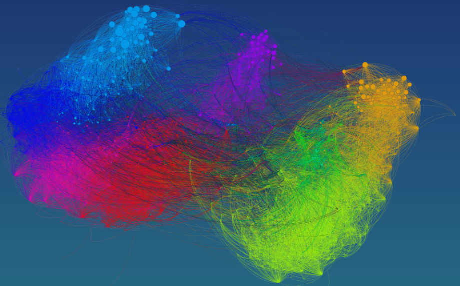
Graphs in Machine Learning

Michal Valko

DeepMind Paris and Inria Lille

TA: Omar Darwiche Domingues with the help of Pierre Perrault

Partially based on material by: Rob Fergus, Nikhil Srivastava, Yiannis Koutis, Joshua Batson, Daniel Spielman



Last Lecture

- Online semi-supervised learning
- Online incremental k-centers
- 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

Previous Lab Session

- 12. 11. 2019 by Omar (+Pierre)
- Content
  - Semi-supervised learning
  - Graph quantization
  - Offline face recognizer
- Short written report
- **Public** questions to piazza
- *Deadline:* 26. 11. 2019

Next Lab Session/Lecture

- 26. 11. 2019 by Marc
- **4**. 12. 2019 - **14h30**-16h30 by Omar (+ Pierre)
- Content: Graph nets

Final class projects

- detailed description on the class website
- preferred option: you come up with the topic
- theory/implementation/review or a combination
- one or two people per project (exceptionally three)
- grade 60%: report + short presentation of the **team**
- deadlines
  - 19. 11. 2019 - **strongly** recommended DL **TODAY!**
  - 26. 11. 2019 - hard DL for taking projects
  - 07. 01. 2020 - submission of the project report
  - 13. 01. 2020 or later - project presentation
- list of suggested topics on piazza

Huge $\mathcal{G}$

when $\mathcal{G}$ does not fit to memory

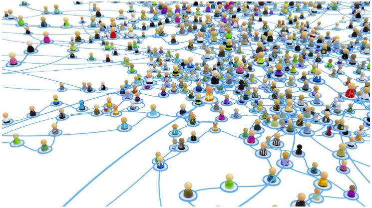
...or when we can't invert $\mathbf{L}$

Sparsify $\mathcal{G}$

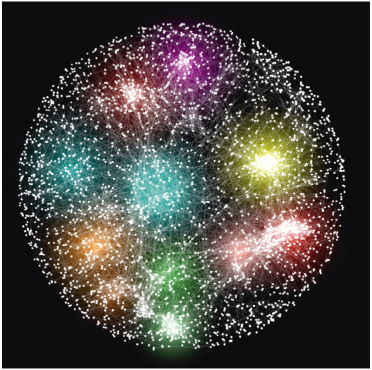
with no assumptions

...and we need to process is anyway

Large scale Machine Learning on Graphs

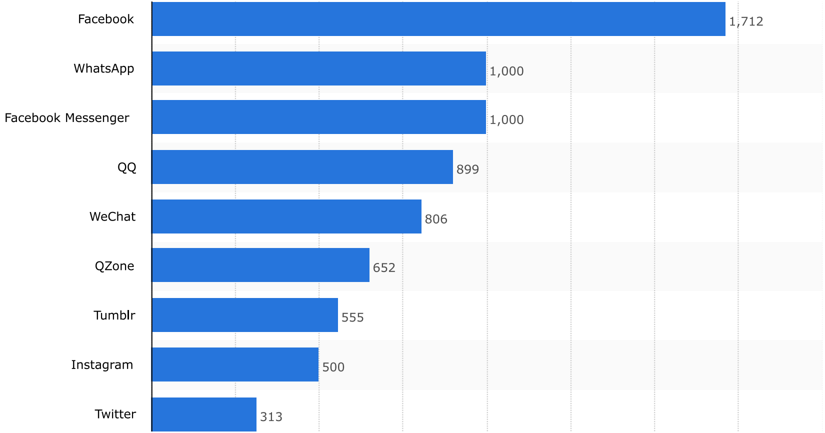


http://blog.carsten-eickhoff.com



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

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:

   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

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
- 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.

- 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.
- 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.

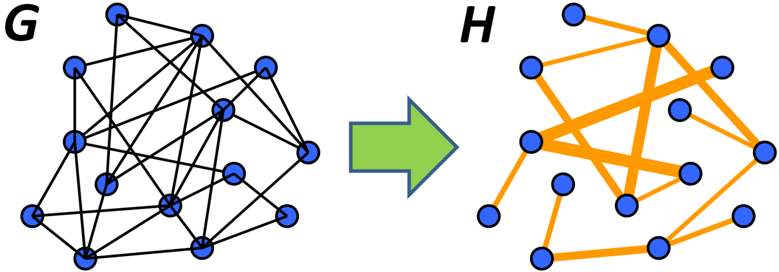
Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ 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?

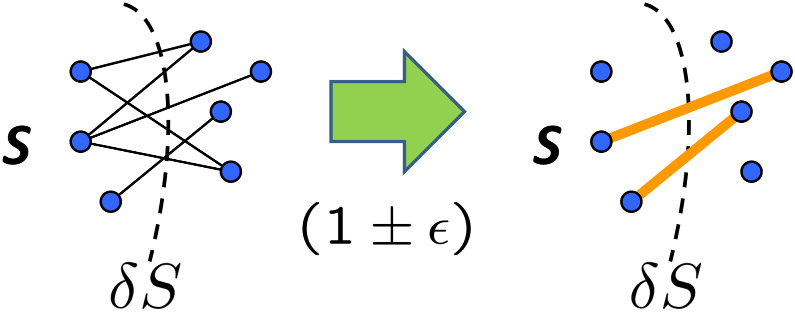
- average degree < 10 is pretty sparse
- for billion nodes even 100 should be ok
- in general: average degree $<$ polylog n

Are all edges important?

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

Graph Sparsification: What is good sparse?

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? 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 $\forall S$

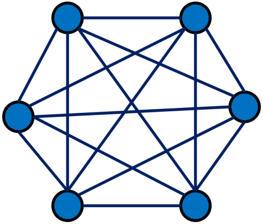
$$(1-\varepsilon)\text{cut}_H(S) \leq \text{cut}_G(S) \leq (1+\varepsilon)\text{cut}_H(S)$$

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

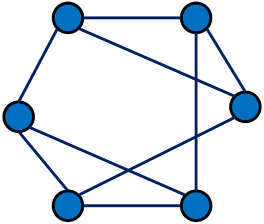
$\forall \varepsilon \; \exists \; (1+\varepsilon)$-cut similar $\widetilde{G}$ with $\mathcal{O}(n \log n/\varepsilon^2)$ edges s.t. $E_H \subseteq E$ and computable in $\mathcal{O}(m \log^3 n + m \log n/\varepsilon^2)$ time n nodes, m edges

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

$G = K_n$



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

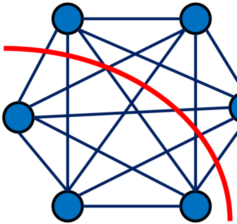


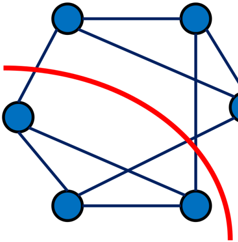
How many edges?

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

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

Graph Sparsification: What is good sparse?





What are the cut weights for any S?

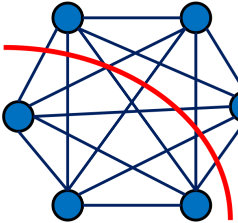
$w_G(\delta S) = |S| \cdot |\overline{S}|$ $\qquad$ $w_H(\delta S) \approx \frac{d}{n} \cdot |S| \cdot |\overline{S}|$

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

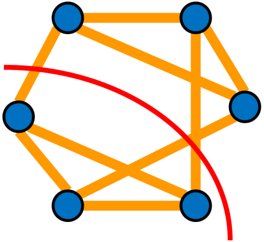
Could be large :(What to do?

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

$G = K_n$



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



What are the cut weights for any S?

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

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

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 $\mathbf{f}^\mathsf{T}\mathbf{L}_G\mathbf{f} = \text{cut}_G(S)$

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

becomes

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

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:

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

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

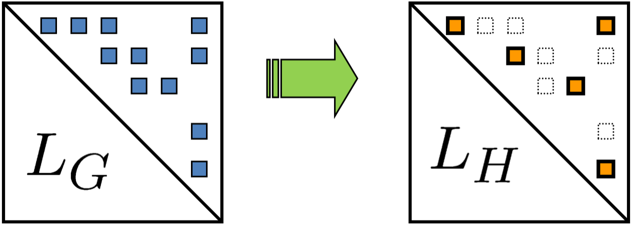
Spectral Graph Sparsification

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

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

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

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. $\mathbf{I} \preceq \sum_{e\in E} s_e\mathbf{v}_e\mathbf{v}_e^\mathsf{T} \preceq \kappa\cdot\mathbf{I}$

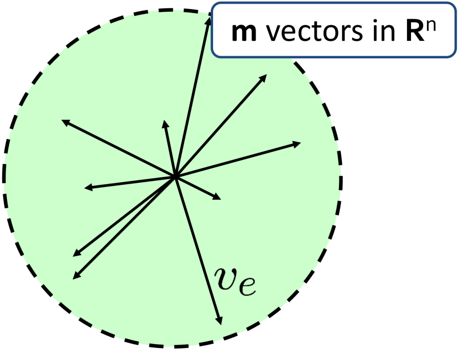
How to get it? $\mathbf{v}_e \leftarrow \mathbf{A}^{-1/2}\mathbf{a}_e$

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?



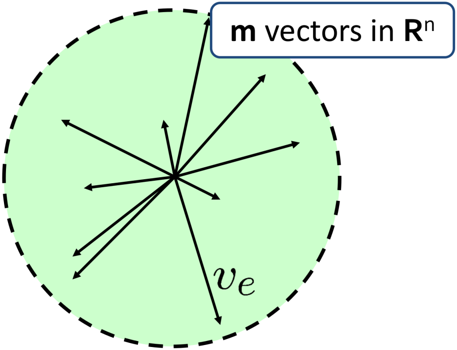
Decomposition of identity: $\forall \mathbf{u}$ (unit vector): $\sum_{e \in E} (\mathbf{u}^\mathsf{T} \mathbf{v}_e)^2 = 1$

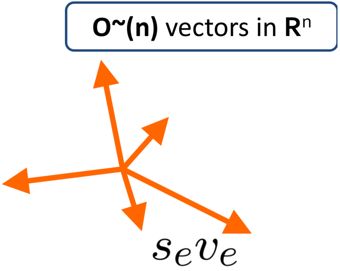
moment ellipse is a sphere

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

Spectral Graph Sparsification: Intuition

What are we doing by choosing H?



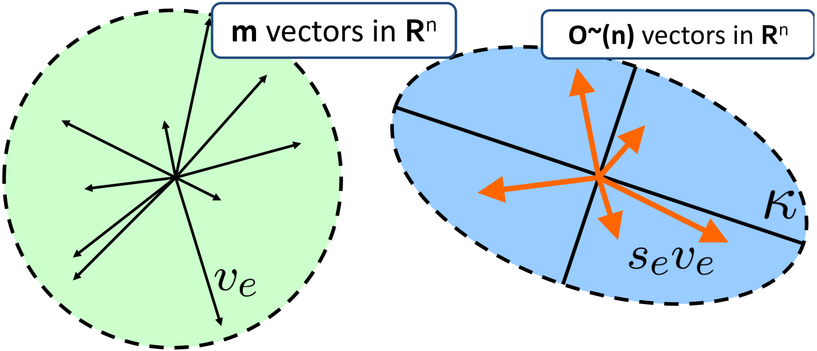


We take a subset of these $\mathbf{e}_e$s and scale them!

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

Spectral Graph Sparsification: Intuition

What kind of scaling go we want?



Such that the blue ellipsoid looks like identity!

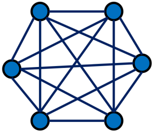
the blue eigenvalues are between 1 and κ

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

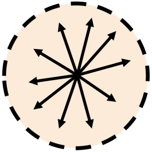
Spectral Graph Sparsification: Intuition

Example: What happens with K_n?

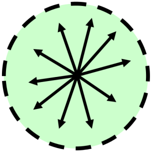
K_n graph



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



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



It is already isotropic! (looks like a sphere)

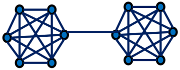
rescaling $\mathbf{v}_e = \mathbf{L}^{-1/2}\mathbf{b}_e$ does not change the shape

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

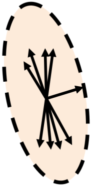
Spectral Graph Sparsification: Intuition

Example: What happens with a dumbbell?

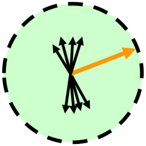
K_n graph



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



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



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?

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

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 $\mathbf{s}$:

- 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.
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}

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
- 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
    ↳ 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**

Graph Spectral Sparsification

Definition ([SS11])

An ε-sparsifier of $\mathcal{G}$ is a re[illegible]ose Laplacian $\mathbf{L}_{\mathcal{H}}$ satisfies

$$(1-\varepsilon)\mathbf{L}_{\mathcal{G}} \preceq \mathbf{L}_{\mathcal{H}} \preceq (1+\varepsilon)\mathbf{L}_{\mathcal{G}} \quad (1)$$

Proposition ([SS11; Kyn+16])

There exists an algorithm that can construct an ε-sparsifier

- with only $\mathcal{O}(n\log(n)/\varepsilon^2)$ edges
- in $\mathcal{O}(m\log^2(n))$ time and $\mathcal{O}(n\log(n)/\varepsilon^2)$ space
- a single pass over the data

Graph Spectral Sparsification in Machine Learning

Laplacian smoothing (denoising): given $\mathbf{y} \triangleq \mathbf{f}^\star + \xi$ and $\mathcal{G}$ compute

$$\min_{\mathbf{f}\in\mathbb{R}^n} (\mathbf{f}-\mathbf{y})^\mathsf{T}(\mathbf{f}-\mathbf{y}) + \lambda \mathbf{f}^\mathsf{T}\mathbf{L}_\mathcal{G}\mathbf{f} \qquad (2)$$

	Preproc	Time	Space
$\widehat{\mathbf{f}} = (\lambda \mathbf{L}_\mathcal{G} + \mathbf{I})^{-1}\mathbf{y}$	0	$\mathcal{O}(m\log(n))$	$\mathcal{O}(m)$
$\widetilde{\mathbf{f}} = (\lambda \mathbf{L}_\mathcal{H} + \mathbf{I})^{-1}\mathbf{y}$	$\mathcal{O}(m\log^2(n))$	$\mathcal{O}(n\log^2(n))$	$\mathcal{O}(n\log(n))$

Large computational improvement
↳ accuracy guarantees! [SWT16]

Need to approximate spectrum only up to regularization level λ

Ridge Graph Spectral Sparsification

Laplacian $\mathbf{L}_{\mathcal{H}}$

$$(1-\varepsilon)\mathbf{L}_{\mathcal{G}} - \varepsilon\gamma\mathbf{I} \preceq \mathbf{L}_{\mathcal{H}} \preceq (1+\varepsilon)\mathbf{L}_{\mathcal{G}} + \varepsilon\gamma\mathbf{I} \quad (3)$$

Mixed multiplicative/additive error

- large (i.e. $\geq \gamma$) directions reconstructed accurately
- small (i.e. $\leq \gamma$) directions uniformly approximated ($\gamma\mathbf{I}$)

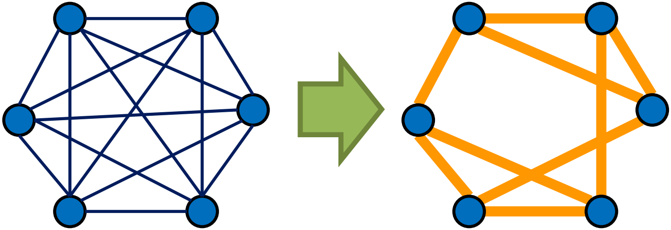
Adapted from Randomized Linear Algebra (RLA) community
↳ PSD matrix low-rank approx. [AM15]

RLA $\rightarrow$ Graph: Improve over $\mathcal{O}(n \log n)$ exploiting regularization

Graph $\rightarrow$ RLA: Exploit $\mathbf{L}_{\mathcal{G}}$ structure for fast (ε, γ)-sparsification

How to construct an ε-sparsifier

For complete graphs, sample $\mathcal{O}(n \log(n))$ edges uniformly and reweight

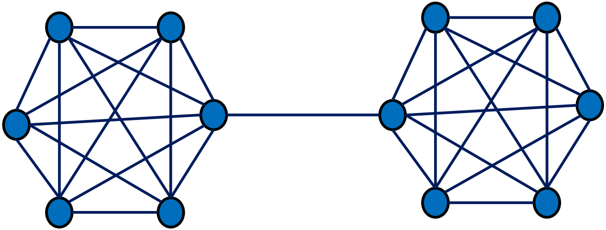


How to construct an ε-sparsifier

For generic graphs, sample $\mathcal{O}(n \log(n))$ edges uniformly?

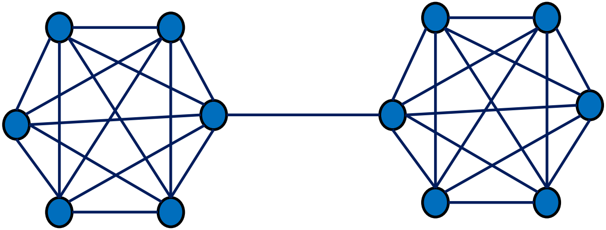
How to construct an ε-sparsifier

For generic graphs, sample $\mathcal{O}(n\log(n))$ edges uniformly?



How to construct an ε-sparsifier

For generic graphs, sample $\mathcal{O}(n\log(n))$ edges using effective resistance



Effective resistance $r_e = \mathbf{b}_e^\top \mathbf{L}_\mathcal{G}^+ \mathbf{b}_e$ of an edge

↳ inverse of number of alternative paths

↳ sum of r_e is $n-1$

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

How to construct an (ε, γ)-sparsifier

Definition

γ-effective resistance: $r_e(\gamma) = \mathbf{b}_e^\mathsf{T}(\mathbf{L}_\mathcal{G} + \gamma\mathbf{I})^{-1}\mathbf{b}_e$

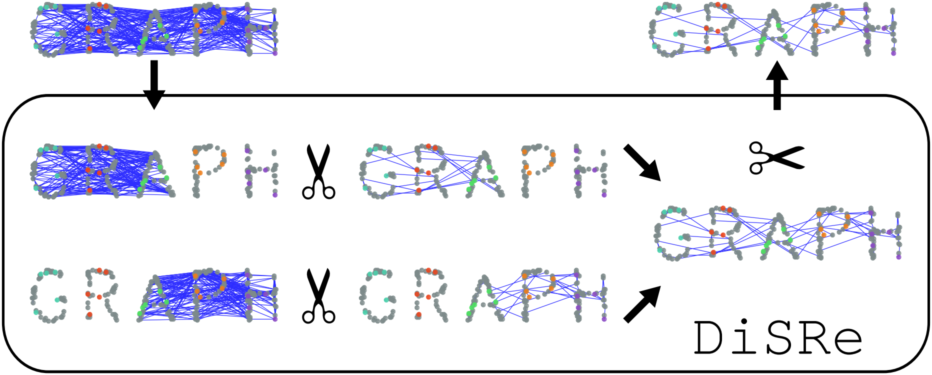
Effective dim.: $\mathbf{d}_{\text{eff}}(\gamma) = \sum_e r_e(\gamma) = \sum_{\mathbf{i}=1}^{\mathbf{n}} \frac{\lambda_\mathbf{i}(\mathbf{L}_\mathcal{G})}{\lambda_\mathbf{i}(\mathbf{L}_\mathcal{G})+\gamma} \leq n$

Can still be computed using fast graph solvers
↳ interpretation as inverse of alternative paths lost

Most existing graph algorithms inapplicable [Kyn+16]
Most existing RLA algorithms too slow [CMM17]

Adapt SOA algorithm for kernel matrix approximation
SQUEAK, [CLV17]

DisRe



arbitrarily split in subgraphs that fit in a single machine

recursively merge-and-reduce until one graph left

↳ additive error cumulates!

↳ merge-and-resparsify

Sparsification

Compute $\widetilde{p}_e^{(1)} \propto \widetilde{r}_e^{(1)}(\gamma)$ using fast graph solver

For each edge e sample with probability $\widetilde{p}_e^{(1)}$

w.h.p. (ε, γ)-accurate and use only $\mathcal{O}(d_{\text{eff}}(\gamma) \log(n)) \leq \mathcal{O}(n \log(n))$ space

Sparsification

Compute $\widetilde{p}_e^{(1)} \propto \widetilde{r}_e^{(1)}(\gamma)$ using fast graph solver

For each edge e sample with probability $\widetilde{p}_e^{(1)}$

w.h.p. (ε, γ)-accurate and use only

$\mathcal{O}(d_{\text{eff}}(\gamma)\log(n)) \leq \mathcal{O}(n\log(n))$ space

Sparsification

Compute $\widetilde{p}_e^{(1)} \propto \widetilde{r}_e^{(1)}(\gamma)$ using fast graph solver

For each edge e sample with probability $\widetilde{p}_e^{(1)}$

w.h.p. (ε, γ)-accurate and use only

$\mathcal{O}(d_{\text{eff}}(\gamma) \log(n)) \leq \mathcal{O}(n \log(n))$ space

Sparsification

Compute $\widetilde{p}_e^{(1)} \propto \widetilde{r}_e^{(1)}(\gamma)$ using fast graph solver

For each edge e sample with probability $\widetilde{p}_e^{(1)}$

w.h.p. (ε, γ)-accurate and use only $\mathcal{O}(d_{\text{eff}}(\gamma)\log(n)) \leq \mathcal{O}(n\log(n))$ space

Merge

Combine sparsifiers, using $2\mathcal{O}(d_{\text{eff}}(\gamma)\log(n))$ space

twice as large as necessary

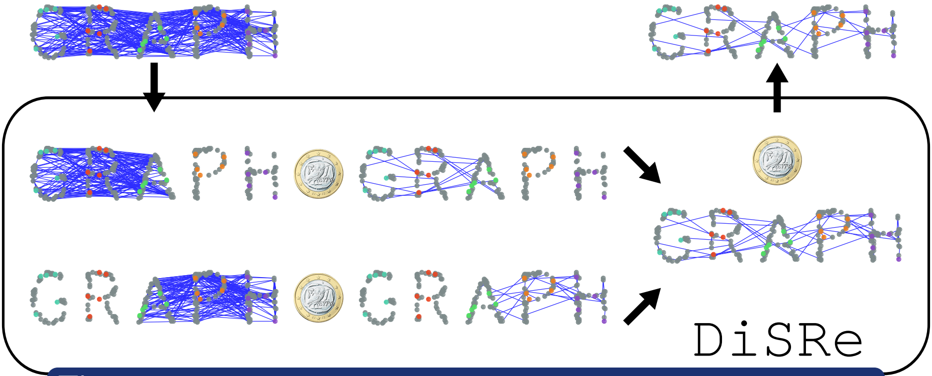
Merge-and-Resparsify

Compute $\widetilde{p}_e^{(2)} \propto \min\{\widetilde{r}_e^{(2)}(\gamma), \widetilde{p}_e^{(1)}\}$ using fast graph solver

For each edge e sample with probability $\widetilde{p}_e^{(2)}/\widetilde{p}_e^{(1)}$

$$\text{survival probability} \quad \frac{\widetilde{p}_e^{(2)}}{\widetilde{p}_e^{(1)}}\,\widetilde{p}_e^{(1)}$$

DisRe guarantees



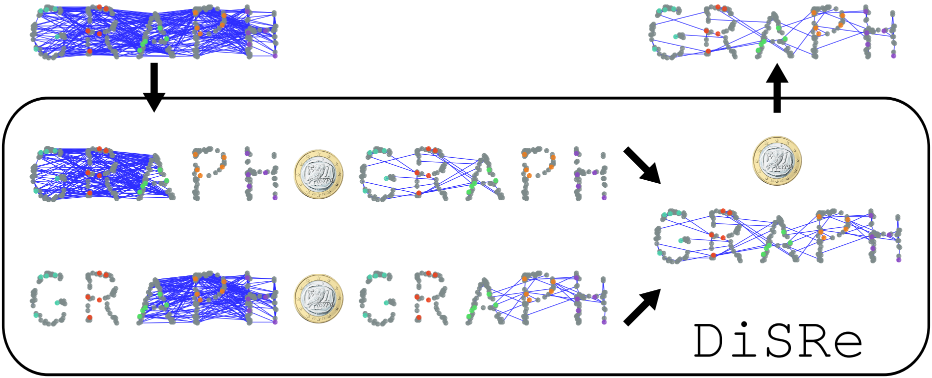
Theorem

Given an arbitrary graph $\mathcal{G}$ w.h.p. DisRe satisfies

(1) each sub-graphs is an (ε, γ)-sparsifier

(2) with at most $\mathcal{O}(d_{\text{eff}}(\gamma)\log(n))$ edges.

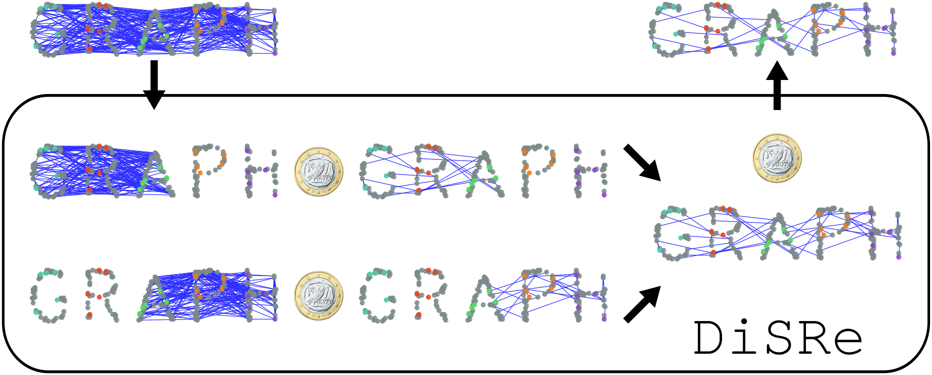
DisRe guarantees



Space: independent from m $\mathcal{O}(d_{\text{eff}}(\gamma)\log(n)) \leq \mathcal{O}(n\log(n))$

Time: $\mathcal{O}(d_{\text{eff}}(\gamma)\log^3(n))$ for fully balanced tree

DisRe guarantees



Communication: only $\mathcal{O}(\log(n))$ rounds

↳ removed edges are forgotten single pass/streaming

↳ point-to-point, centralization only to choose tree

Bibliography I

Ahmed El Alaoui and Michael W. Mahoney. “Fast randomized kernel methods with statistical guarantees”. In: *Neural Information Processing Systems*. 2015.

Daniele Calandriello, Alessandro Lazaric, and Michal Valko. “Distributed adaptive sampling for kernel matrix approximation”. In: *International Conference on Artificial Intelligence and Statistics*. 2017.

Michael B. Cohen, Cameron Musco, and Christopher Musco. “Input sparsity time low-rank approximation via ridge leverage score sampling”. In: *Symposium on Discrete Algorithms*. 2017.

Bibliography II

Rob Fergus, Yair Weiss, and Antonio Torralba. "Semi-Supervised Learning in Gigantic Image Collections". In: *Neural Information Processing Systems*. 2009.

David F Gleich and Michael W Mahoney. "Using Local Spectral Methods to Robustify Graph-Based Learning Algorithms". In: *Knowledge Discovery and Data Mining*. 2015.

Rasmus Kyng et al. "A Framework for Analyzing Resparsification Algorithms". In: *Symposium on Discrete Algorithms*. 2016.

Bibliography III

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. "Graph evolution: Densification and shrinking diameters". In: *Knowledge Discovery from Data* (Mar. 2007).

Daniel A Spielman and Nikhil Srivastava. "Graph sparsification by effective resistances". In: *Journal on Computing* 40.6 (2011).

Veeranjaneyulu Sadhanala, Yu-xiang Wang, and Ryan J Tibshirani. "Graph Sparsification Approaches for Laplacian Smoothing". In: *International Conference on Artificial Intelligence and Statistics*. 2016.

Bibliography IV

Jaewon Yang and Jure Leskovec. "Defining and evaluating network communities based on ground-truth". In: *Knowledge and Information Systems* (2015).

Michal Valko
contact via Piazza