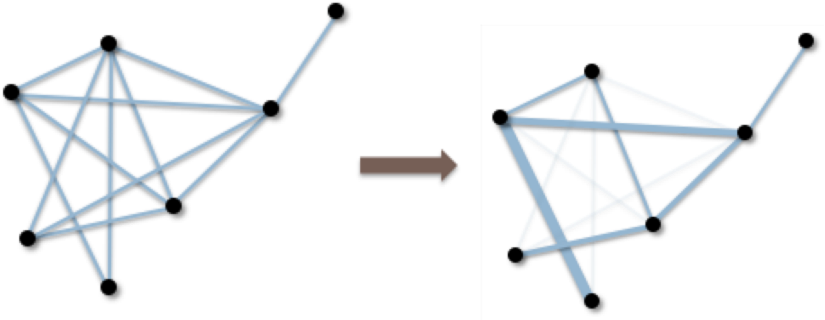
Randomly sample edges (i.e. rows from $\mathbf{B}$):



Graph Sparsification

How are sparsifiers constructed?

Randomly sample edges (i.e. rows from **B**):



Graph Sparsification

How are sparsifiers constructed?

Sampling probabilities:

- Connectivity for cut sparsifiers [Benczúr, Karger '96], [Fung, Hariharan, Harvey, Panigrahi '11].
- **Effective resistances** (i.e statistical leverage scores) for spectral sparsifiers [Spielman, Srivastava '08].

Actually oversample: by (effective resistance) $\times$ $O(\log n)$.
Gives sparsifiers with $O(n \log n)$ edges – reducing from $O(n^2)$.

Graph Sparsification

How are sparsifiers constructed?

Sampling probabilities:

- Connectivity for cut sparsifiers [Benczúr, Karger '96], [Fung, Hariharan, Harvey, Panigrahi '11].
- **Effective resistances** (i.e statistical leverage scores) for spectral sparsifiers [Spielman, Srivastava '08].

Actually oversample: by (effective resistance) $\times$ $O(\log n)$.
Gives sparsifiers with $O(n \log n)$ edges – reducing from $O(n^2)$.

Graph Sparsification

How are sparsifiers constructed?

Sampling probabilities:

- Connectivity for cut sparsifiers [Benczúr, Karger '96], [Fung, Hariharan, Harvey, Panigrahi '11].
- **Effective resistances** (i.e statistical leverage scores) for spectral sparsifiers [Spielman, Srivastava '08].

Actually oversample: by (effective resistance) $\times$ $O(\log n)$.
Gives sparsifiers with $O(n \log n)$ edges – reducing from $O(n^2)$.

Graph Sparsification

How are sparsifiers constructed?

Sampling probabilities:

- Connectivity for cut sparsifiers [Benczúr, Karger '96], [Fung, Hariharan, Harvey, Panigrahi '11].
- **Effective resistances** (i.e statistical leverage scores) for spectral sparsifiers [Spielman, Srivastava '08].

Actually oversample: by (effective resistance) $\times$ $O(\log n)$.
Gives sparsifiers with $O(n \log n)$ edges – reducing from $O(n^2)$.

Graph Sparsification

How are sparsifiers constructed?

Sampling probabilities:

- Connectivity for cut sparsifiers [Benczúr, Karger '96], [Fung, Hariharan, Harvey, Panigrahi '11].
- **Effective resistances** (i.e statistical leverage scores) for spectral sparsifiers [Spielman, Srivastava '08].

Actually oversample: by (effective resistance) $\times$ $O(\log n/\epsilon^2)$.
Gives sparsifiers with $O(n \log n/\epsilon^2)$ edges – reducing from $O(n^2)$.

Motivation

- Makes sense to compress a graph, but what if we cannot afford to store it in the first place?
- Is it possible to “sketch” a graph in small space by maintaining a sparsifier or some other representation?

Overview

Semi-Streaming Model

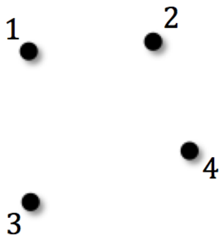
Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.

Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

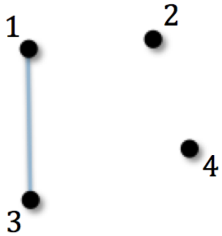
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

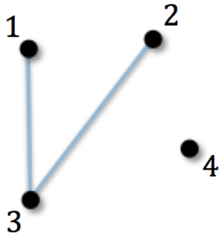
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

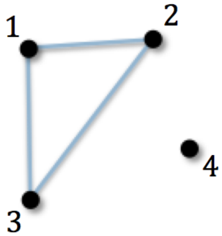
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

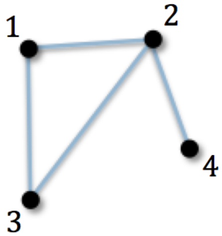
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

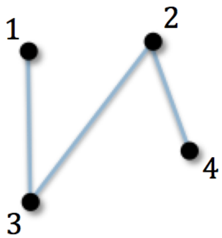
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

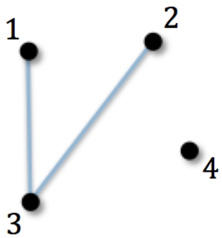
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

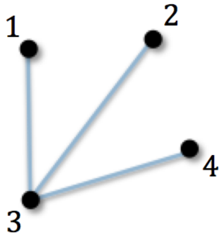
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

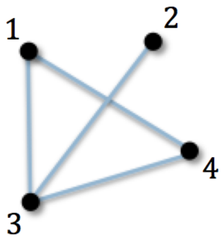
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

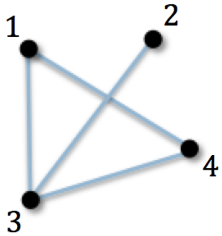
- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Semi-Streaming Model

Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang '05.

- Space allowance $n \log^c(n)$.
- Receive data via edge updates.
- Minimum spanning tree, maximal matching, graph connectivity, etc.



Overview

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Prior Work

- **[Ahn, Guha '09], [Kelner, Levin '11]**: Cut and spectral sparsifiers in *insertion only* streams.
- **[Ahn, Guha, McGregor '12a]**: Introduced linear sketching for graphs. This breakthrough work is the first to handle edge deletions for graph problems. Connectivity, MST, multi-pass sparsifiers.
  - **[Ahn, Guha, McGregor '12b], [Goel, Kapralov, Post '12]**: Extend techniques to get single pass cut sparsifiers.
- **[Ahn, Guha, McGregor '13]**: Dynamic spectral sparsifiers, but $O(n^{5/3})$ space.
- **[Kapralov, Woodruff '14]**: Dynamic spectral sparsifiers, but multi-pass.

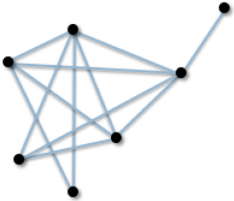
Our result: 1-Pass dynamic spectral sparsifiers in $\tilde{O}(n)$ space.

Overview

1. Graph Sparsification
2. Semi-Streaming Computational Model
3. Prior Work Review
4. Our Algorithm
   - Recover High Effective Resistance Edges
   - Sampling by Effective Resistance
   - Recursive Sparsification [Li, Miller, Peng '12]

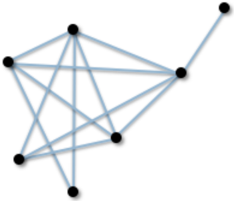
Why is the dynamic case hard?

Graph:

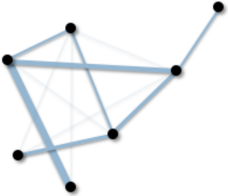


Why is the dynamic case hard?

Graph:

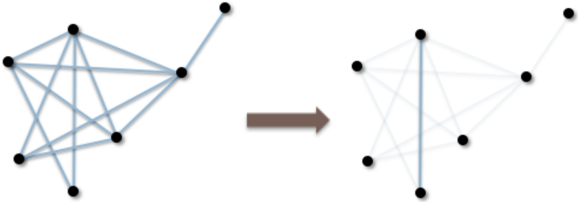


Sketch:

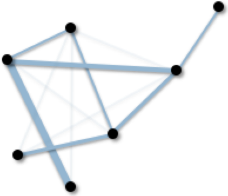


Why is the dynamic case hard?

Graph:

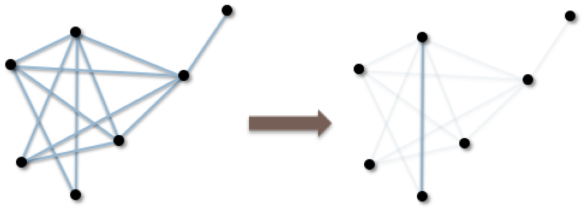


Sketch:

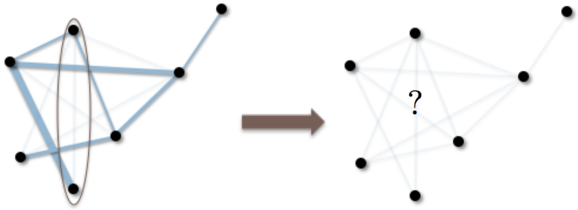


Why is the dynamic case hard?

Graph:



Sketch:



Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

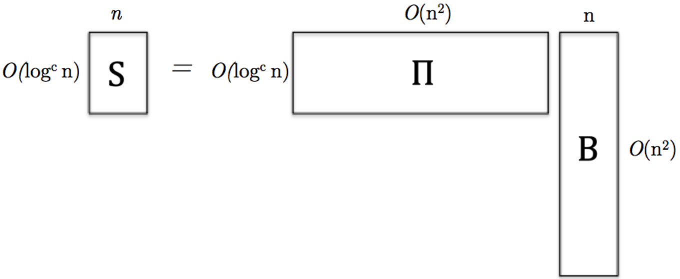
- Linear Sketching!
- Does *not* depend on insertion/deletion order.

Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

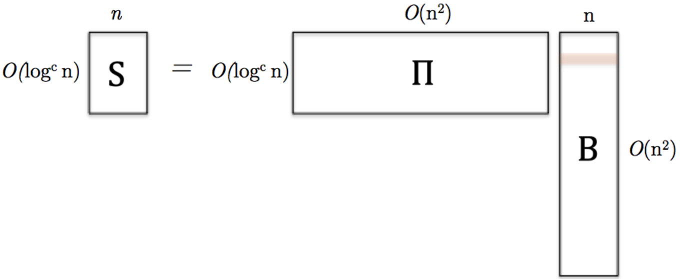


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

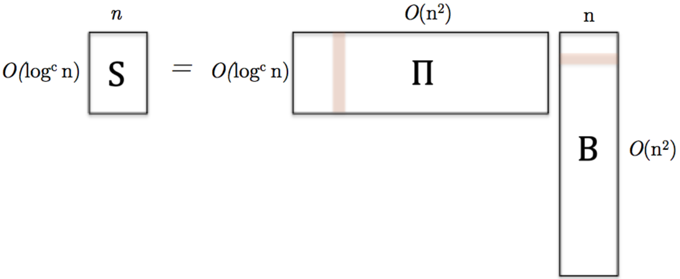


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

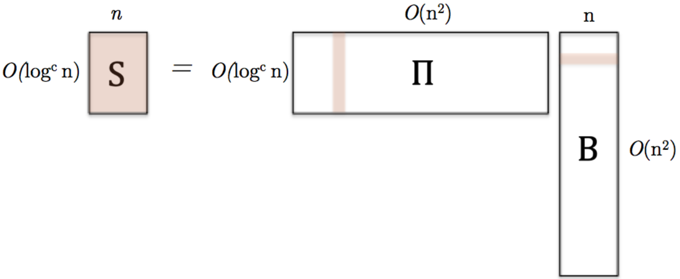


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

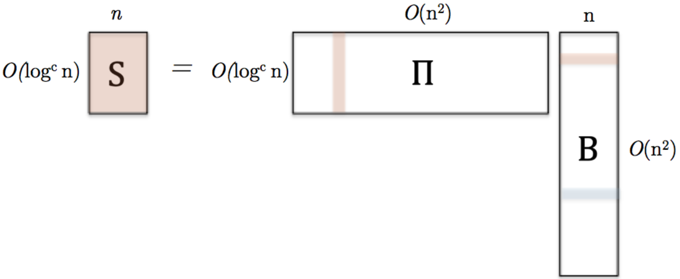


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

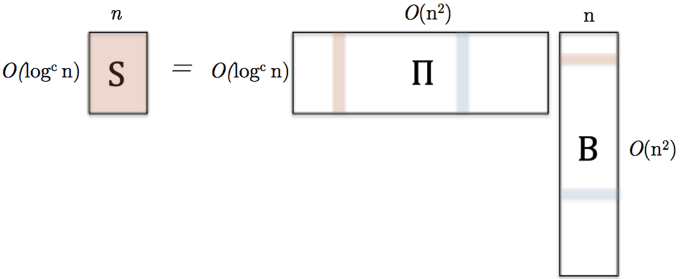


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

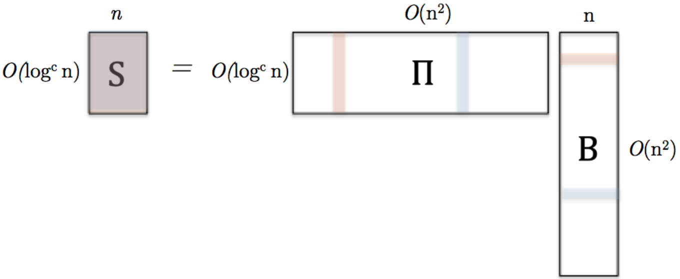


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

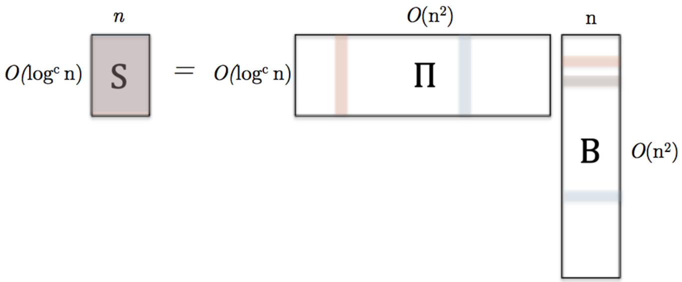


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

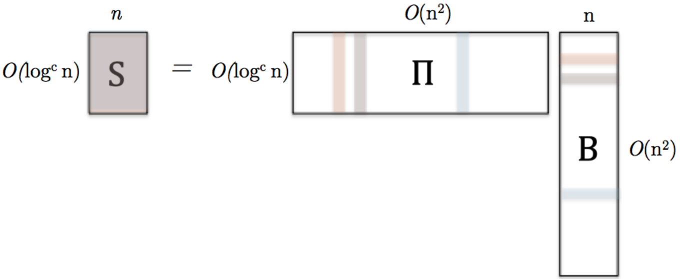


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

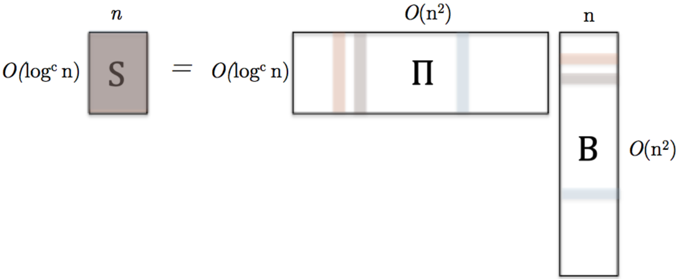


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

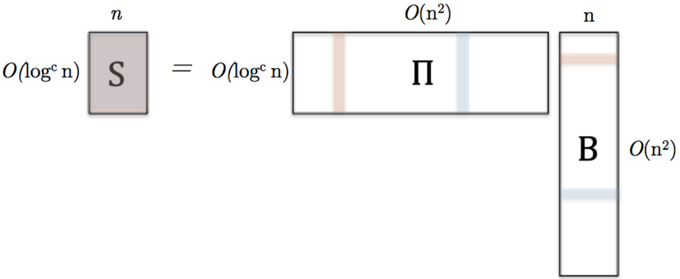
- Linear Sketching!
- Does *not* depend on insertion/deletion order.



Why is the dynamic case hard?

How do we get around this issue?
Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.

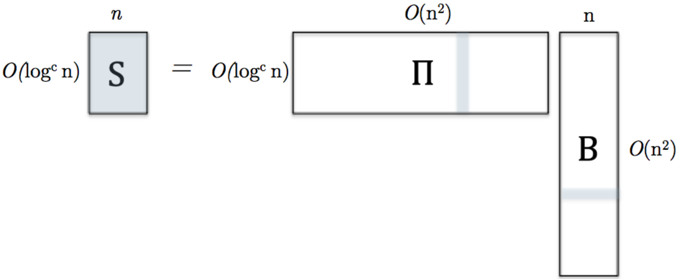


Why is the dynamic case hard?

How do we get around this issue?

Take a cue from standard streaming algorithms:

- Linear Sketching!
- Does *not* depend on insertion/deletion order.



Algorithm Overview

High level strategy:

1. Assume we have a coarse sparsifier – i.e. $(1 \pm \frac{1}{2})$ approximation $\tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} = \tilde{\mathbf{L}}$.
2. Show procedure for recovering high effective resistance edges
3. Use black-box to sample edges by effective resistance

Algorithm Overview

High level strategy:

1. Assume we have a coarse sparsifier – i.e. $(1 \pm \frac{1}{2})$ approximation $\tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} = \tilde{\mathbf{L}}$.
2. Show procedure for recovering high effective resistance edges
3. Use black-box to sample edges by effective resistance

Algorithm Overview

High level strategy:

1. Assume we have a coarse sparsifier – i.e. $(1 \pm \frac{1}{2})$ approximation $\tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} = \tilde{\mathbf{L}}$.
2. Show procedure for recovering high effective resistance edges
3. Use black-box to sample edges by effective resistance

Algorithm Overview

High level strategy:

1. Assume we have a coarse sparsifier – i.e. $(1 \pm \frac{1}{2})$ approximation $\tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} = \tilde{\mathbf{L}}$.
2. Show procedure for recovering high effective resistance edges
3. Use black-box to sample edges by effective resistance

Overview

Linear Sketching

Sketching is an extremely popular tool for compressing *vectors*.

Used for approximating:

- Distinct elements
- Vector norms
- Entropy estimation
- Really any streaming problem...

Linear Sketching

Sketching is an extremely popular tool for compressing *vectors*.

Used for approximating:

- Distinct elements
- Vector norms
- Entropy estimation
- Really any streaming problem...

Linear Sketching

Sketching is an extremely popular tool for compressing *vectors*.

Used for approximating:

- Distinct elements
- Vector norms
- Entropy estimation
- Really any streaming problem...

Linear Sketching

Sketching is an extremely popular tool for compressing *vectors*.

Used for approximating:

- Distinct elements
- Vector norms
- Entropy estimation
- Really any streaming problem...

Linear Sketching

Sketching is an extremely popular tool for compressing *vectors*.

Used for approximating:

- Distinct elements
- Vector norms
- Entropy estimation
- Really any streaming problem...

Graph Sketching

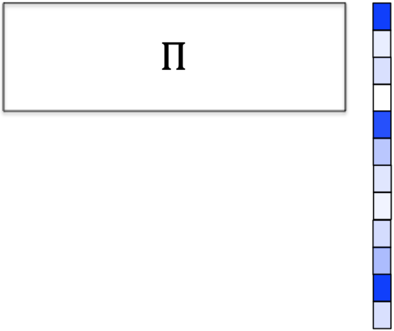
Analyzing Graph Structure via Linear Measurements,
Ahn, Guha, McGregor 2012

- Use a sparse recovery sketch.

Graph Sketching

Analyzing Graph Structure via Linear Measurements,
Ahn, Guha, McGregor 2012

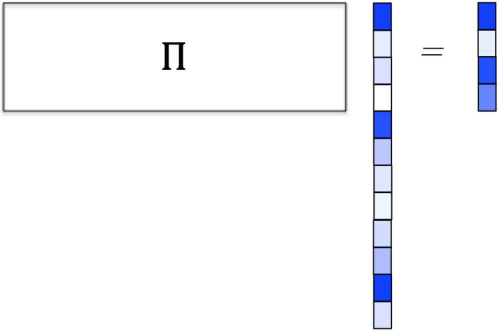
- Use a sparse recovery sketch.



Graph Sketching

Analyzing Graph Structure via Linear Measurements,
Ahn, Guha, McGregor 2012

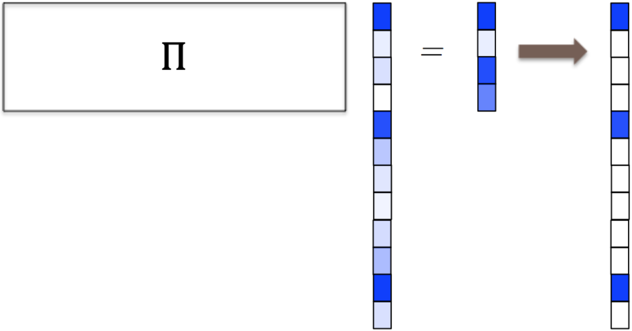
- Use a sparse recovery sketch.



Graph Sketching

Analyzing Graph Structure via Linear Measurements,
Ahn, Guha, McGregor 2012

- Use a sparse recovery sketch.



Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of $\mathbf{B}$.
- Recover *cut information* $\rightarrow$ k-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* $\mathbf{B}$.

Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of $\mathbf{B}$.
- Recover *cut information* $\rightarrow$ k-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* $\mathbf{B}$.

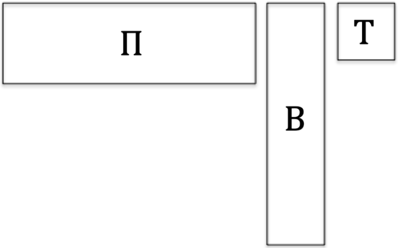
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of **B**.
- Recover *cut information* $\rightarrow$ *k*-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* **B**.



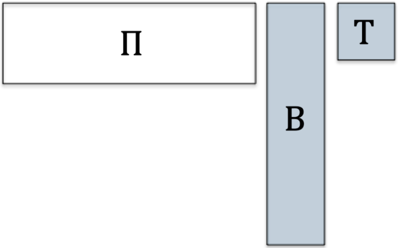
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of **B**.
- Recover *cut information* $\rightarrow$ *k*-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* **B**.



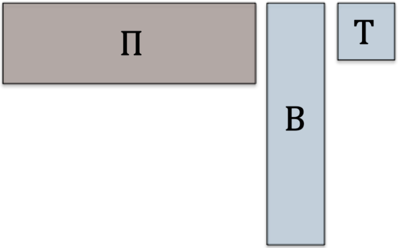
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of $\mathbf{B}$.
- Recover *cut information* $\rightarrow$ k-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* $\mathbf{B}$.



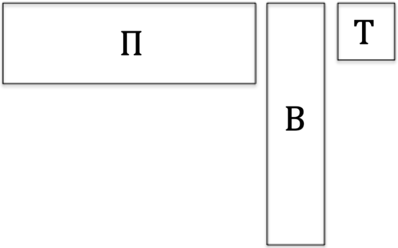
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of $\mathbf{B}$.
- Recover *cut information* $\rightarrow$ k-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* $\mathbf{B}$.



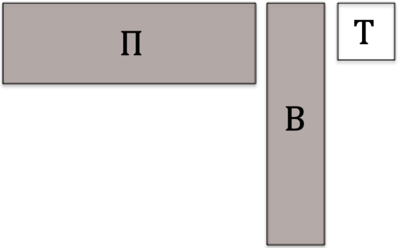
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of **B**.
- Recover *cut information* → *k*-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* **B**.



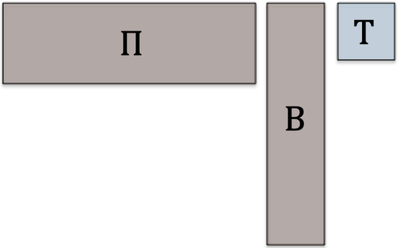
Graph Sketching

Prior Work:

- Apply sparse recovery sketches to the columns of $\mathbf{B}$.
- Recover *cut information* $\rightarrow$ k-connectivity, cut sparsifiers!

Our Approach:

- Apply to *right transformed* $\mathbf{B}$.



Recovering High Resistance Edges

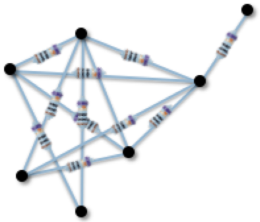
We are still going to sample by effective resistance.

- Treat graph as resistor network, each edge has resistance 1.
- Flow 1 unit of current from node i to j and measure voltage drop between the nodes.

Recovering High Resistance Edges

We are still going to sample by effective resistance.

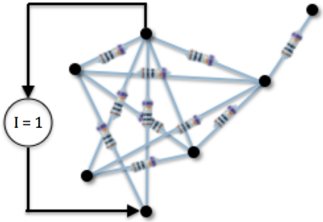
- Treat graph as resistor network, each edge has resistance 1.
- Flow 1 unit of current from node *i* to *j* and measure voltage drop between the nodes.



Recovering High Resistance Edges

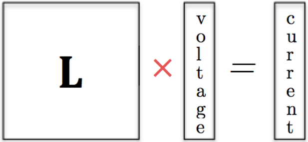
We are still going to sample by effective resistance.

- Treat graph as resistor network, each edge has resistance 1.
- Flow 1 unit of current from node *i* to *j* and measure voltage drop between the nodes.



Recovering High Resistance Edges

Using standard $V = IR$ equations:



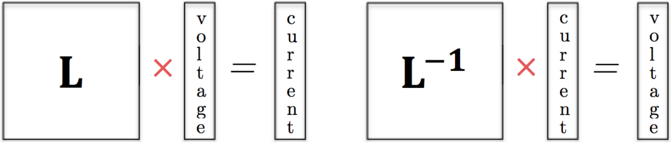
Recovering High Resistance Edges

Using standard $V = IR$ equations:

$$\mathbf{L} \times \text{voltage} = \text{current} \qquad \mathbf{L}^{-1} \times \text{current} = \text{voltage}$$

Recovering High Resistance Edges

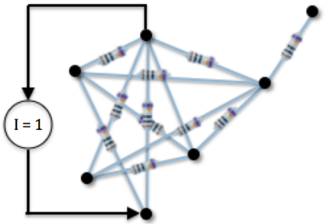
Using standard $V = IR$ equations:



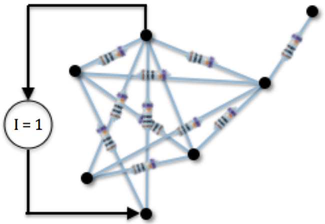
If

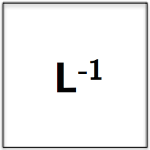
$$\mathbf{x}_e = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}, \text{ } e\text{'s effective resistance is } \tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e.$$

Recovering High Resistance Edges

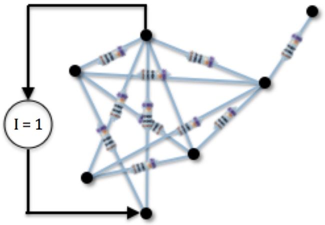


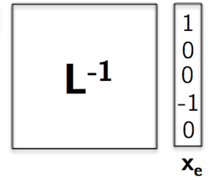
Recovering High Resistance Edges



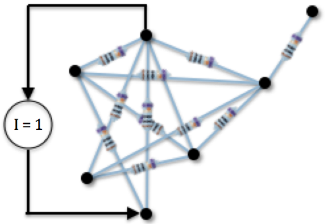


Recovering High Resistance Edges



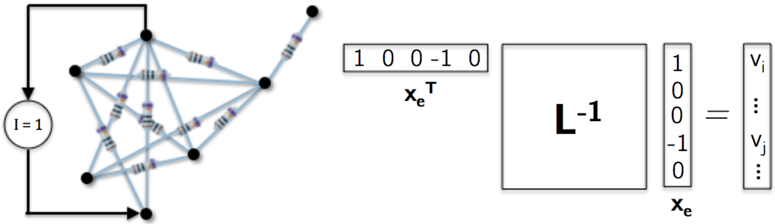


Recovering High Resistance Edges

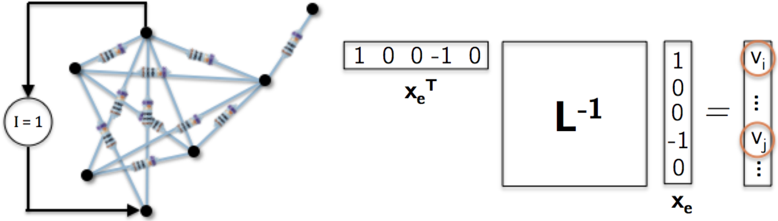


$$L^{-1} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}}_{x_e} = \begin{bmatrix} v_i \\ \vdots \\ v_j \\ \vdots \end{bmatrix}$$

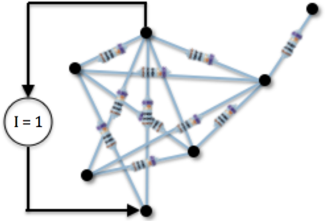
Recovering High Resistance Edges



Recovering High Resistance Edges



Recovering High Resistance Edges



$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & -1 & 0 \end{bmatrix}}_{x_e^T} \; L^{-1} \; \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}}_{x_e} = (v_i - v_j)$$

Recovering High Resistance Edges

Effective resistance of edge e is $\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e$.
Alternatively, τ_e is the e^{th} entry in the vector:

$$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$$

AND

$$\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e = \mathbf{x}_e^\top (\mathbf{L}^{-1})^\top \mathbf{B}^\top \mathbf{B} \mathbf{L}^{-1} \mathbf{x}_e = \|\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e\|_2^2$$

Recovering High Resistance Edges

Effective resistance of edge e is $\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e$.
Alternatively, τ_e is the e^{th} entry in the vector:

$$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$$

AND

$$\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e = \mathbf{x}_e^\top (\mathbf{L}^{-1})^\top \mathbf{B}^\top \mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e = \|\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e\|_2^2$$

Recovering High Resistance Edges

Effective resistance of edge e is $\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e$.
Alternatively, τ_e is the e^{th} entry in the vector:

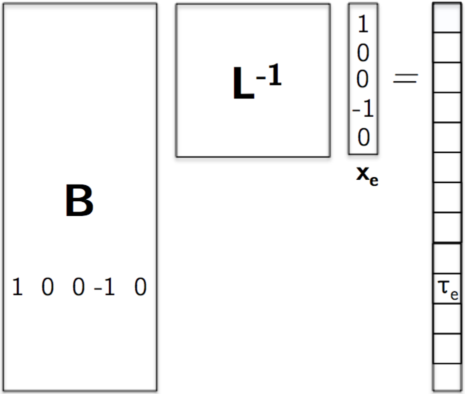
$$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$$

AND

$$\tau_e = \mathbf{x}_e^\top \mathbf{L}^{-1} \mathbf{x}_e = \mathbf{x}_e^\top (\mathbf{L}^{-1})^\top \mathbf{B}^\top \mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e = \|\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e\|_2^2$$

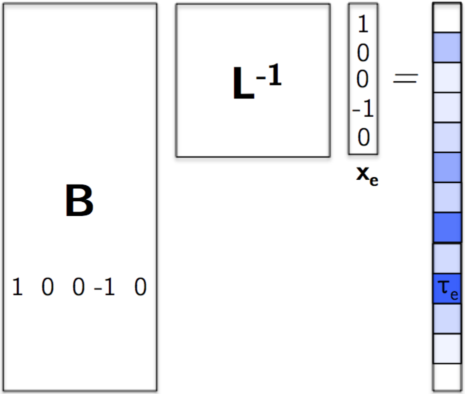
Recovering High Resistance Edges

Effective Resistance: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$



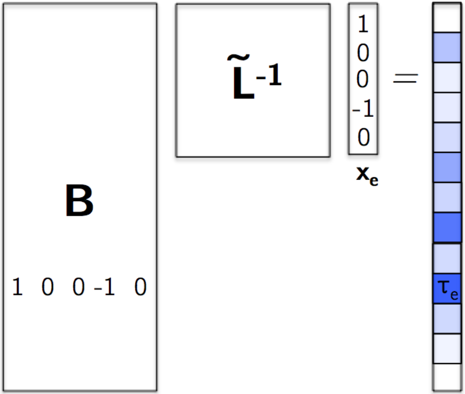
Recovering High Resistance Edges

Effective Resistance: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$



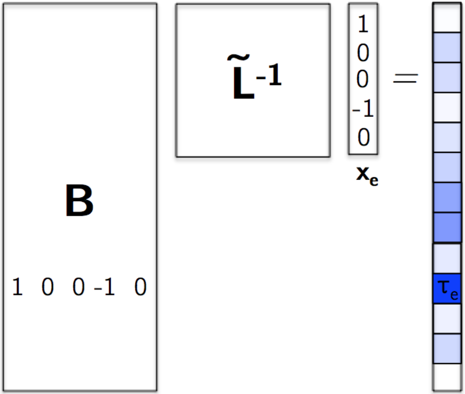
Recovering High Resistance Edges

Effective Resistance: $\mathbf{BL}^{-1}\mathbf{x}_e$



Recovering High Resistance Edges

Effective Resistance: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

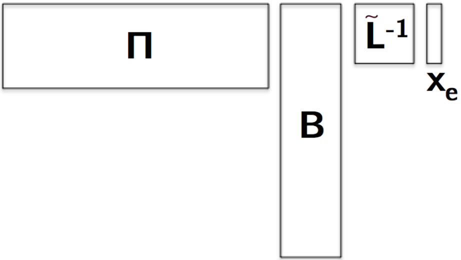


Recovering High Resistance Edges

Full sketching procedure:

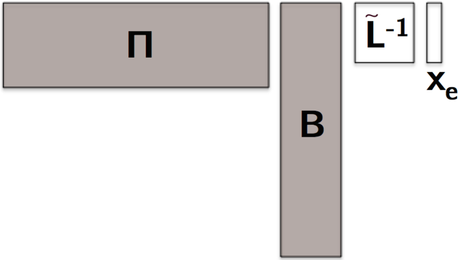
Recovering High Resistance Edges

Full sketching procedure:



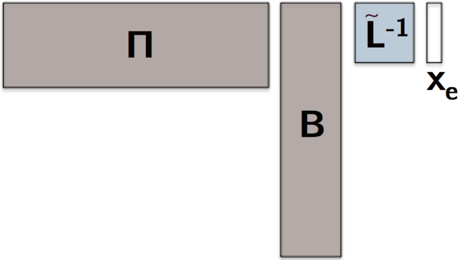
Recovering High Resistance Edges

Full sketching procedure:



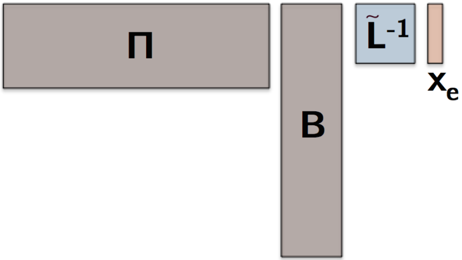
Recovering High Resistance Edges

Full sketching procedure:



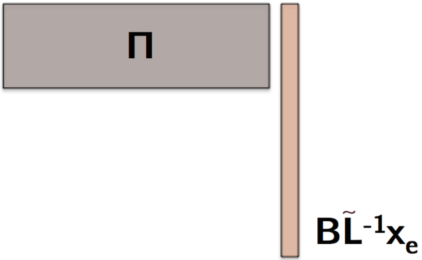
Recovering High Resistance Edges

Full sketching procedure:



Recovering High Resistance Edges

Full sketching procedure:



Recovering High Resistance Edges

Sparse recovery specifics: $\mathbf{BL}^{-1}\mathbf{x}_e$

ℓ_2 **Heavy Hitters [GLPS10]:**

- Sketch *poly*(n) vector in *polylog*(n) space.
- Extract any element who's square is a $O(1/\log n)$ fraction of the vector's squared norm.

Recovering High Resistance Edges

Sparse recovery specifics: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

ℓ_2 **Heavy Hitters [GLPS10]:**

- Sketch *poly*(n) vector in *polylog*(n) space.
- Extract any element who's square is a $O(1/\log n)$ fraction of the vector's squared norm.

Recovering High Resistance Edges

Putting it all together: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

1. Sketch $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}$ in $n \log^c n$ space.
2. Compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}$.
3. For every possible edge e, compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e$
4. Extract heavy hitters from the vector, check if e^{th} entry is one.

$$\frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx \frac{\tau_e^2}{\tau_e} = \tau_e$$

So, as long as $\tau_e > O(1/\log n)$, we will recover the edge!

Recovering High Resistance Edges

Putting it all together:

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

1. Sketch $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}$ in $n \log^c n$ space.
2. Compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}$.
3. For every possible edge e, compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e$
4. Extract heavy hitters from the vector, check if e^{th} entry is one.

$$\frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx \frac{\tau_e^2}{\tau_e} = \tau_e$$

So, as long as $\tau_e > O(1/\log n)$, we will recover the edge!

Recovering High Resistance Edges

Putting it all together: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

1. Sketch $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}$ in $n \log^c n$ space.
2. Compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}$.
3. For every possible edge e, compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e$
4. Extract heavy hitters from the vector, check if e^{th} entry is one.

$$\frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx \frac{\tau_e^2}{\tau_e} = \tau_e$$

So, as long as $\tau_e > O(1/\log n)$, we will recover the edge!

Recovering High Resistance Edges

Putting it all together: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

1. Sketch $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}$ in $n \log^c n$ space.
2. Compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-\mathbf{1}}$.
3. For every possible edge e, compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-\mathbf{1}}\mathbf{x}_e$
4. Extract heavy hitters from the vector, check if e^{th} entry is one.

$$\frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx \frac{\tau_e^2}{\tau_e} = \tau_e$$

So, as long as $\tau_e > O(1/\log n)$, we will recover the edge!

Recovering High Resistance Edges

Putting it all together: $\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

1. Sketch $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}$ in $n \log^c n$ space.
2. Compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-\mathbf{1}}$.
3. For every possible edge e, compute $(\mathbf{\Pi}_{\text{heavy hitters}})\mathbf{B}\tilde{\mathbf{L}}^{-\mathbf{1}}\mathbf{x}_e$
4. Extract heavy hitters from the vector, check if e^{th} entry is one.

$$\frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx \frac{\tau_e^2}{\tau_e} = \tau_e$$

So, as long as $\tau_e > O(1/\log n)$, we will recover the edge!

Overview

Sampling in the Streaming Model

How about edges with lower effective resistance? Sketch:

$\mathbf{BL}^{-1}\mathbf{x}_e$

Sampling in the Streaming Model

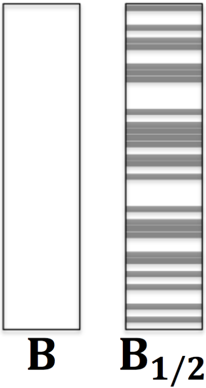
How about edges with lower effective resistance? Sketch:

$\mathbf{B}$

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

Sampling in the Streaming Model

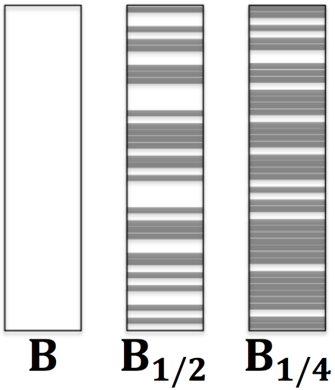
How about edges with lower effective resistance? Sketch:



$\mathbf{BL^{-1}x_e}$

Sampling in the Streaming Model

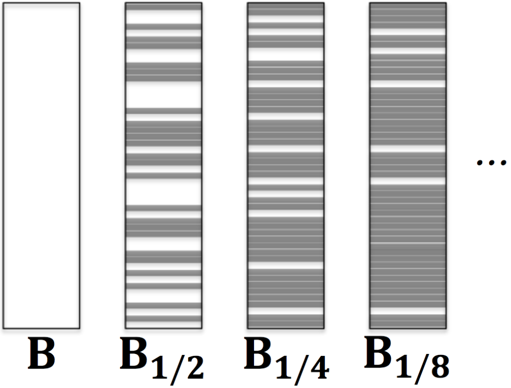
How about edges with lower effective resistance? Sketch:



$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

Sampling in the Streaming Model

How about edges with lower effective resistance? Sketch:



$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

Sampling in the Streaming Model

$\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2 \approx \frac{1}{2} \times \|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2$

HOWEVER, if e makes it through the sampling procedure:

$\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2 = \mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2$

So,

Ratio for heavy-hitters $= \frac{\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx 2 \times \frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2}$

Sampling in the Streaming Model

$$\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2 \approx \frac{1}{2} \times \|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2$$

HOWEVER, if e makes it through the sampling procedure:

$$\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2 = \mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2$$

So,

$$\text{Ratio for heavy-hitters } = \frac{\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx 2 \times \frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2}$$

Sampling in the Streaming Model

$\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2 \approx \frac{1}{2} \times \|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2$

HOWEVER, if e makes it through the sampling procedure:

$\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2 = \mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2$

So,

Ratio for heavy-hitters $= \frac{\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}_{1/2}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2} \approx 2 \times \frac{\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e(e)^2}{\|\mathbf{B}\tilde{\mathbf{L}}^{-1}\mathbf{x}_e\|_2^2}$

Sampling in the Streaming Model

$\mathbf{BL}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2 \log n$ with probability 1/2.
- Third level: $\tau_e > 1/4 \log n$ with probability 1/4.
- Forth level: $\tau_e > 1/8 \log n$ with probability 1/8.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2\log n$ with probability 1/2.
- Third level: $\tau_e > 1/4\log n$ with probability 1/4.
- Forth level: $\tau_e > 1/8\log n$ with probability 1/8.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

$\mathbf{BL}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2 \log n$ with probability $1/2$.
- Third level: $\tau_e > 1/4 \log n$ with probability $1/4$.
- Forth level: $\tau_e > 1/8 \log n$ with probability $1/8$.
- ...

So, we can sample every edge by (effective resistance) $\times\ O(\log n)$.

Sampling in the Streaming Model

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2 \log n$ with probability $1/2$.
- Third level: $\tau_e > 1/4 \log n$ with probability $1/4$.
- Forth level: $\tau_e > 1/8 \log n$ with probability $1/8$.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

$\mathbf{BL}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2\log n$ with probability $1/2$.
- Third level: $\tau_e > 1/4\log n$ with probability $1/4$.
- Forth level: $\tau_e > 1/8\log n$ with probability $1/8$.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2 \log n$ with probability $1/2$.
- Third level: $\tau_e > 1/4 \log n$ with probability $1/4$.
- Forth level: $\tau_e > 1/8 \log n$ with probability $1/8$.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

$\mathbf{B}\mathbf{L}^{-1}\mathbf{x}_e$

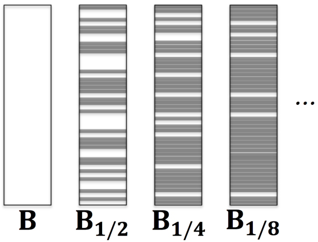
How about edges with lower effective resistance?

- First level: $\tau_e > 1/\log n$ with probability 1.
- Second level: $\tau_e > 1/2 \log n$ with probability $1/2$.
- Third level: $\tau_e > 1/4 \log n$ with probability $1/4$.
- Forth level: $\tau_e > 1/8 \log n$ with probability $1/8$.
- ...

So, we can sample every edge by (effective resistance) $\times$ $O(\log n)$.

Sampling in the Streaming Model

Caveat!

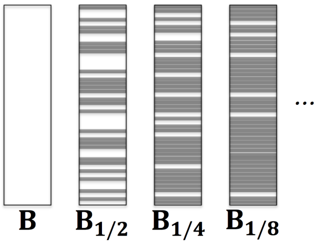


Performing this sampling while processing edges in the stream requires $O(\log n)$ random bits *per edge*. $O(n^2 \log n)$ bits in total.

Fixed using a pseudorandom number generator.

Sampling in the Streaming Model

Caveat!



Performing this sampling while processing edges in the stream requires $O(\log n)$ random bits *per edge*. $O(n^2 \log n)$ bits in total.

Fixed using a pseudorandom number generator.

Overview

Sparsifer Chain

Final Piece [Li, Miller, Peng '12]

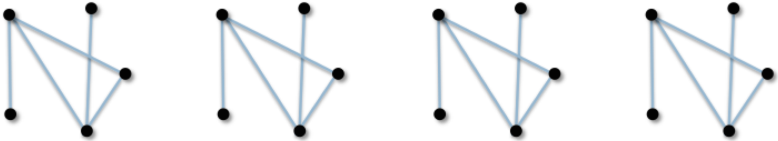
- We need a constant error sparsifier to get a $(1 \pm \epsilon)$ sparsifier.

First graph can be sparsified by constructing an expander.

Sparsifer Chain

Final Piece [Li, Miller, Peng '12]

- We need a constant error sparsifier to get a $(1 \pm \epsilon)$ sparsifier.

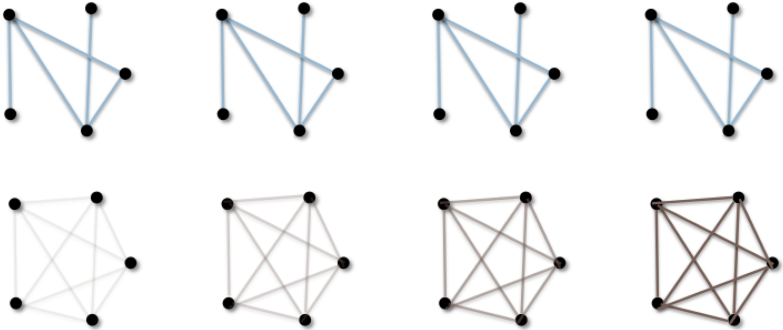


First graph can be sparsified by constructing an expander.

Sparsifer Chain

Final Piece [Li, Miller, Peng '12]

- We need a constant error sparsifier to get a $(1 \pm \epsilon)$ sparsifier.

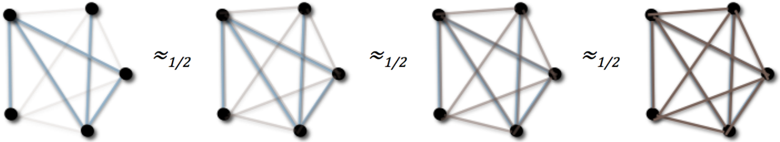


First graph can be sparsified by constructing an expander.

Sparsifer Chain

Final Piece [Li, Miller, Peng '12]

- We need a constant error sparsifier to get a $(1 \pm \epsilon)$ sparsifier.

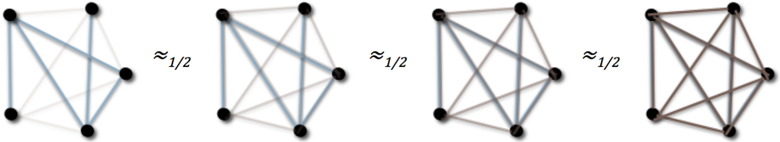


First graph can be sparsified by constructing an expander.

Sparsifer Chain

Final Piece [Li, Miller, Peng '12]

- We need a constant error sparsifier to get a $(1 \pm \epsilon)$ sparsifier.



First graph can be sparsified by constructing an expander.

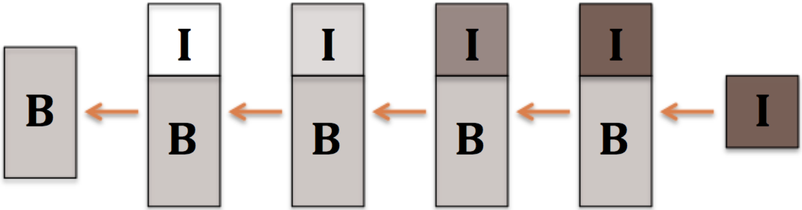
Sparsifer Chain

Actual Implementation:
We add an identity matrix to **B** instead of complete graph edges.

No need for an expander – the identity is already sparse!

Sparsifer Chain

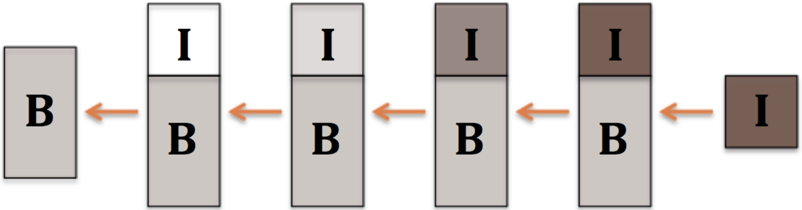
Actual Implementation:
We add an identity matrix to **B** instead of complete graph edges.



No need for an expander – the identity is already sparse!

Sparsifer Chain

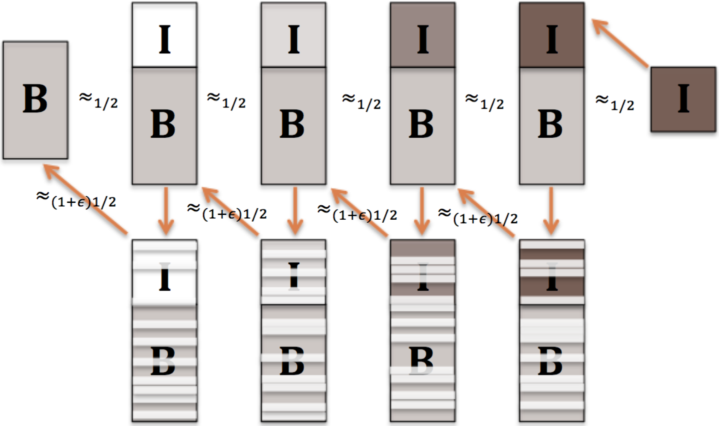
Actual Implementation:
We add an identity matrix to **B** instead of complete graph edges.



No need for an expander – the identity is already sparse!

Sparsifer Chain

Full Procedure:



Sparsifier Chain

Number of levels depends on log condition number of $\mathbf{B}$, which is bounded for an unweighted graph.

Works for any matrix!

- To work for a general matrix $\mathbf{B}$ and general quadratic form $\mathbf{B}^\top\mathbf{B}$ we need:
  - A row dictionary to test every possible entry.
  - A condition number bound.
- Generically, storing a compression of $\mathbf{B}^\top\mathbf{B}$ takes $\Omega(n^2)$ space. Avoid lower bound simply when the row dictionary is $\mathrm{poly}(n)$ size.

Sparsifier Chain

Number of levels depends on log condition number of $\mathbf{B}$, which is bounded for an unweighted graph.

Works for any matrix!

- To work for a general matrix $\mathbf{B}$ and general quadratic form $\mathbf{B}^\top \mathbf{B}$ we need:
  - A row dictionary to test every possible entry.
  - A condition number bound.
- Generically, storing a compression of $\mathbf{B}^\top \mathbf{B}$ takes $\Omega(n^2)$ space. Avoid lower bound simply when the row dictionary is $\mathrm{poly}(n)$ size.

Sparsifier Chain

Number of levels depends on log condition number of $\mathbf{B}$, which is bounded for an unweighted graph.

Works for any matrix!

- To work for a general matrix $\mathbf{B}$ and general quadratic form $\mathbf{B}^\top\mathbf{B}$ we need:
  - A row dictionary to test every possible entry.
  - A condition number bound.
- Generically, storing a compression of $\mathbf{B}^\top\mathbf{B}$ takes $\Omega(n^2)$ space. Avoid lower bound simply when the row dictionary is $\mathrm{poly}(n)$ size.

Sparsifier Chain

Number of levels depends on log condition number of $\mathbf{B}$, which is bounded for an unweighted graph.

Works for any matrix!

- To work for a general matrix $\mathbf{B}$ and general quadratic form $\mathbf{B}^\top \mathbf{B}$ we need:
  - A row dictionary to test every possible entry.
  - A condition number bound.
- Generically, storing a compression of $\mathbf{B}^\top \mathbf{B}$ takes $\Omega(n^2)$ space. Avoid lower bound simply when the row dictionary is $\mathrm{poly}(n)$ size.

Sparsifier Chain

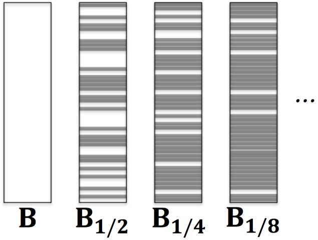
Number of levels depends on log condition number of $\mathbf{B}$, which is bounded for an unweighted graph.

Works for any matrix!

- To work for a general matrix $\mathbf{B}$ and general quadratic form $\mathbf{B}^\top\mathbf{B}$ we need:
  - A row dictionary to test every possible entry.
  - A condition number bound.
- Generically, storing a compression of $\mathbf{B}^\top\mathbf{B}$ takes $\Omega(n^2)$ space. Avoid lower bound simply when the row dictionary is $\mathrm{poly}(n)$ size.

Using a Pseudorandom Number Generator

Recall



Requires $O(n^2 \log n)$ bits in total. We need to store these bits *persistently*.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]

Theorem

Any algorithm running in S space and using R random bits can be simulated using a PRG that uses a seed of $O(S \log R)$ truly random bits.

1. The probability of any outcome changes by at most $2^{-O(S)}$.
2. Each random bit can be generated in $S \log R$ time.

We have $S = O(n \log^c n)$ and $R = O(n^2 \log n)$, so $S \log R$ is just $O(n \log^c n)$ truly random bits for our seed.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]

Theorem

Any algorithm running in S space and using R random bits can be simulated using a PRG that uses a seed of $O(S \log R)$ truly random bits.

1. The probability of any outcome changes by at most $2^{-O(S)}$.
2. Each random bit can be generated in $S \log R$ time.

We have $S = O(n \log^c n)$ and $R = O(n^2 \log n)$, so $S \log R$ is just $O(n \log^c n)$ truly random bits for our seed.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]

But out algorithm doesn't run in S space as described!

Solution: [Indyk '00] Our algorithm can run in $O(n \log^c n)$ if our edges come in order $\rightarrow$ we can throw away hash bits as we go.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]

But out algorithm doesn't run in S space as described!

Solution: [Indyk '00] Our algorithm can run in $O(n \log^c n)$ if our edges come in order $\rightarrow$ we can throw away hash bits as we go.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]

But out algorithm doesn't run in S space as described!

Solution: [Indyk '00] Our algorithm can run in $O(n \log^c n)$ if our edges come in order $\rightarrow$ we can throw away hash bits as we go.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]
So, we can apply the PRG to our algorithm assuming ordered insertions/deletions.

But, since the algorithm is linear, the order in which edges are received does not matter. Thus, the algorithm works for any edge stream.

Unfortunately, every time we need a random has bit, we require $S \log R = O(n \log^c n)$ computation $\rightarrow$ slow update time.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]
So, we can apply the PRG to our algorithm assuming ordered insertions/deletions.

But, since the algorithm is linear, the order in which edges are received does not matter. Thus, the algorithm works for any edge stream.

Unfortunately, every time we need a random has bit, we require $S \log R = O(n \log^c n)$ computation $\rightarrow$ slow update time.

Using a Pseudorandom Number Generator

Nisan's PRG [Nisan '92]
So, we can apply the PRG to our algorithm assuming ordered insertions/deletions.

But, since the algorithm is linear, the order in which edges are received does not matter. Thus, the algorithm works for any edge stream.

Unfortunately, every time we need a random has bit, we require $S \log R = O(n \log^c n)$ computation $\rightarrow$ slow update time.

Conclusion

Thank you!