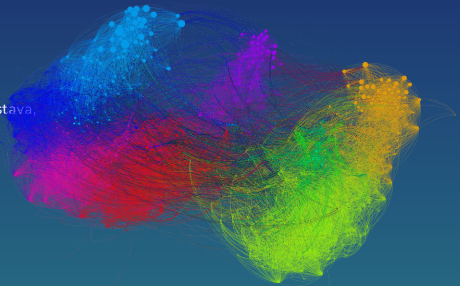
Graphs in Machine Learning

Michal Valko

Inria Lille - Nord Europe, France

TA: Pierre Perrault

Partially based on material by: Daniele Calandriello, Nikhil Srivastava, Yiannis Koutis, Joshua Batson, Daniel Spielman



Last Lecture

- Examples of applications of online SSL
- Analysis of online SSL
- SSL Learnability
- When does graph-based SSL provably help?
- Scaling harmonic functions to millions of samples

This Lecture

- Large-scale graph construction and processing (in class)
- Scalable algorithms:
  - Graph sparsification (presented in class)
  - Online face recognizer (to code in Matlab)
  - Iterative label propagation (to code in Matlab)

This Lecture/Lab Session

- AR: record a video with faces

This Lecture/Lab Session

- AR: record a video with faces
- Short written report

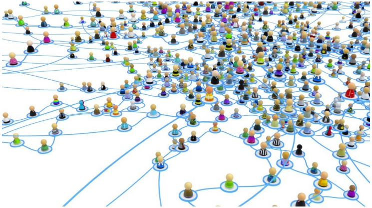
This Lecture/Lab Session

- AR: record a video with faces
- Short written report
- Questions to piazza

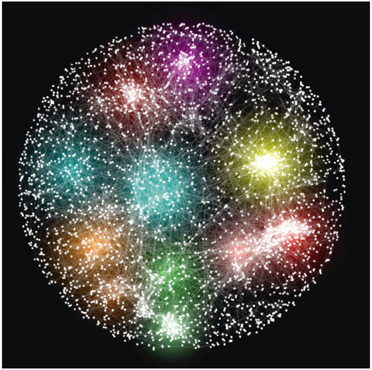
This Lecture/Lab Session

- AR: record a video with faces
- Short written report
- Questions to piazza
- *Deadline:* 11. 12. 2017

Large scale Machine Learning on Graphs

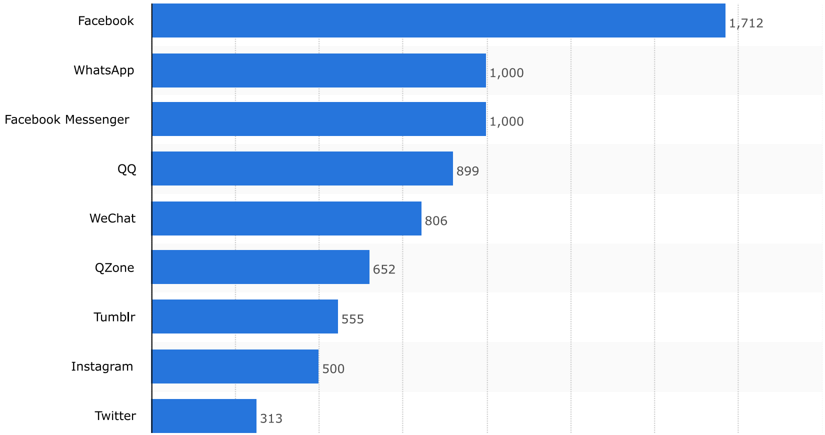


http://blog.carsten-eickhoff.com



Botstein et al.

Are we large yet?



"One **trillion** edges: graph processing at Facebook-scale."
Ching et al., VLDB 2015

Computational bottlenecks

In theory:

Space

$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store

Time

$\mathcal{O}(n^2)$ to construct

$\mathcal{O}(n^3)$ to run algorithms

Computational bottlenecks

In theory:

Space

$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store

Time

$\mathcal{O}(n^2)$ to construct
$\mathcal{O}(n^3)$ to run algorithms

In practice:

- 2012 Common Crawl Corpus:
  3.5 Billion pages (45 GB)
  128 Billion edges (331 GB)

Computational bottlenecks

In theory:

Space	**Time**
$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store	$\mathcal{O}(n^2)$ to construct $\mathcal{O}(n^3)$ to run algorithms

In practice:

- 2012 Common Crawl Corpus:

  3.5 Billion pages (45 GB)
  128 Billion edges (331 GB)
- Pagerank on Facebook Graph:

  3 minutes per iteration, hundreds of iterations, tens of hours on 200 machines, run once per day

Two phases

1. Preprocessing:

Two phases

1 Preprocessing:

From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function

Two phases

1 Preprocessing:

From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function

Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian

Two phases

1. Preprocessing:

   From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph **G** using a similarity function
   Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian
   Load it on the machine: On a single machine if possible, if not find smart way to distribute it

Two phases

1 Preprocessing:

From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function

Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian

Load it on the machine: On a single machine if possible, if not find smart way to distribute it

2 Run your algorithm on the graph

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Fundamental limit: just looking at all similarities already too slow.

Large scale graph construction

Main bottleneck: **time**

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Fundamental limit: just looking at all similarities already too slow.

Can we find close neighbours without checking all distances?

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

Distance Approximation

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

More general problem: learning good codeword representation

Storing graph in memory

Main bottleneck: **space**.

As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.

Storing graph in memory

Main bottleneck: **space**.
As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.
- For standard cloud providers, the largest compute-optimized instances has 36 cores, but only 60 GB of memory.

Storing graph in memory

Main bottleneck: **space**.
As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.
- For standard cloud providers, the largest compute-optimized instances has 36 cores, but only 60 GB of memory.
- We can store $60 * 1024^3/(3 * 8) \sim 2.6 \times 10^9$ (2.6 billion) edges in a single machine memory.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  - ↳ Losing large scale relationship, losing regularization

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  ↳ Your algorithm does not know that.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  - ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  - ↳ Your algorithm does not know that.
    What if it needs nonlocal data? Iterative algorithms?

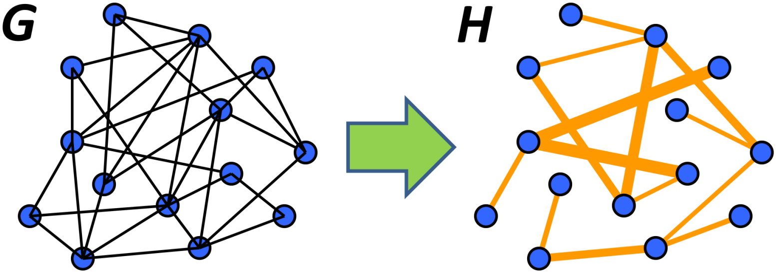
Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  - ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  - ↳ Your algorithm does not know that.
    What if it needs nonlocal data? Iterative algorithms?
    More on this later

Graph Sparsification

Goal: Get graph G and find sparse H



Graph Sparsification: What is sparse?

What does **sparse** graph mean?

- average degree < 10 is pretty sparse

Graph Sparsification: What is sparse?

What does **sparse** graph mean?

- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok

Graph Sparsification: What is sparse?

What does **sparse** graph mean?

- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok
- in general: average degree $<$ polylog n

Graph Sparsification: What is sparse?

What does **sparse** graph mean?

- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok
- in general: average degree $<$ polylog n

Graph Sparsification: What is sparse?

What does sparse graph mean?

- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok
- in general: average degree $<$ polylog n

Are all edges important?

Graph Sparsification: What is sparse?

What does **sparse** graph mean?

- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok
- in general: average degree $<$ polylog n

Are all edges important?

in a tree — sure, in a dense graph perhaps not

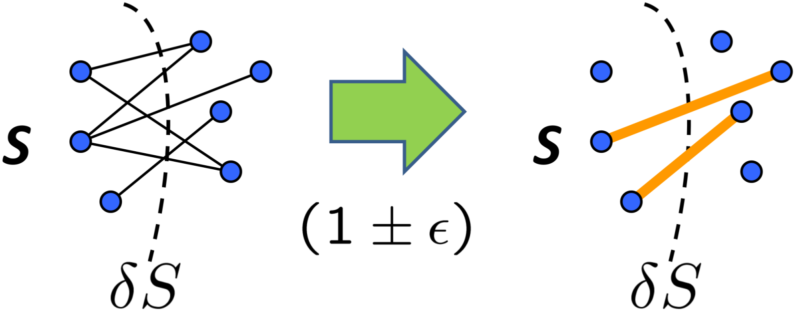
Graph Sparsification: What is **good** sparse?

Graph Sparsification: What is **good** sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

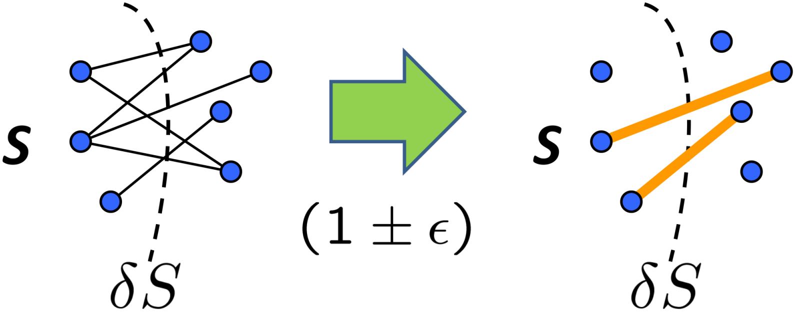
Graph Sparsification: What is good sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!



Graph Sparsification: What is good sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!



H approximates G well iff $\forall S \subset V$, sum of edges on δS remains

δS = edges leaving S

https://math.berkeley.edu/~nikhil/

Graph Sparsification: What is **good** sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care?

Graph Sparsification: What is good sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Graph Sparsification: What is **good** sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Recall what is a cut: $\text{cut}_G(S) =$

Graph Sparsification: What is **good** sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Recall what is a cut: $\text{cut}_G(S) = \sum_{i \in S, j \in \overline{S}} w_{i,j}$

Graph Sparsification: What is **good** sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Recall what is a cut: $\text{cut}_G(S) = \sum_{i \in S, j \in \overline{S}} w_{i,j}$

Define G and H are $(1 \pm \varepsilon)$-**cut similar** when $\forall S$

$$(1-\varepsilon)\text{cut}_H(S) \leq \text{cut}_G(S) \leq (1+\varepsilon)\text{cut}_H(S)$$

Graph Sparsification: What is good sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Recall what is a cut: $\text{cut}_G(S) = \sum_{i \in S, j \in \overline{S}} w_{i,j}$

Define G and H are $(1 \pm \varepsilon)$-**cut similar** when $\forall S$

$$(1-\varepsilon)\text{cut}_H(S) \leq \text{cut}_G(S) \leq (1+\varepsilon)\text{cut}_H(S)$$

Is this always possible?

Graph Sparsification: What is good sparse?

Good sparse by Benczúr and Karger (1996) = **cut preserving**!

Why did they care? faster mincut/maxflow

Recall what is a cut: $\text{cut}_G(S) = \sum_{i \in S, j \in \overline{S}} w_{i,j}$

Define G and H are $(1 \pm \varepsilon)$-**cut similar** when $\forall S$

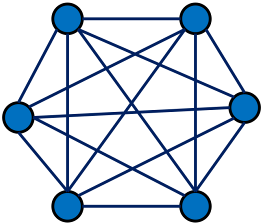
$$(1-\varepsilon)\text{cut}_H(S) \leq \text{cut}_G(S) \leq (1+\varepsilon)\text{cut}_H(S)$$

Is this always possible? Benczúr and Karger (1996): Yes!

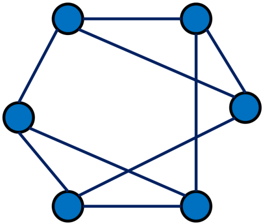
$\forall \varepsilon \; \exists \; (1+\varepsilon)$-cut similar $\widetilde{G}$ with $\mathcal{O}(n \log n/\varepsilon^2)$ edges s.t. $E_H \subseteq E$ and computable in $\mathcal{O}(m \log^3 n + m \log n/\varepsilon^2)$ time n nodes, m edges

Graph Sparsification: What is **good** sparse?

$G = K_n$

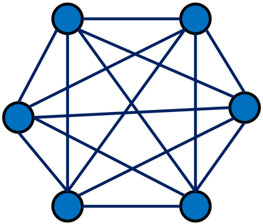


$H = d$-**regular** (random)

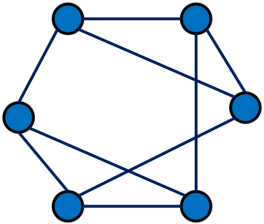


Graph Sparsification: What is **good** sparse?

$G = K_n$



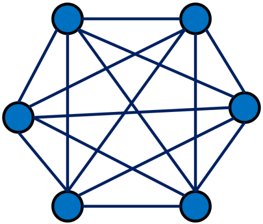
$H = d$-**regular** (random)



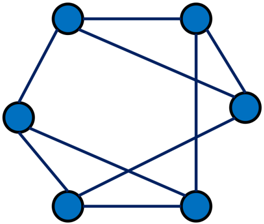
How many edges?

Graph Sparsification: What is good sparse?

$G = K_n$



$H = d$-**regular** (random)

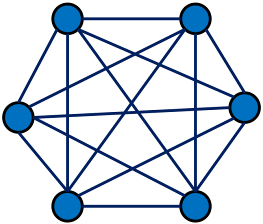


How many edges?

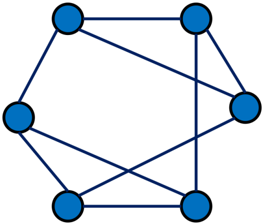
$$|E_G| = \mathcal{O}(n^2)$$

Graph Sparsification: What is good sparse?

$G = K_n$



$H =$ d-**regular** (random)



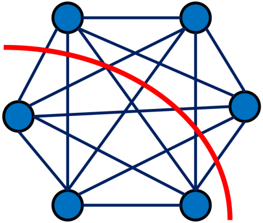
How many edges?

$|E_G| = \mathcal{O}(n^2)$

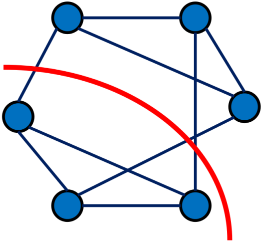
$|E_H| = \mathcal{O}(dn)$

Graph Sparsification: What is **good** sparse?

$G = K_n$

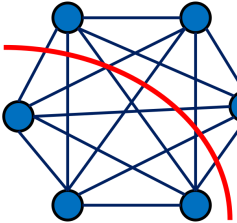


$H = d$-**regular** (random)

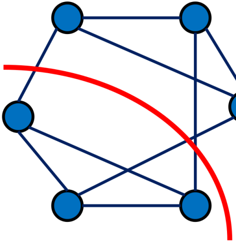


Graph Sparsification: What is **good** sparse?

$G = K_n$



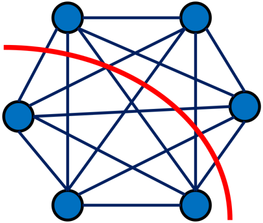
$H = d$-**regular** (random)



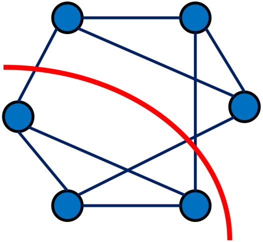
What are the cut weights for any S?

Graph Sparsification: What is **good** sparse?

$G = K_n$



$H = d$-**regular** (random)

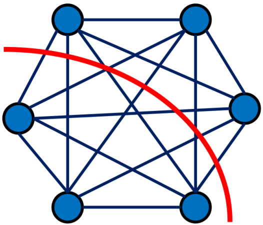


What are the cut weights for any S?

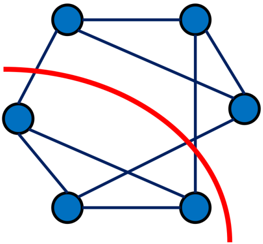
$$w_G(\delta S) = |S| \cdot |\overline{S}|$$

Graph Sparsification: What is **good** sparse?

$G = K_n$



$H = d$-**regular** (random)

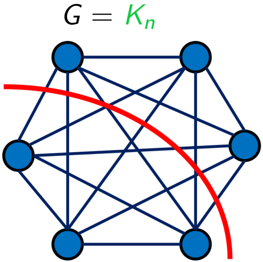


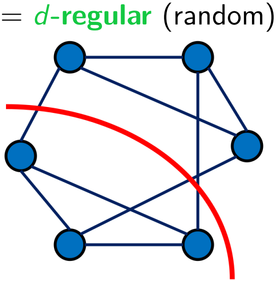
What are the cut weights for any S?

$$w_G(\delta S) = |S| \cdot |\overline{S}| \qquad w_H(\delta S) \approx \frac{d}{n} \cdot |S| \cdot |\overline{S}|$$

$$\forall S \subset V : \frac{w_G(\delta S)}{w_H(\delta S)} \approx \frac{n}{d}$$

Graph Sparsification: What is **good** sparse?





What are the cut weights for any S?

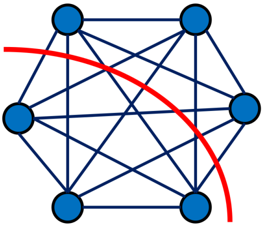
$$w_G(\delta S) = |S| \cdot |\overline{S}| \qquad w_H(\delta S) \approx \frac{d}{n} \cdot |S| \cdot |\overline{S}|$$

$$\forall S \subset V : \frac{w_G(\delta S)}{w_H(\delta S)} \approx \frac{n}{d}$$

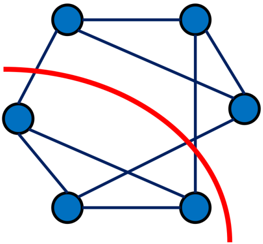
Could be large :(

Graph Sparsification: What is good sparse?

$G = K_n$



$H =$ d-**regular** (random)



What are the cut weights for any S?

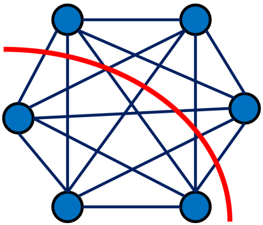
$w_G(\delta S) = |S| \cdot |\overline{S}|$ $\quad$ $w_H(\delta S) \approx \frac{d}{n} \cdot |S| \cdot |\overline{S}|$

$$\forall S \subset V : \frac{w_G(\delta S)}{w_H(\delta S)} \approx \frac{n}{d}$$

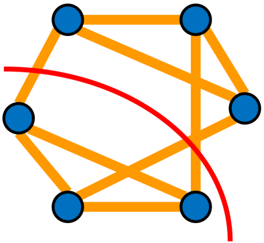
Could be large :(What to do?

Graph Sparsification: What is **good** sparse?

$G = K_n$

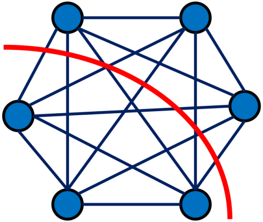


$H = d$-**regular** (random)

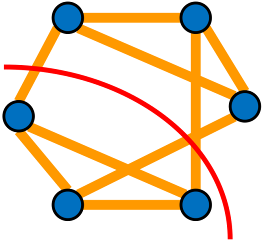


Graph Sparsification: What is good sparse?

$G = K_n$



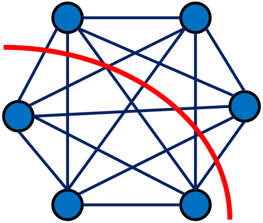
$H = d$-**regular** (random)



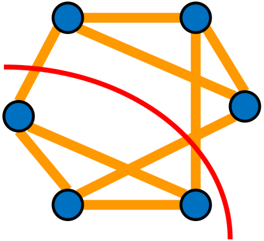
What are the cut weights for any S?

Graph Sparsification: What is good sparse?

$G = K_n$



$H =$ d-**regular** (random)

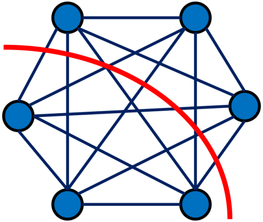


What are the cut weights for any S?

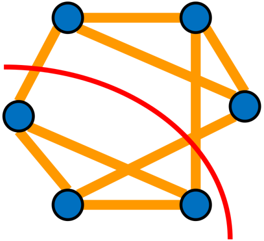
$$w_G(\delta S) = |S| \cdot |\overline{S}|$$

Graph Sparsification: What is good sparse?

$G = K_n$



$H =$ d-**regular** (random)



What are the cut weights for any S?

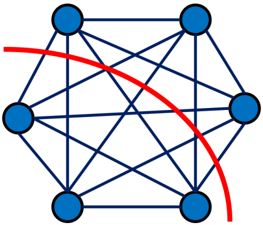
$w_G(\delta S) = |S| \cdot |\overline{S}|$

$w_H(\delta S) \approx \frac{d}{n} \cdot \frac{n}{d} \cdot |S| \cdot |\overline{S}|$

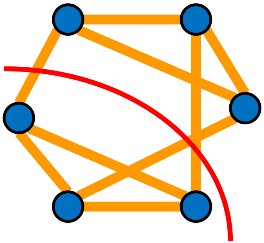
$$\forall S \subset V : \frac{w_G(\delta S)}{w_H(\delta S)} \approx 1$$

Graph Sparsification: What is good sparse?

$G = K_n$



$H = d$-**regular** (random)



What are the cut weights for any S?

$$w_G(\delta S) = |S| \cdot |\overline{S}| \qquad w_H(\delta S) \approx \frac{d}{n} \cdot \frac{n}{d} \cdot |S| \cdot |\overline{S}|$$

$$\forall S \subset V : \frac{w_G(\delta S)}{w_H(\delta S)} \approx 1$$

Benczúr & Karger: Can find such H quickly for any G!

Graph Sparsification: What is **good** sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} =$

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \text{cut}_G(S)$

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \mathrm{cut}_G(S)$

$$(1-\varepsilon)\mathrm{cut}_H(S) \leq \mathrm{cut}_G(S) \leq (1+\varepsilon)\mathrm{cut}_H(S)$$

Graph Sparsification: What is **good** sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \text{cut}_G(S)$

$$(1-\varepsilon)\text{cut}_H(S) \leq \text{cut}_G(S) \leq (1+\varepsilon)\text{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

Graph Sparsification: What is **good** sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \mathrm{cut}_G(S)$

$$(1-\varepsilon)\mathrm{cut}_H(S) \leq \mathrm{cut}_G(S) \leq (1+\varepsilon)\mathrm{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

If we ask this only for $\mathbf{f} \in \{0,1\}^n \rightarrow (1+\varepsilon)$-cut similar combinatorial
Benczúr & Karger (1996)

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \mathrm{cut}_G(S)$

$$(1-\varepsilon)\mathrm{cut}_H(S) \leq \mathrm{cut}_G(S) \leq (1+\varepsilon)\mathrm{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

If we ask this only for $\mathbf{f} \in \{0,1\}^n \to (1+\varepsilon)$-cut similar combinatorial
Benczúr & Karger (1996)

If we ask this for all $\mathbf{f} \in \mathbb{R}^n \to$

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \text{cut}_G(S)$

$$(1-\varepsilon)\text{cut}_H(S) \le \text{cut}_G(S) \le (1+\varepsilon)\text{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \le \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \le (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

If we ask this only for $\mathbf{f} \in \{0,1\}^n \rightarrow (1+\varepsilon)$-cut similar combinatorial
Benczúr & Karger (1996)

If we ask this for all $\mathbf{f} \in \mathbb{R}^n \rightarrow (1+\varepsilon)$-**spectrally similar**
Spielman & Teng (2004)

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \mathrm{cut}_G(S)$

$$(1-\varepsilon)\mathrm{cut}_H(S) \leq \mathrm{cut}_G(S) \leq (1+\varepsilon)\mathrm{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

If we ask this only for $\mathbf{f} \in \{0,1\}^n \rightarrow (1+\varepsilon)$-cut similar combinatorial
Benczúr & Karger (1996)

If we ask this for all $\mathbf{f} \in \mathbb{R}^n \rightarrow (1+\varepsilon)$-**spectrally similar**
Spielman & Teng (2004)

Spectral sparsifiers are stronger!

Graph Sparsification: What is good sparse?

Recall if $\mathbf{f} \in \{0,1\}^n$ represents S then $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \mathrm{cut}_G(S)$

$$(1-\varepsilon)\mathrm{cut}_H(S) \leq \mathrm{cut}_G(S) \leq (1+\varepsilon)\mathrm{cut}_H(S)$$

becomes

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f}$$

If we ask this only for $\mathbf{f} \in \{0,1\}^n \rightarrow (1+\varepsilon)$-cut similar combinatorial
Benczúr & Karger (1996)

If we ask this for all $\mathbf{f} \in \mathbb{R}^n \rightarrow$ **$(1+\varepsilon)$-spectrally similar**
Spielman & Teng (2004)

Spectral sparsifiers are stronger!

but checking for spectral similarity is easier

Spectral Graph Sparsification

Rayleigh-Ritz gives:

Spectral Graph Sparsification

Rayleigh-Ritz gives:

$$\lambda_{\min} = \min \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}} \quad \text{and} \quad \lambda_{\max} = \max \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}}$$

Spectral Graph Sparsification

Rayleigh-Ritz gives:

$$\lambda_{\min} = \min \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}} \quad \text{and} \quad \lambda_{\max} = \max \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}}$$

What can we say about $\lambda_i(G)$ and $\lambda_i(H)$?

Spectral Graph Sparsification

Rayleigh-Ritz gives:

$$\lambda_{\min} = \min \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}} \quad \text{and} \quad \lambda_{\max} = \max \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}}$$

What can we say about $\lambda_i(G)$ and $\lambda_i(H)$?

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f}$$

Spectral Graph Sparsification

Rayleigh-Ritz gives:

$$\lambda_{\min} = \min \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}} \quad \text{and} \quad \lambda_{\max} = \max \frac{\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}}$$

What can we say about $\lambda_i(G)$ and $\lambda_i(H)$?

$$(1-\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} \leq \mathbf{f}^\mathsf{T}\mathbf{L}_H\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f}$$

Eigenvalues are approximated well!

$$(1-\varepsilon)\lambda_i(G) \leq \lambda_i(H) \leq (1+\varepsilon)\lambda_i(G)$$

Using matrix ordering notation $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Spectral Graph Sparsification

Rayleigh-Ritz gives:

$$\lambda_{\min} = \min \frac{\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x}}{\mathbf{x}^{\mathsf{T}}\mathbf{x}} \quad \text{and} \quad \lambda_{\max} = \max \frac{\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x}}{\mathbf{x}^{\mathsf{T}}\mathbf{x}}$$

What can we say about $\lambda_i(G)$ and $\lambda_i(H)$?

$$(1-\varepsilon)\mathbf{f}^{\mathsf{T}}\mathbf{L}_G\mathbf{f} \leq \mathbf{f}^{\mathsf{T}}\mathbf{L}_H\mathbf{f} \leq (1+\varepsilon)\mathbf{f}^{\mathsf{T}}\mathbf{L}_G\mathbf{f}$$

Eigenvalues are approximated well!

$$(1-\varepsilon)\lambda_i(G) \leq \lambda_i(H) \leq (1+\varepsilon)\lambda_i(G)$$

Using matrix ordering notation $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

As a consequence, $\arg\min_{\mathbf{x}} \|\mathbf{L}_H\mathbf{x} - \mathbf{b}\| \approx \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}\|$

Spectral Graph Sparsification

Let us consider unweighted graphs: $w_{ij} \in \{0, 1\}$

$$\mathbf{L}_G = \sum_{ij} w_{ij} \mathbf{L}_{ij} = \sum_{ij \in E} \mathbf{L}_{ij}$$

Spectral Graph Sparsification

Let us consider unweighted graphs: $w_{ij} \in \{0, 1\}$

$$\mathbf{L}_G = \sum_{ij} w_{ij} \mathbf{L}_{ij} = \sum_{ij \in E} \mathbf{L}_{ij} = \sum_{ij \in E} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^\mathsf{T}$$

Spectral Graph Sparsification

Let us consider unweighted graphs: $w_{ij} \in \{0, 1\}$

$$\mathbf{L}_G = \sum_{ij} w_{ij}\mathbf{L}_{ij} = \sum_{ij \in E} \mathbf{L}_{ij} = \sum_{ij \in E} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^\mathsf{T} = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T}$$

Spectral Graph Sparsification

Let us consider unweighted graphs: $w_{ij} \in \{0, 1\}$

$$\mathbf{L}_G = \sum_{ij} w_{ij}\mathbf{L}_{ij} = \sum_{ij\in E} \mathbf{L}_{ij} = \sum_{ij\in E} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^\mathsf{T} = \sum_{e\in E} \mathbf{b}_e\mathbf{b}_e^\mathsf{T}$$

We look for a **subgraph** H

$$\mathbf{L}_H = \sum_{e\in E} s_e\mathbf{b}_e\mathbf{b}_e^\mathsf{T}$$

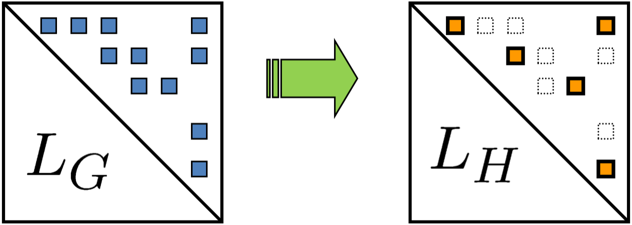
Spectral Graph Sparsification

Let us consider unweighted graphs: $w_{ij} \in \{0,1\}$

$$\mathbf{L}_G = \sum_{ij} w_{ij}\mathbf{L}_{ij} = \sum_{ij\in E} \mathbf{L}_{ij} = \sum_{ij\in E} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^{\mathsf{T}} = \sum_{e\in E} \mathbf{b}_e\mathbf{b}_e^{\mathsf{T}}$$

We look for a **subgraph** H

$$\mathbf{L}_H = \sum_{e\in E} s_e\mathbf{b}_e\mathbf{b}_e^{\mathsf{T}} \quad \text{where } s_e \text{ is a new weight of edge e}$$



https://math.berkeley.edu/~nikhil/

Spectral Graph Sparsification

We want $\quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Spectral Graph Sparsification

We want $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Equivalent, given $\mathbf{L}_G = \sum_{e\in E} \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}}$

Spectral Graph Sparsification

$$\text{We want} \quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$$

$$\text{Equivalent, given } \mathbf{L}_G = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} \text{ find } \mathbf{s}\text{, s.t. } \mathbf{L}_G \preceq \sum_{e \in E} s_e \mathbf{b}_e \mathbf{b}_e^\mathsf{T} \preceq \kappa \cdot \mathbf{L}_G$$

Spectral Graph Sparsification

$$\text{We want} \quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$$

Equivalent, given $\mathbf{L}_G = \sum_{e\in E} \mathbf{b}_e\mathbf{b}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{L}_G \preceq \sum_{e\in E} s_e\mathbf{b}_e\mathbf{b}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{L}_G$

Forget $\mathbf{L}$, given $\mathbf{A} = \sum_{e\in E} \mathbf{a}_e\mathbf{a}_e^\mathsf{T}$

Spectral Graph Sparsification

$$\text{We want} \quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$$

$$\text{Equivalent, given } \mathbf{L}_G = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} \text{ find } \mathbf{s}, \text{ s.t. } \mathbf{L}_G \preceq \sum_{e \in E} s_e \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{L}_G$$

$$\text{Forget } \mathbf{L}, \text{ given } \mathbf{A} = \sum_{e \in E} \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}} \text{ find } \mathbf{s}, \text{ s.t. } \mathbf{A} \preceq \sum_{e \in E} s_e \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{A}$$

Spectral Graph Sparsification

We want $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Equivalent, given $\mathbf{L}_G = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{L}_G \preceq \sum_{e \in E} s_e \mathbf{b}_e \mathbf{b}_e^\mathsf{T} \preceq \kappa \cdot \mathbf{L}_G$

Forget $\mathbf{L}$, given $\mathbf{A} = \sum_{e \in E} \mathbf{a}_e \mathbf{a}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{A} \preceq \sum_{e \in E} s_e \mathbf{a}_e \mathbf{a}_e^\mathsf{T} \preceq \kappa \cdot \mathbf{A}$

Same as, given $\mathbf{I} = \sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$

Spectral Graph Sparsification

We want $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Equivalent, given $\mathbf{L}_G = \sum_{e\in E} \mathbf{b}_e\mathbf{b}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{L}_G \preceq \sum_{e\in E} s_e\mathbf{b}_e\mathbf{b}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{L}_G$

Forget $\mathbf{L}$, given $\mathbf{A} = \sum_{e\in E} \mathbf{a}_e\mathbf{a}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{A} \preceq \sum_{e\in E} s_e\mathbf{a}_e\mathbf{a}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{A}$

Same as, given $\mathbf{I} = \sum_{e\in E} \mathbf{v}_e\mathbf{v}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{I} \preceq \sum_{e\in E} s_e\mathbf{v}_e\mathbf{v}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{I}$

Spectral Graph Sparsification

We want $(1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Equivalent, given $\mathbf{L}_G = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}}$ find $\mathbf{s}$, s.t. $\mathbf{L}_G \preceq \sum_{e \in E} s_e \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{L}_G$

Forget $\mathbf{L}$, given $\mathbf{A} = \sum_{e \in E} \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}}$ find $\mathbf{s}$, s.t. $\mathbf{A} \preceq \sum_{e \in E} s_e \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{A}$

Same as, given $\mathbf{I} = \sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}}$ find $\mathbf{s}$, s.t. $\mathbf{I} \preceq \sum_{e \in E} s_e \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{I}$

How to get it?

Spectral Graph Sparsification

$$\text{We want} \quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$$

$$\text{Equivalent, given } \mathbf{L}_G = \sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} \text{ find } \mathbf{s}, \text{ s.t. } \mathbf{L}_G \preceq \sum_{e \in E} s_e \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{L}_G$$

$$\text{Forget } \mathbf{L}, \text{ given } \mathbf{A} = \sum_{e \in E} \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}} \text{ find } \mathbf{s}, \text{ s.t. } \mathbf{A} \preceq \sum_{e \in E} s_e \mathbf{a}_e \mathbf{a}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{A}$$

$$\text{Same as, given } \mathbf{I} = \sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} \text{ find } \mathbf{s}, \text{ s.t. } \mathbf{I} \preceq \sum_{e \in E} s_e \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} \preceq \kappa \cdot \mathbf{I}$$

How to get it? $\mathbf{v}_e \leftarrow \mathbf{A}^{-1/2} \mathbf{a}_e$

Spectral Graph Sparsification

We want $\quad (1-\varepsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1+\varepsilon)\mathbf{L}_G$

Equivalent, given $\mathbf{L}_G = \sum_{e\in E} \mathbf{b}_e\mathbf{b}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{L}_G \preceq \sum_{e\in E} s_e\mathbf{b}_e\mathbf{b}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{L}_G$

Forget $\mathbf{L}$, given $\mathbf{A} = \sum_{e\in E} \mathbf{a}_e\mathbf{a}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{A} \preceq \sum_{e\in E} s_e\mathbf{a}_e\mathbf{a}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{A}$

Same as, given $\mathbf{I} = \sum_{e\in E} \mathbf{v}_e\mathbf{v}_e^\mathsf{T}$ find $\mathbf{s}$, s.t. $\mathbf{I} \preceq \sum_{e\in E} s_e\mathbf{v}_e\mathbf{v}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{I}$

How to get it? $\mathbf{v}_e \leftarrow \mathbf{A}^{-1/2}\mathbf{a}_e$

Then $\sum_{e\in E} s_e\mathbf{v}_e\mathbf{v}_e^\mathsf{T} \approx \mathbf{I} \iff \sum_{e\in E} s_e\mathbf{a}_e\mathbf{a}_e^\mathsf{T} \approx \mathbf{A}$

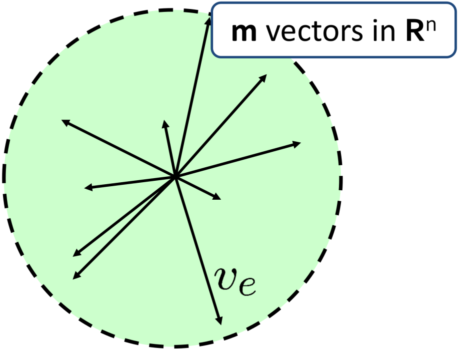
multiplying by $\mathbf{A}^{1/2}$ on both sides

Spectral Graph Sparsification: Intuition

How does $\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} = \mathbf{I}$ look like geometrically?

Spectral Graph Sparsification: Intuition

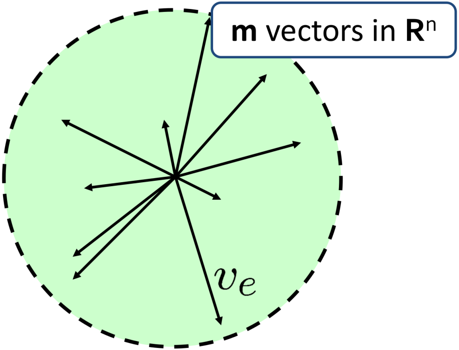
How does $\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} = \mathbf{I}$ look like geometrically?



Decomposition of identity: $\forall \mathbf{u}$ (unit vector): $\sum_{e \in E} (\mathbf{u}^{\mathsf{T}} \mathbf{v}_e)^2 = 1$

Spectral Graph Sparsification: Intuition

How does $\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$ look like geometrically?



Decomposition of identity: $\forall \mathbf{u}$ (unit vector): $\sum_{e \in E} (\mathbf{u}^\mathsf{T} \mathbf{v}_e)^2 = 1$

moment ellipse is a sphere

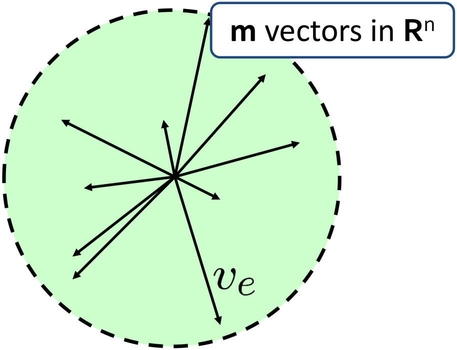
https://math.berkeley.edu/~nikhil/

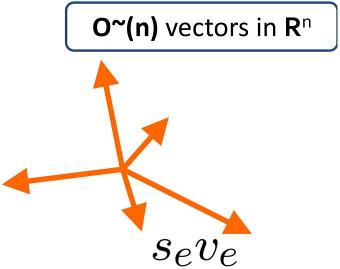
Spectral Graph Sparsification: Intuition

What are we doing by choosing H?

Spectral Graph Sparsification: Intuition

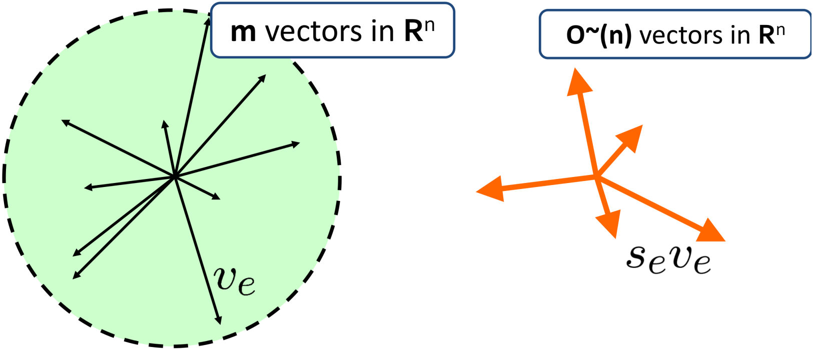
What are we doing by choosing H?





Spectral Graph Sparsification: Intuition

What are we doing by choosing H?



We take a subset of these $\mathbf{e}_e$s and scale them!

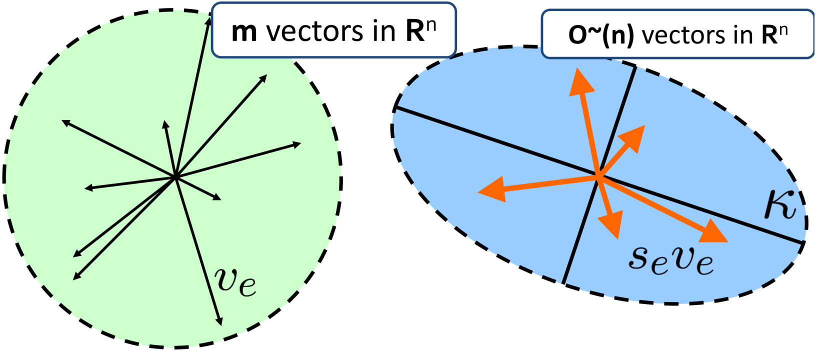
https://math.berkeley.edu/~nikhil/

Spectral Graph Sparsification: Intuition

What kind of scaling go we want?

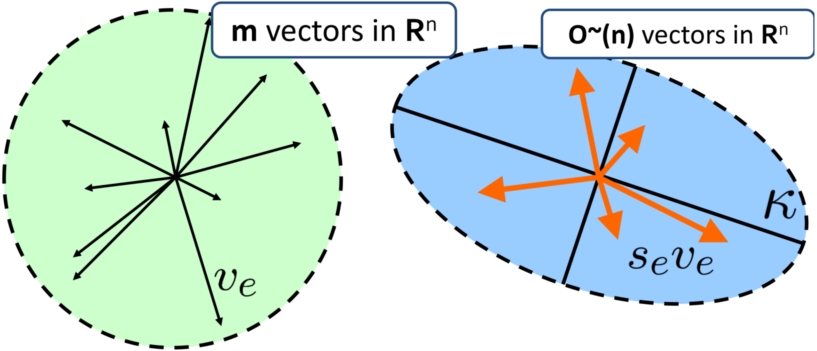
Spectral Graph Sparsification: Intuition

What kind of scaling go we want?



Spectral Graph Sparsification: Intuition

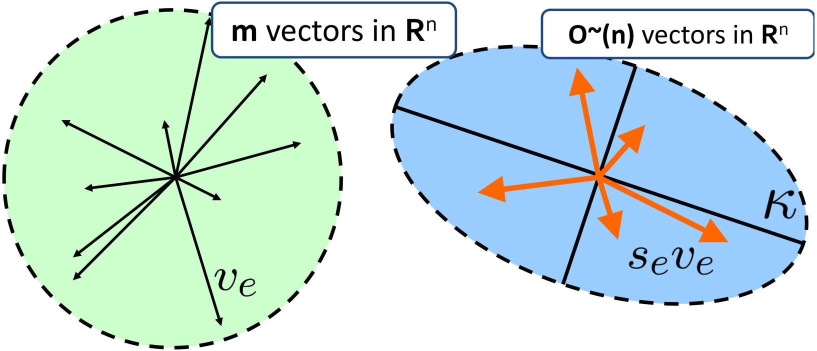
What kind of scaling go we want?



Such that the blue ellipsoid looks like identity!

Spectral Graph Sparsification: Intuition

What kind of scaling go we want?



Such that the blue ellipsoid looks like identity!

the blue eigenvalues are between 1 and κ

https://math.berkeley.edu/~nikhil/

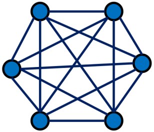
Spectral Graph Sparsification: Intuition

Example: What happens with K_n?

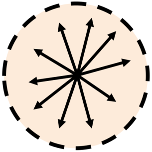
Spectral Graph Sparsification: Intuition

Example: What happens with K_n?

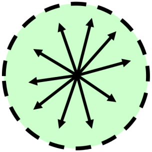
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



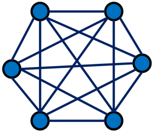
$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$



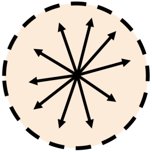
Spectral Graph Sparsification: Intuition

Example: What happens with K_n?

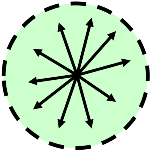
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$

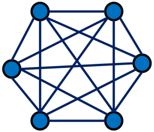


It is already isotropic! (looks like a sphere)

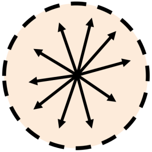
Spectral Graph Sparsification: Intuition

Example: What happens with K_n?

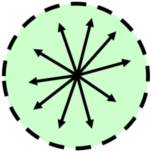
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$



It is already isotropic! (looks like a sphere)

rescaling $\mathbf{v}_e = \mathbf{L}^{-1/2}\mathbf{b}_e$ does not change the shape

https://math.berkeley.edu/~nikhil/

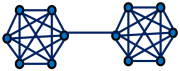
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

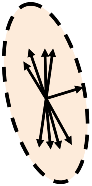
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

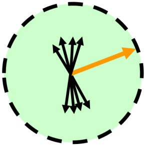
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



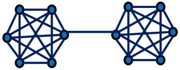
$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$



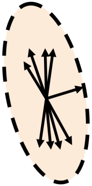
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

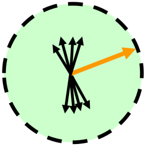
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$

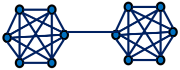


The vector corresponding to the link gets stretched!

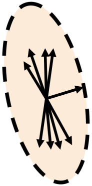
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

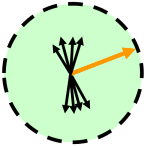
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^\mathsf{T} = \mathbf{L}_G$



$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^\mathsf{T} = \mathbf{I}$



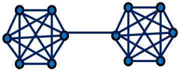
The vector corresponding to the link gets stretched!

because this transformation makes all the directions important

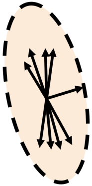
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

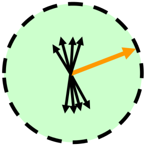
K_n graph



$\sum_{e \in E} \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} = \mathbf{L}_G$



$\sum_{e \in E} \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}} = \mathbf{I}$



The vector corresponding to the link gets stretched!

because this transformation makes all the directions important

rescaling reveals the vectors that are critical

https://math.berkeley.edu/~nikhil/

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2$$

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2$$

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^\mathsf{T}\mathbf{L}_G^{-1}\mathbf{b}_e$$

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^\mathsf{T}\mathbf{L}_G^{-1}\mathbf{b}_e = R_{\text{eff}}(e)$$

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^\mathsf{T}\mathbf{L}_G^{-1}\mathbf{b}_e = R_{\text{eff}}(e)$$

reminder $R_{\text{eff}}(e)$ is the potential difference between the nodes when injecting a unit current

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\left\|\mathbf{v}_e\right\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^{\mathsf{T}}\mathbf{L}_G^{-1}\mathbf{b}_e = R_{\text{eff}}(e)$$

reminder $R_{\text{eff}}(e)$ is the potential difference between the nodes when injecting a unit current

In other words: $R_{\text{eff}}(e)$ is related to the edge importance!

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\left\|\mathbf{v}_e\right\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^{\mathsf{T}}\mathbf{L}_G^{-1}\mathbf{b}_e = R_{\text{eff}}(e)$$

reminder $R_{\text{eff}}(e)$ is the potential difference between the nodes when injecting a unit current

In other words: $R_{\text{eff}}(e)$ is related to the edge importance!

Electrical intuition: We want to find an electrically similar H and the importance of the edge is its effective resistance $R_{\text{eff}}(e)$.

Spectral Graph Sparsification: Intuition

What it this rescaling $\mathbf{v}_e = \mathbf{L}_G^{-1/2}\mathbf{b}_e$ doing to the norm?

$$\|\mathbf{v}_e\|^2 = \left\|\mathbf{L}_G^{-1/2}\mathbf{b}_e\right\|^2 = \mathbf{b}_e^\mathsf{T}\mathbf{L}_G^{-1}\mathbf{b}_e = R_{\text{eff}}(e)$$

reminder $R_{\text{eff}}(e)$ is the potential difference between the nodes when injecting a unit current

In other words: $R_{\text{eff}}(e)$ is related to the edge importance!

Electrical intuition: We want to find an electrically similar H and the importance of the edge is its effective resistance $R_{\text{eff}}(e)$.

Edges with higher R_{eff} are more **electrically significant**!

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^{\mathsf{T}}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n/\varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n / \varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n/\varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Does this work?

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n/\varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Does this work?

Application of Matrix Chernoff Bound by Rudelson (1999)

$$1 - \varepsilon \prec \lambda \left(\sum_e s_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T} \right) \prec 1 + \varepsilon$$

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n / \varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Does this work?

Application of Matrix Chernoff Bound by Rudelson (1999)

$$1 - \varepsilon \prec \lambda \left(\sum_e s_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T} \right) \prec 1 + \varepsilon$$

finer bounds now available

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n/\varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Does this work?

Application of Matrix Chernoff Bound by Rudelson (1999)

$$1 - \varepsilon \prec \lambda \left(\sum_e s_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T} \right) \prec 1 + \varepsilon$$

finer bounds now available

What is the the biggest problem here?

Spectral Graph Sparsification

Todo: Given $\mathbf{I} = \sum_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T}$, find a sparse reweighting.

Randomized algorithm that finds $\mathbf{s}$:

- Sample $n \log n/\varepsilon^2$ with replacement $p_i \propto \|\mathbf{v}_e\|^2$ (resistances)
- Reweigh: $s_i = 1/p_i$ (to be unbiased)

Does this work?

Application of Matrix Chernoff Bound by Rudelson (1999)

$$1 - \varepsilon \prec \lambda \left(\sum_e s_e \mathbf{v}_e \mathbf{v}_e^\mathsf{T} \right) \prec 1 + \varepsilon$$

finer bounds now available

What is the the biggest problem here? Getting the p_is!

Spectral Graph Sparsification

We want to make this algorithm fast.

Spectral Graph Sparsification

We want to make this algorithm fast.

How can we compute the effective resistances?

Spectral Graph Sparsification

We want to make this algorithm fast.

How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}}\widehat{\mathbf{x}}$

Spectral Graph Sparsification

We want to make this algorithm fast.

How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}}\widehat{\mathbf{x}}$

Gaussian Elimination $\quad \mathcal{O}(n^3)$

Spectral Graph Sparsification

We want to make this algorithm fast.
How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}}\widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$

Spectral Graph Sparsification

We want to make this algorithm fast.
How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^\mathsf{T}\widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$
Spielman & Teng (2004)	$\mathcal{O}(m \log^{30} n)$

Spectral Graph Sparsification

We want to make this algorithm fast.
How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G \mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}} \widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$
Spielman & Teng (2004)	$\mathcal{O}(m \log^{30} n)$
Koutis, Miller, and Peng (2010)	$\mathcal{O}(m \log n)$

Spectral Graph Sparsification

We want to make this algorithm fast.
How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^\mathsf{T}\widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$
Spielman & Teng (2004)	$\mathcal{O}(m \log^{30} n)$
Koutis, Miller, and Peng (2010)	$\mathcal{O}(m \log n)$

- Fast solvers for SDD systems:

Spectral Graph Sparsification

We want to make this algorithm fast.
How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}}\widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$
Spielman & Teng (2004)	$\mathcal{O}(m \log^{30} n)$
Koutis, Miller, and Peng (2010)	$\mathcal{O}(m \log n)$

- Fast solvers for SDD systems:
  - ↳ use sparsification internally

all the way until you hit the turtles

Spectral Graph Sparsification

We want to make this algorithm fast.

How can we compute the effective resistances?

Solve a linear system $\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{L}_G\mathbf{x} - \mathbf{b}_e\|$ and then $R_{\text{eff}} = \mathbf{b}_e^{\mathsf{T}}\widehat{\mathbf{x}}$

Gaussian Elimination	$\mathcal{O}(n^3)$
Fast Matrix Multiplication	$\mathcal{O}(n^{2.37})$
Spielman & Teng (2004)	$\mathcal{O}(m\log^{30} n)$
Koutis, Miller, and Peng (2010)	$\mathcal{O}(m\log n)$

- Fast solvers for SDD systems:
  - ↳ use sparsification internally

    all the way until you hit the turtles

    still unfeasible when m is large

Spectral Graph Sparsification

Chicken and egg problem

We need R_{eff} to compute a sparsifier H ↰

↳ We need a sparsifier H to compute R_{eff}

Spectral Graph Sparsification

Chicken and egg problem

We need R_{eff} to compute a sparsifier H ↰

↳ We need a sparsifier H to compute R_{eff}

Sampling according to approximate effective resistances
$R_{\text{eff}} \leq \widetilde{R}_{\text{eff}} \leq \alpha R_{\text{eff}}$ give approximate sparsifier $\mathbf{L}_G \preceq \mathbf{L}_H \preceq \alpha\kappa\mathbf{L}_G$

Spectral Graph Sparsification

Chicken and egg problem

We need R_{eff} to compute a sparsifier H ↰
↳ We need a sparsifier H to compute R_{eff}

Sampling according to approximate effective resistances
$R_{\text{eff}} \leq \widetilde{R}_{\text{eff}} \leq \alpha R_{\text{eff}}$ give approximate sparsifier $\mathbf{L}_G \preceq \mathbf{L}_H \preceq \alpha\kappa\mathbf{L}_G$

Start with very poor approximation $\widetilde{R}_{\text{eff}}$ and poor sparsifier.

Use $\widetilde{R}_{\text{eff}}$ to compute an improved approximate sparsifier H ↰
↳ Use the sparsifier H to compute improved approximate $\widetilde{R}_{\text{eff}}$

Spectral Graph Sparsification

Chicken and egg problem

We need R_{eff} to compute a sparsifier H ↰
↳ We need a sparsifier H to compute R_{eff}

Sampling according to approximate effective resistances
$R_{\text{eff}} \leq \widetilde{R}_{\text{eff}} \leq \alpha R_{\text{eff}}$ give approximate sparsifier $\mathbf{L}_G \preceq \mathbf{L}_H \preceq \alpha\kappa\mathbf{L}_G$

Start with very poor approximation $\widetilde{R}_{\text{eff}}$ and poor sparsifier.
Use $\widetilde{R}_{\text{eff}}$ to compute an improved approximate sparsifier H ↰
↳ Use the sparsifier H to compute improved approximate $\widetilde{R}_{\text{eff}}$

Computing $\widetilde{R}_{\text{eff}}$ using the sparsifier is fast ($m = \boldsymbol{O}(n\log(n))$), and not too many iterations are necessary.

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL
- More in general, solving Strongly Diagonally Dominant (SDD) linear systems

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL
- More in general, solving Strongly Diagonally Dominant (SDD) linear systems

  ↳ electric circuit, fluid equations, finite elements methods

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL
- More in general, solving Strongly Diagonally Dominant (SDD) linear systems
  - ↳ electric circuit, fluid equations, finite elements methods
- Various embeddings: k-means, spectral clustering.

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL
- More in general, solving Strongly Diagonally Dominant (SDD) linear systems
  - ↳ electric circuit, fluid equations, finite elements methods
- Various embeddings: k-means, spectral clustering.

What can I use sparsifiers for?

- Graph linear systems: minimum cut, maximum flow, Laplacian regression, SSL
- More in general, solving Strongly Diagonally Dominant (SDD) linear systems
  - ↳ electric circuit, fluid equations, finite elements methods
- Various embeddings: k-means, spectral clustering.

But what if my problems have no use for spectral guarantees?

Or if my boss does not trust approximation methods

Distributed graph processing

Large graphs do not fit in memory

Get more memory

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory
Or fast memory but divided among many machines

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory
Or fast memory but divided among many machines

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory
Or fast memory but divided among many machines

Many challenges

Needs to be scalable

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory
Or fast memory but divided among many machines

Many challenges

Needs to be scalable

↳ minimimize pass over data / communication costs

Distributed graph processing

Large graphs do not fit in memory

- Get more memory
  - ↳ Either slower but larger memory
    Or fast memory but divided among many machines

Many challenges

- Needs to be scalable
  - ↳ minimimize pass over data / communication costs
- Needs to be consistent

Distributed graph processing

Large graphs do not fit in memory

Get more memory

↳ Either slower but larger memory
Or fast memory but divided among many machines

Many challenges

Needs to be scalable

↳ minimimize pass over data / communication costs

Needs to be consistent

↳ updates should propagate properly

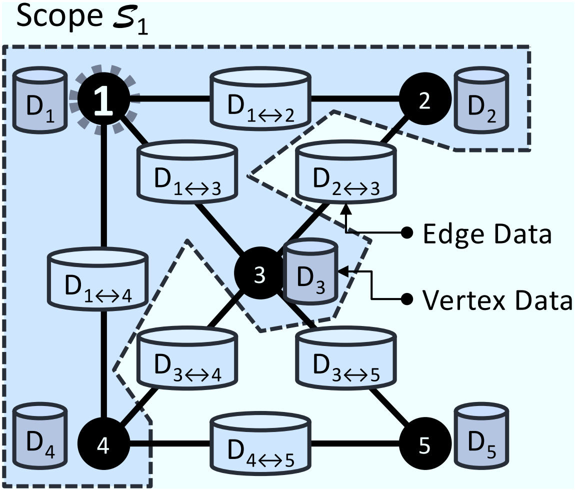
Distributed graph processing

Different choices have different impacts: for example splitting the graph according to nodes or according to edges.

Many computation models (academic and commercial) each with its pros and cons

- MapReduce
- MPI
- Pregel
- **Graphlab**

The GraphLab abstraction



The GraphLab abstraction

```
In [1]: import sframe

In [2]: edges = sframe.SFrame.read_csv('/media/sf_share/td3_example_edges.csv')

In [3]: vertices = sframe.SFrame.read_csv('/media/sf_share/td3_example_vertices.csv')

In [4]: G = sframe.SGraph(edges= edges, vertices=vertices, src_field='src', dst_field='dst')

In [5]: G

Out[5]: SGraph({'num_edges': 26, 'num_vertices': 9})
        Vertex Fields:['__id', 'f']
        Edge Fields:['__src_id', '__dst_id', 'weight']
```

The GraphLab abstraction

Under the hood: tabular representation

```
Columns:
    __id int
    f float

Rows: 9

Data:
+------+------+
| __id |  f   |
+------+------+
|  5   | 0.51 |
|  7   | 0.82 |
|  10  | 0.08 |
|  2   | 0.82 |
|  6   | 0.85 |
|  9   | 0.83 |
|  3   | 0.18 |
|  1   | 0.35 |
|  4   | 0.36 |
+------+------+
[9 rows x 2 columns]
```

```
Columns:
    __src_id int
    __dst_id int
    weight float

Rows: 26

Data:
+----------+----------+----------+
| __src_id | __dst_id |  weight  |
+----------+----------+----------+
|    7     |    5     | 0.13185  |
|    5     |    7     | 0.13185  |
|    7     |    7     | 0.026779 |
|    10    |    7     | 0.57121  |
|    7     |    10    | 0.57121  |
|    10    |    2     | 0.94047  |
|    7     |    6     | 0.64528  |
|    5     |    3     | 0.93374  |
|    10    |    3     | 0.31713  |
|    5     |    1     | 0.57796  |
+----------+----------+----------+
[26 rows x 3 columns]
Note: Only the head of the SFrame is printed.
```

The GraphLab abstraction

```
In [1]: import sframe

In [2]: G = sframe.SGraph()

In [3]: G

Out[3]: SGraph({'num_edges': 0, 'num_vertices': 0})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']
```

The GraphLab abstraction

```
In [1]: import sframe

In [2]: G = sframe.SGraph()

In [3]: G

Out[3]: SGraph({'num_edges': 0, 'num_vertices': 0})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']

In [4]: G.add_edges(sframe.Edge(1,2))

Out[4]: SGraph({'num_edges': 1, 'num_vertices': 2})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']
```

The GraphLab abstraction

```
In [1]: import sframe

In [2]: G = sframe.SGraph()

In [3]: G

Out[3]: SGraph({'num_edges': 0, 'num_vertices': 0})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']

In [4]: G.add_edges(sframe.Edge(1,2))

Out[4]: SGraph({'num_edges': 1, 'num_vertices': 2})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']

In [5]: G

Out[5]: SGraph({'num_edges': 0, 'num_vertices': 0})
        Vertex Fields:['__id']
        Edge Fields:['__src_id', '__dst_id']
```

The GraphLab abstraction

- The graph is immutable. why?

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously
  - ↳ We do not know the order of execution

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously
  - ↳ We do not know the order of execution
    We do not even know where the node is stored
    what data can we access?

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously
  ↳ We do not know the order of execution
  We do not even know where the node is stored
  what data can we access?
- The data is stored in the graph itself

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously
  ↳ We do not know the order of execution
  We do not even know where the node is stored
  what data can we access?
- The data is stored in the graph itself
  ↳ only access local data

The GraphLab abstraction

- The graph is immutable. why?
- All computations are executed asyncronously
  ↳ We do not know the order of execution
  We do not even know where the node is stored
  what data can we access?
- The data is stored in the graph itself
  ↳ only access local data
- Functional programming approach

The GraphLab abstraction

```
triple_apply(triple_apply_fn, mutated_fields, input_fields=None)
```

processes all edges asyncronously and in parallel

```
>>> PARALLEL FOR (source, edge, target) AS triple in G:
...     LOCK (triple.source, triple.target)
...     (source, edge, target) = triple_apply_fn(triple)
...     UNLOCK (triple.source, triple.target)
... END PARALLEL FOR
```

- No guarantees on order of execution

The GraphLab abstraction

```
triple_apply(triple_apply_fn, mutated_fields, input_fields=None)
```

processes all edges asyncronously and in parallel

```
>>> PARALLEL FOR (source, edge, target) AS triple in G:
...     LOCK (triple.source, triple.target)
...     (source, edge, target) = triple_apply_fn(triple)
...     UNLOCK (triple.source, triple.target)
... END PARALLEL FOR
```

- No guarantees on order of execution
- Updating `(src,edge,dst)` violates immutability

The GraphLab abstraction

```
triple_apply(triple_apply_fn, mutated_fields, input_fields=None)
```

processes all edges asyncronously and in parallel

```
>>> PARALLEL FOR (source, edge, target) AS triple in G:
...     LOCK (triple.source, triple.target)
...     (source, edge, target) = triple_apply_fn(triple)
...     UNLOCK (triple.source, triple.target)
... END PARALLEL FOR
```

- No guarantees on order of execution
- Updating `(src,edge,dst)` violates immutability
- `triple_apply_fn` receives a copy of `(src,edge,dst)`

The GraphLab abstraction

```
triple_apply(triple_apply_fn, mutated_fields, input_fields=None)
```

processes all edges asyncronously and in parallel

```
>>> PARALLEL FOR (source, edge, target) AS triple in G:
...     LOCK (triple.source, triple.target)
...     (source, edge, target) = triple_apply_fn(triple)
...     UNLOCK (triple.source, triple.target)
... END PARALLEL FOR
```

- No guarantees on order of execution
- Updating `(src,edge,dst)` violates immutability
- `triple_apply_fn` receives a copy of `(src,edge,dst)`
  ↳ returns an updated `(src',edge',dst')`

The GraphLab abstraction

```
triple_apply(triple_apply_fn, mutated_fields, input_fields=None)
```

processes all edges asyncronously and in parallel

```
>>> PARALLEL FOR (source, edge, target) AS triple in G:
...     LOCK (triple.source, triple.target)
...     (source, edge, target) = triple_apply_fn(triple)
...     UNLOCK (triple.source, triple.target)
... END PARALLEL FOR
```

- No guarantees on order of execution
- Updating (src,edge,dst) violates immutability
- triple_apply_fn receives a copy of (src,edge,dst)
  ↳ returns an updated (src',edge',dst')
  use return values to build a new graph

The GraphLab abstraction

triple_apply_fn is a pure function

Function in the mathematical sense, same input gives same output.

```
def triple_apply_fn(src, edge, dst):
    #can only access data stored in src, edge, and dst,
    #three dictionaries containing a copy of the
    #fields indicated in mutated_fields
    f = dst['f']

    #inputs are copies, this does not change original edge
    edge['weight'] = g(f)

    return ({'f': dst['f']}, edge, dst)
```

The GraphLab abstraction

An example, computing degree of nodes

```
def degree_count_fn (src, edge, dst):
    src['degree'] += 1
    dst['degree'] += 1
    return (src, edge, dst)

G_count = G.triple_apply(degree_count_fn, 'degree')
```

The GraphLab abstraction

Slightly more complicated example, suboptimal pagerank

```
#assume each node in G has a field 'degree' and 'pagerank'
#initialize 'pagerank' = 1/n for all nodes

def weight_count_fn (src, edge, dst):
    dst['degree'] += edge['weight']
    return (src, edge, dst)

def pagerank_step_fn (src, edge, dst):
    dst['pagerank'] += (edge['weight']*src['pagerank']
                                     /dst['degree'])
    return (src, edge, dst)

G_pagerank = G.triple_apply(weight_count_fn, 'degree')

while not converged(G_pagerank):
    G_pagerank = G_pagerank.triple_apply(
                        pagerank_step_fn, 'pagerank')
```

The GraphLab abstraction

Slightly more complicated example, suboptimal pagerank

```
#assume each node in G has a field 'degree' and 'pagerank'
#initialize 'pagerank' = 1/n for all nodes

def weight_count_fn (src, edge, dst):
    dst['degree'] += edge['weight']
    return (src, edge, dst)

def pagerank_step_fn (src, edge, dst):
    dst['pagerank'] += (edge['weight']*src['pagerank']
                                  /dst['degree'])
    return (src, edge, dst)

G_pagerank = G.triple_apply(weight_count_fn, 'degree')

while not converged(G_pagerank):
    G_pagerank = G_pagerank.triple_apply(
                        pagerank_step_fn, 'pagerank')
```

How many iterations to convergence?

Michal Valko
michal.valko@inria.fr
ENS Paris-Saclay, MVA 2017/2018
SequeL team, Inria Lille — Nord Europe
https://team.inria.fr/sequel/