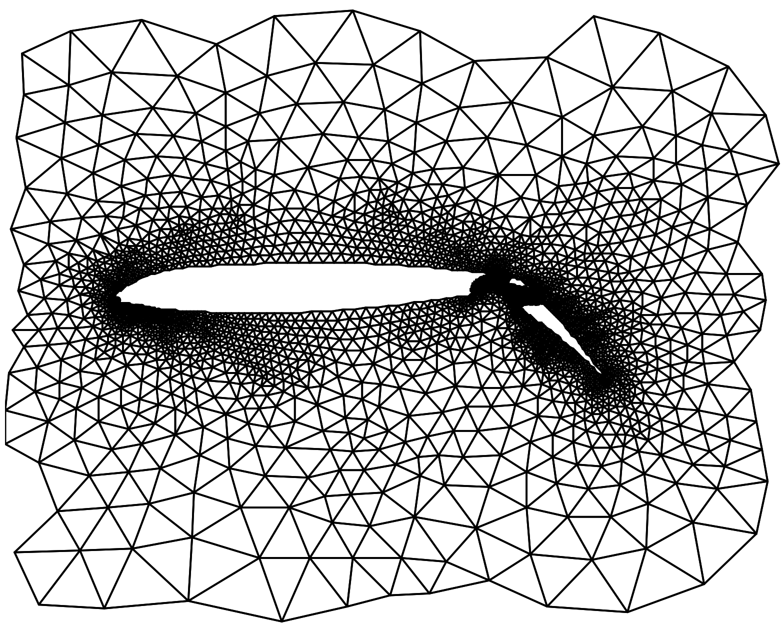
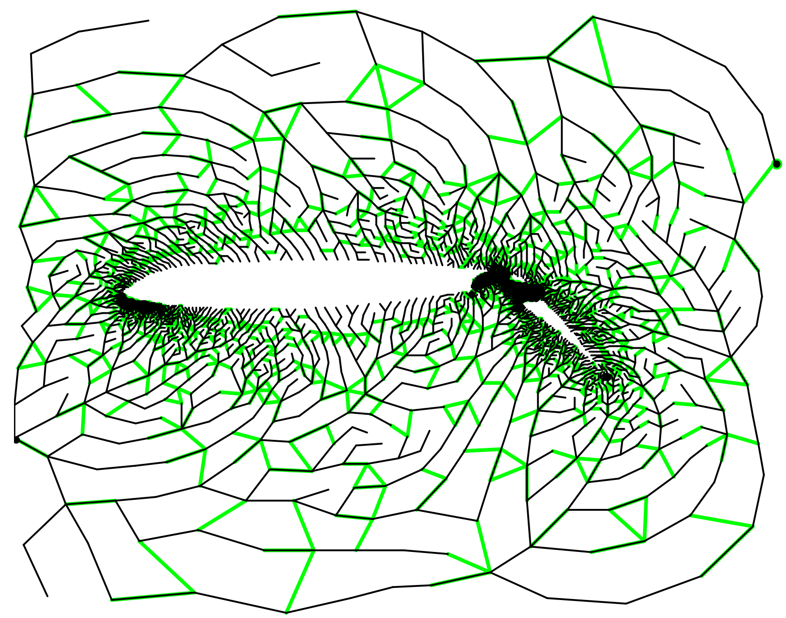
Approximating Graphs
and
Solving Systems of Linear Equations





Daniel A. Spielman
Yale University

BMS, May 13, 2011

Outline

Complexity of solving linear equations $Ax = b$

- Matrix Inversion, Sparse LU, CG

Nearly-linear time for Laplacian Linear Systems

- What
- Sparsification
- Low-stretch spanning trees
- Ultra-sparsification

The future

Matrix Inversion: $x = A^{-1}b$

Can invert an n-by-n matrix in time $O(n^3)$

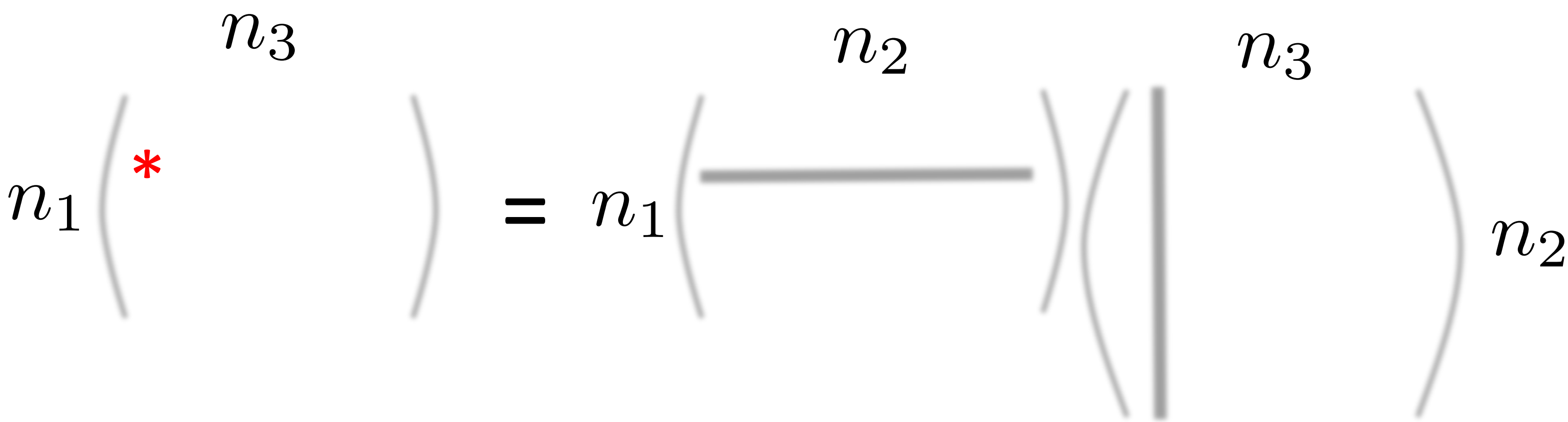
Matrix Inversion: $x = A^{-1}b$

Can invert an n-by-n matrix in time $O(n^3)$

Strassen '69: Can actually do it in time $O(n^{2.81})$

Coppersmith-Winograd '90: in time $O(n^{2.38})$

Matrix Inversion ≈ Matrix Multiplication



Easy to do in time $O(n_1 n_2 n_3)$

If can do it faster, can invert matrices faster.

Group Theoretic Approach [Cohn-Umans ’03]

Can do matrix multiplication in algebra of a group G if G has three sets of elements

$$|S_1| = n_1 \quad |S_2| = n_2 \quad |S_3| = n_3$$

such that for $a_i, b_i \in S_i$

$$a_1 b_1^{-1} a_2 b_2^{-1} a_3 b_3^{-1} = 1 \Longrightarrow a_i b_i^{-1} = 1$$

Fast by discrete Fourier Transform if $\sum_i d_i^3 < n_1 n_2 n_3$

(dimensions of irreducible representations)

Group Theoretic Approach [Cohn-Umans ’03]

Cohn-Kleinberg-Szegedy-Umans ’05
find groups that allow matrix inversion
in time $O(n^{2.41})$

If $\sum_i d_i^3$ small enough,
could invert in time $n^{2+o(1)}$

Group Theoretic Approach [Cohn-Umans '03]

Cohn-Kleinberg-Szegedy-Umans '05
find groups that allow matrix inversion
in time $O(n^{2.41})$

If $\sum_i d_i^3$ small enough,
could invert in time $n^{2+o(1)}$

Can we do it in time $n^{2+o(1)}$?

LU-Factorization (Gaussian Elimination)

Write A = L U, lower- and upper-triangular

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Can be very fast if L and U have few non-zeros.

The inverse usually has $\Omega(n^2)$ non-zeros,
L and U can have $O(n)$ non-zero entries.

Conjugate Gradient for Sparse Systems

[Hestenes '51, Stiefel '52]

Let A have m non-zero entries.

Conjugate Gradient (as a direct method) solves

$$Ax = b$$

in time

$$O(mn)$$

If m is close to n, is much better than inversion!

Laplacian Linear Systems

Solve in time $O(m \log^c m)$

where m = number of non-zeros entries of A

times $\log(1/\epsilon)$ for ϵ-approximate solution.

$$\left\| x - A^{-1}b \right\|_A \leq \epsilon \left\| A^{-1}b \right\|_A$$

Enables solution of all

symmetric, diagonally-dominant systems.

Laplacian Quadratic Form of $G = (V, E)$

For $\boldsymbol{x} : V \to \mathbb{R}$

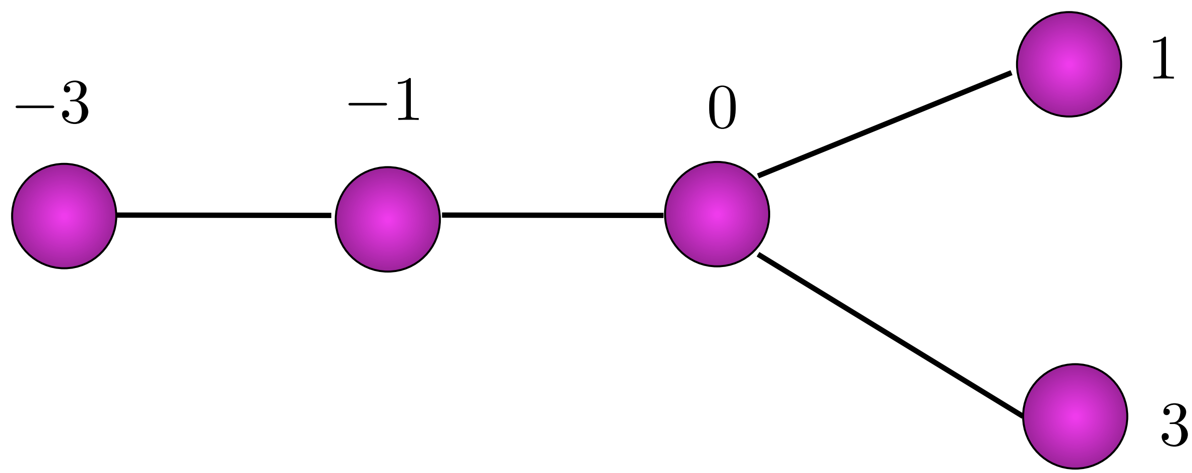
$$\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v)\in E} (\boldsymbol{x}(u) - \boldsymbol{x}(v))^2$$

Laplacian Quadratic Form of $G = (V, E)$

For $\boldsymbol{x} : V \to \mathbb{R}$

$$\boxed{\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v) \in E} \left(\boldsymbol{x}(u) - \boldsymbol{x}(v) \right)^2}$$

$\boldsymbol{x}$:

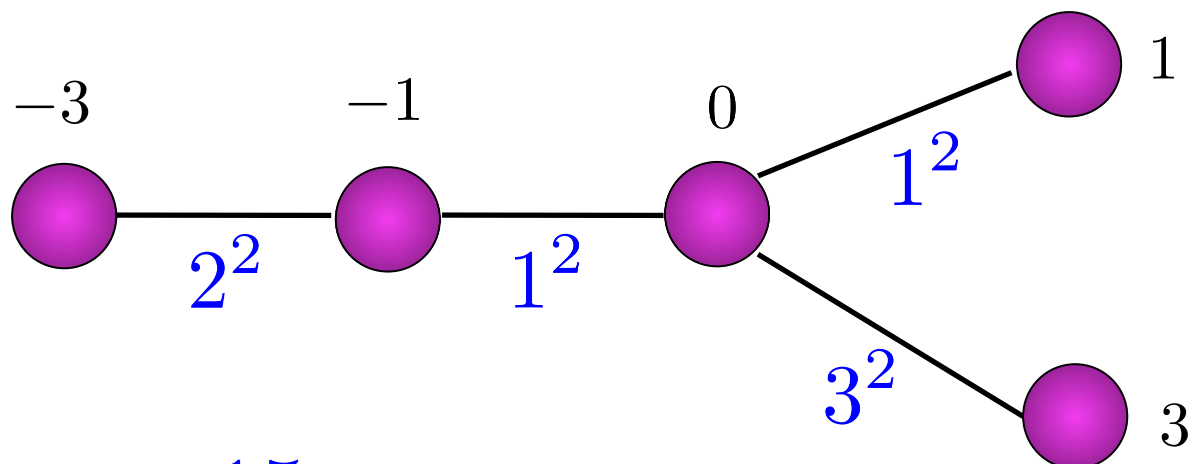


Laplacian Quadratic Form of $G = (V, E)$

For $\boldsymbol{x} : V \to \mathbb{R}$

$$\boxed{\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v) \in E} (\boldsymbol{x}(u) - \boldsymbol{x}(v))^2}$$

$\boldsymbol{x}$:



$\boldsymbol{x}^T L_G \boldsymbol{x} = 15$

Laplacian Quadratic Form for Weighted Graphs

$$G = (V, E, w)$$

$w : E \to \mathbb{R}^+$ assigns a positive weight to every edge

$$\boxed{\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v) \in E} w_{(u,v)} \left(\boldsymbol{x}(u) - \boldsymbol{x}(v) \right)^2}$$

Matrix L_G is positive semi-definite

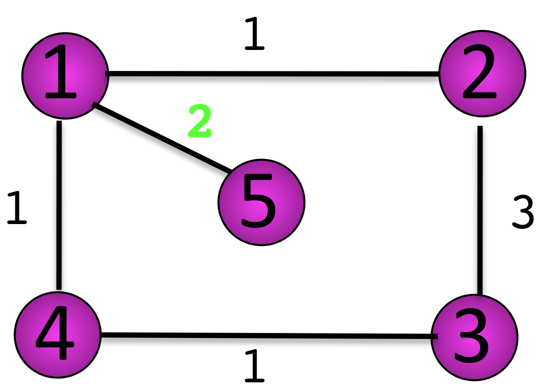
nullspace spanned by const vector, if connected

Laplacian Matrix of a Weighted Graph

$$L_G(u,v) = \begin{cases} -w(u,v) & \text{if } (u,v) \in E \\ d(u) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$$

$$d(u) = \sum_{(v,u) \in E} w(u,v)$$

the weighted degree of u



$$\begin{matrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 4 & -3 & 0 & 0 \\ 0 & -3 & 4 & -1 & 0 \\ -1 & 0 & -1 & 2 & 0 \\ -2 & 0 & 0 & 0 & 2 \end{matrix}$$

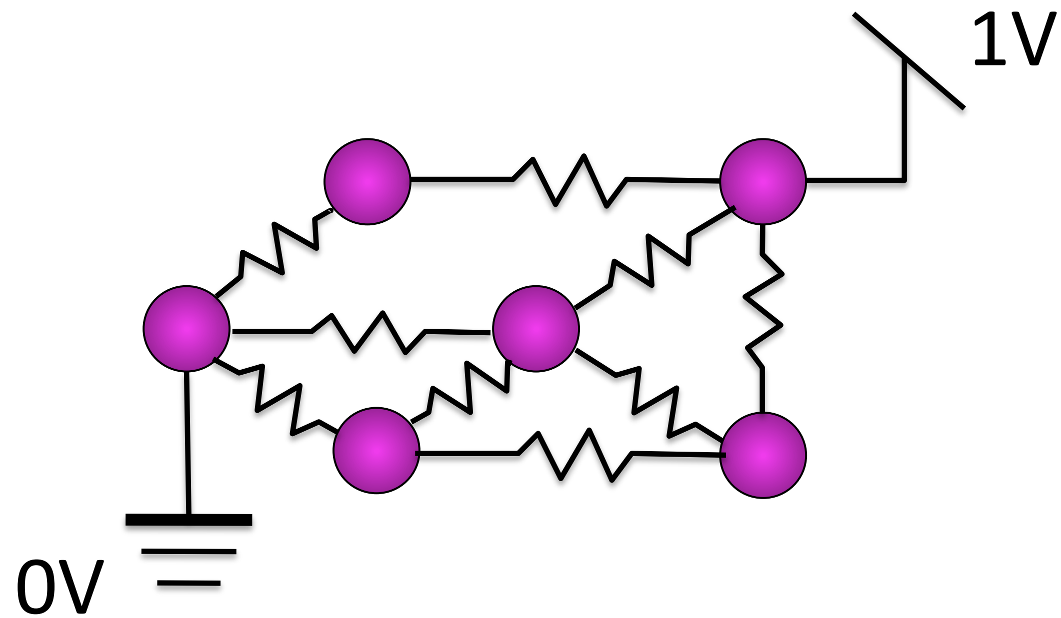
is a diagonally dominant matrix

Networks of Resistors [Kirchhoff]

Ohm's laws gives $i = v/r$

In general, $\boldsymbol{i} = L_G \boldsymbol{v}$ with $w_{(u,v)} = 1/r_{(u,v)}$

Minimize dissipated energy $\boldsymbol{v}^T L_G \boldsymbol{v}$



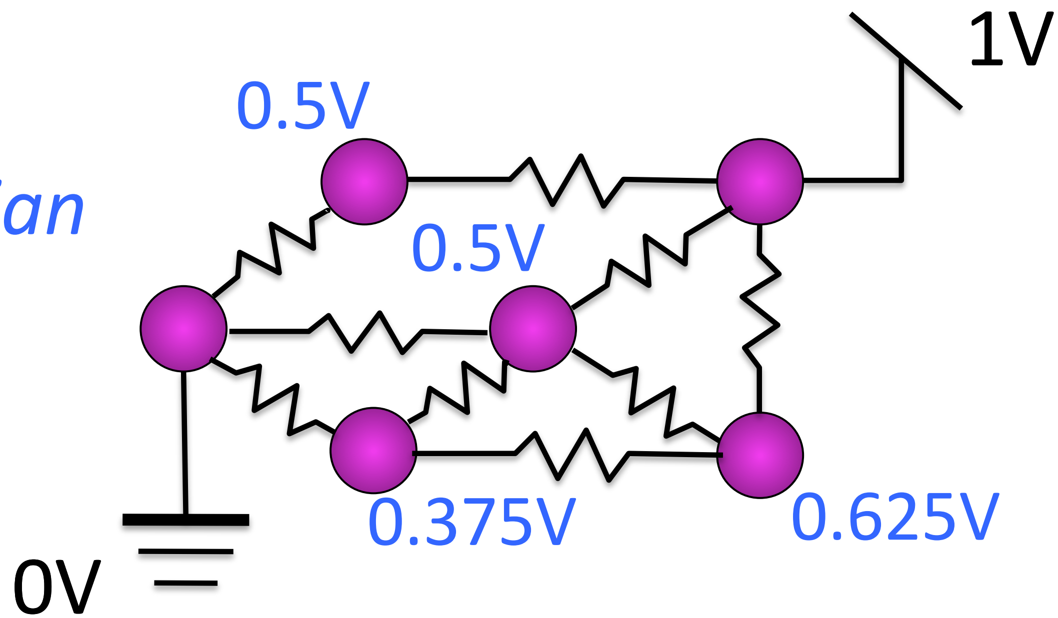
Networks of Resistors

Ohm's laws gives $i = v/r$

In general, $\boldsymbol{i} = L_G \boldsymbol{v}$ with $w_{(u,v)} = 1/r_{(u,v)}$

Minimize dissipated energy $\boldsymbol{v}^T L_G \boldsymbol{v}$

By solving Laplacian

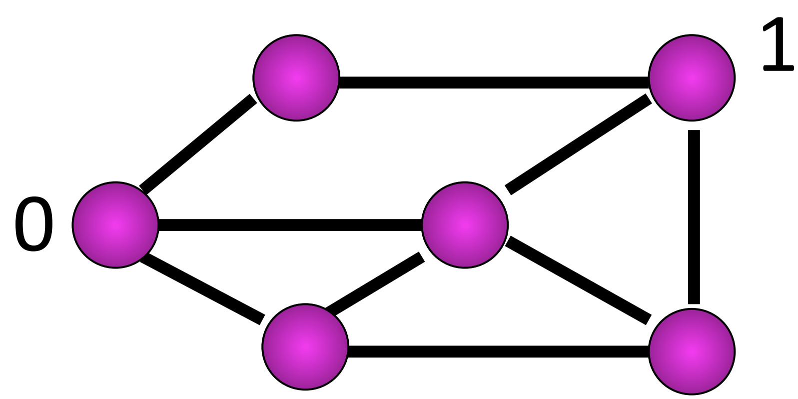


Learning on Graphs [Zhu-Ghahramani-Lafferty '03]

Infer values of a function at all vertices
from known values at a few vertices.

Minimize $$\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v)\in E} w_{(u,v)} \left(\boldsymbol{x}(u) - \boldsymbol{x}(v)\right)^2$$

Subject to known values

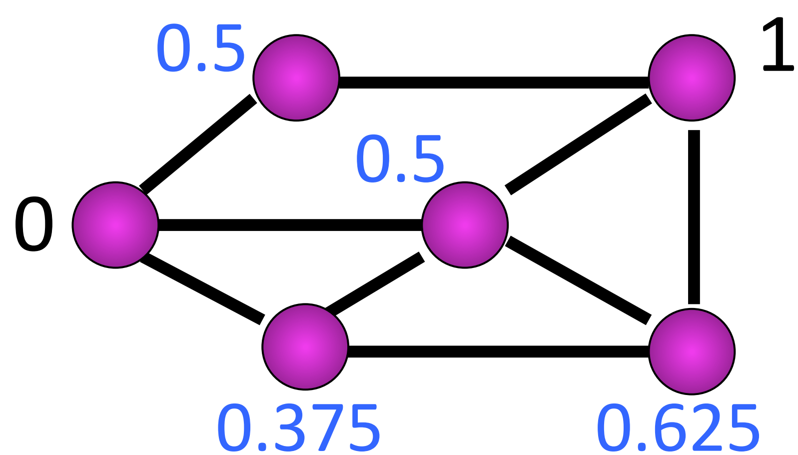


Learning on Graphs [Zhu-Ghahramani-Lafferty '03]

Infer values of a function at all vertices
from known values at a few vertices.

Minimize $\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v) \in E} w_{(u,v)} \left(\boldsymbol{x}(u) - \boldsymbol{x}(v)\right)^2$

Subject to known values



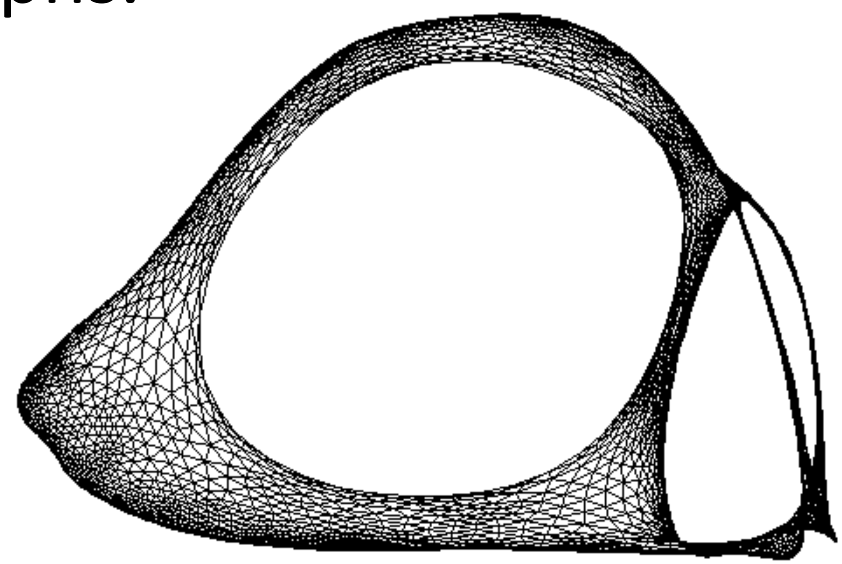
By solving Laplacian

Other Applications

Solving Elliptic PDEs.

Solving Maximum Flow Problems.

Computing Eigenvectors and Eigenvalues of Laplacians of graphs.



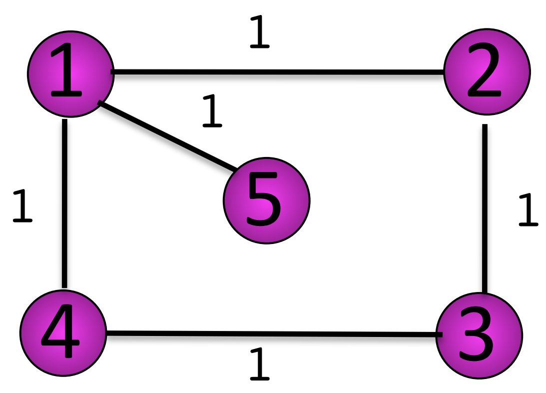
Solving Laplacian Linear Equations Quickly

Fast when graph is simple,
by elimination.

Fast approximation when graph is complicated*,
by Conjugate Gradient

* = random graph or expander

Cholesky Factorization of Laplacians

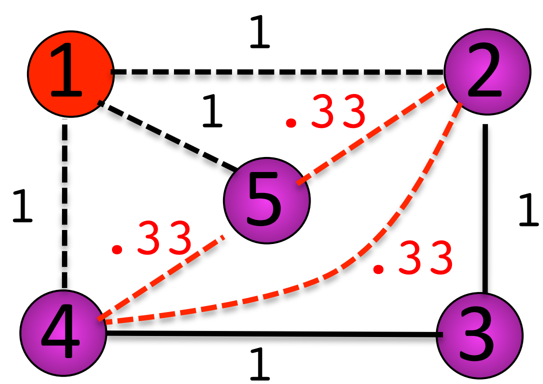


$$
\begin{matrix}
3 & -1 & 0 & -1 & -1 \\
-1 & 2 & -1 & 0 & 0 \\
0 & -1 & 2 & -1 & 0 \\
-1 & 0 & -1 & 2 & 0 \\
-1 & 0 & 0 & 0 & 1
\end{matrix}
$$

When eliminate a vertex,
connect its neighbors.

Also known as Y-Δ

Cholesky Factorization of Laplacians



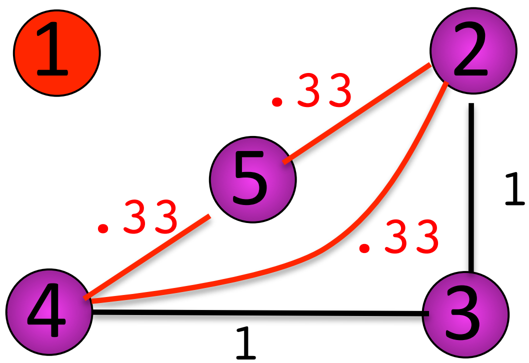
3	-1	0	-1	-1
-1	2	-1	0	0
0	-1	2	-1	0
-1	0	-1	2	0
-1	0	0	0	1

When eliminate a vertex, connect its neighbors.

3	0	0	0	0
0	1.67	-1.00	-0.33	-0.33
0	-1.00	2.00	-1.00	0
0	-0.33	-1.00	1.67	-0.33
0	-0.33	0	-0.33	0.67

Also known as Y-Δ

Cholesky Factorization of Laplacians



3	-1	0	-1	-1
-1	2	-1	0	0
0	-1	2	-1	0
-1	0	-1	2	0
-1	0	0	0	1

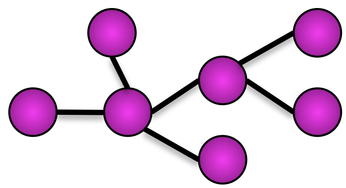
3	0	0	0	0
0	1.67	-1.00	-0.33	-0.33
0	-1.00	2.00	-1.00	0
0	-0.33	-1.00	1.67	-0.33
0	-0.33	0	-0.33	0.67

When eliminate a vertex, connect its neighbors.

Also known as Y-Δ

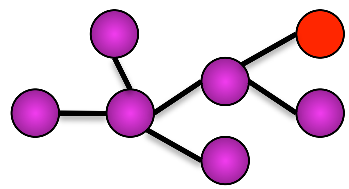
Complexity of Cholesky Factorization

#ops ~ Σ_v (degree of v when eliminate)^2

Tree  #ops ~ O(|V|)

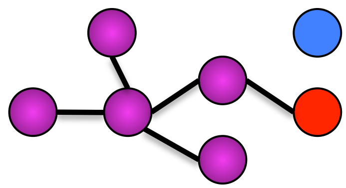
Complexity of Cholesky Factorization

#ops ~ Σ_v (degree of v when eliminate)^2

Tree  #ops ~ O(|V|)

Complexity of Cholesky Factorization

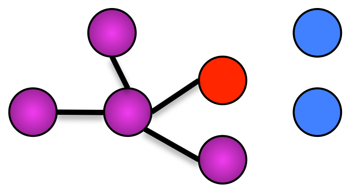
#ops ~ Σ_v (degree of v when eliminate)^2

Tree  #ops ~ O(|V|)

Complexity of Cholesky Factorization

#ops ~ Σ_v (degree of v when eliminate)^2

Tree

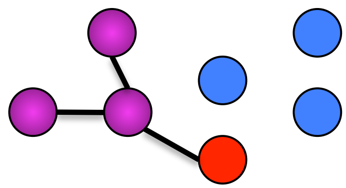


#ops ~ O(|V|)

Complexity of Cholesky Factorization

#ops ~ Σ_v (degree of v when eliminate)^2

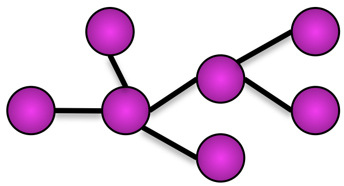
Tree



#ops ~ $O(|V|)$

Complexity of Cholesky Factorization

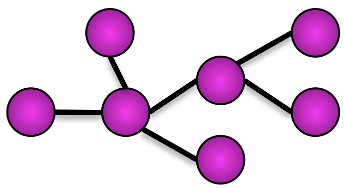
#ops ~ Σ_v (degree of v when eliminate)^2

Tree  #ops ~ O(|V|)

Complexity of Cholesky Factorization

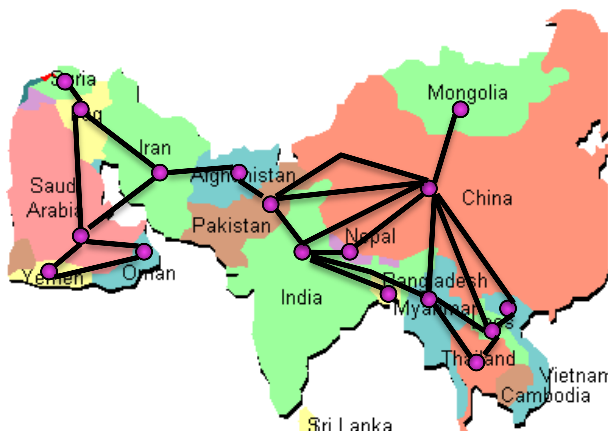
#ops ~ Σ_v (degree of v when eliminate)^2

Tree



#ops ~ O(|V|)

Planar



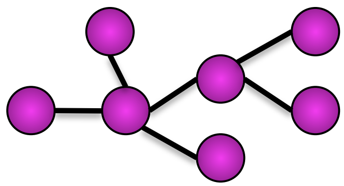
#ops ~ O(|V|$^{3/2}$)

Lipton-Rose-Tarjan '79

Complexity of Cholesky Factorization

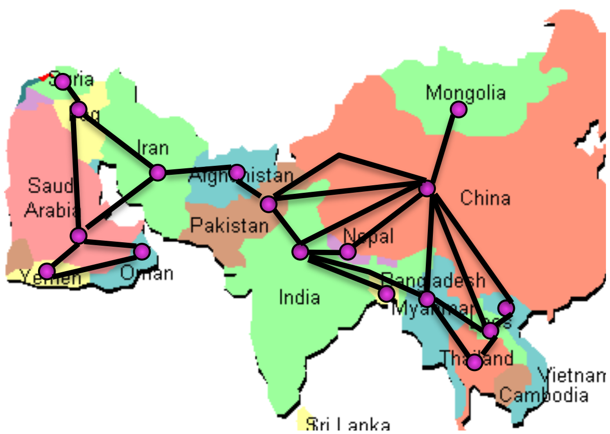
#ops ~ Σ_v (degree of v when eliminate)^2

Tree



#ops ~ O(|V|)

Planar



#ops ~ O(|V|$^{3/2}$)

Lipton-Rose-Tarjan '79

Expander

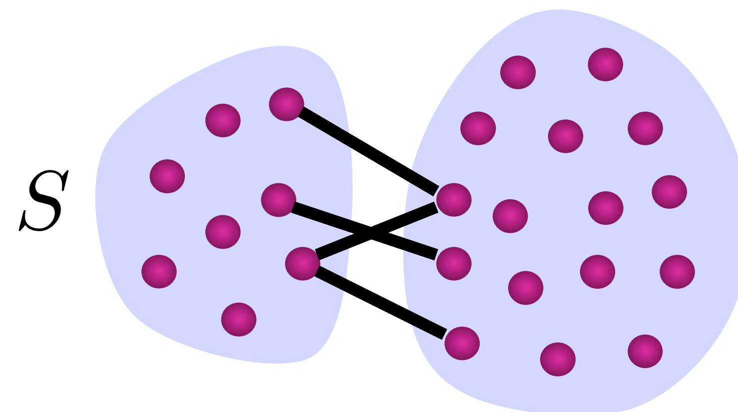
like random,
but O(|V|) edges

#ops $\gtrsim$ Ω(|V|^3)

Lipton-Rose-Tarjan '79

Conductance and Cholesky Factorization

For $S \subset V$



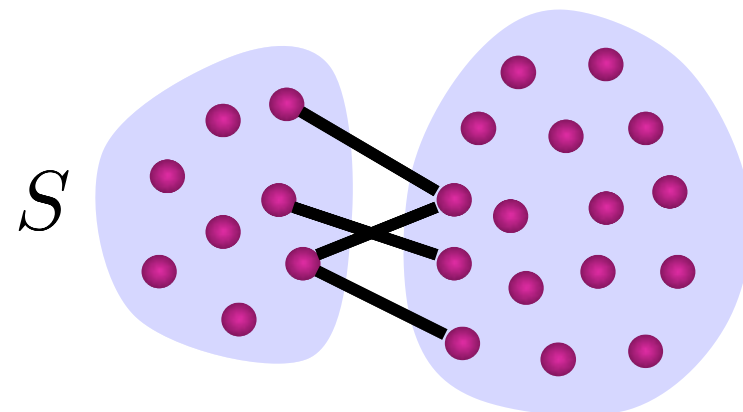
$$\Phi(S) = \frac{\#\text{ edges leaving } S}{\text{sum degrees on smaller side, } S \text{ or } V - S}$$

$$\boxed{\Phi_G = \min_{S \subset V} \Phi(S)}$$

Conductance and Cholesky Factorization

Cholesky slow when conductance high
Cholesky fast when low for G and all subgraphs

For $S \subset V$

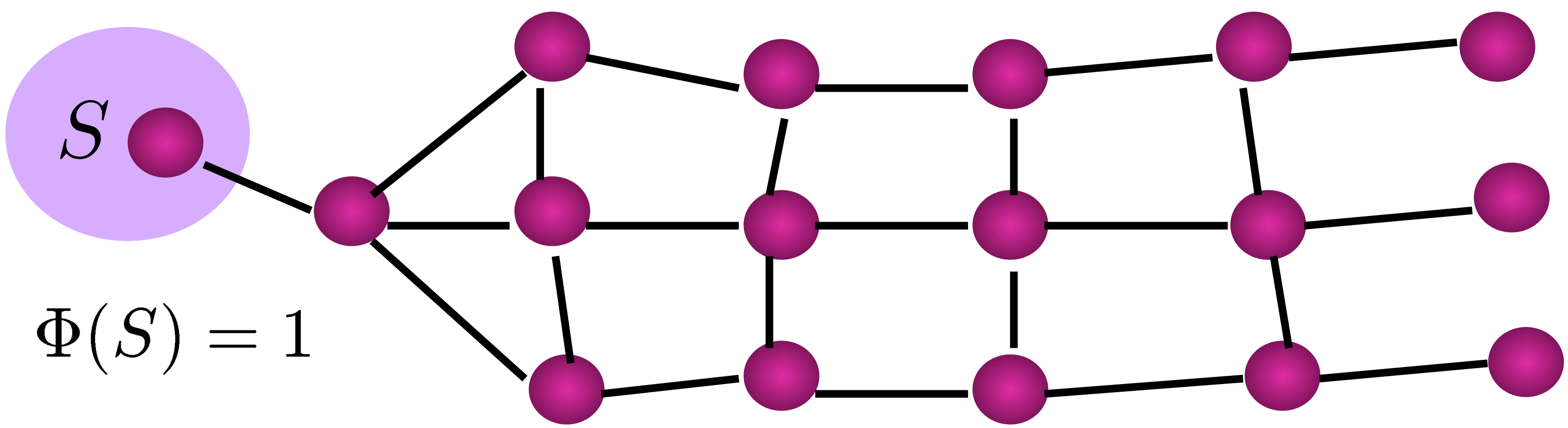


$$\Phi(S) = \frac{\#\ \text{edges leaving } S}{\text{sum degrees on smaller side, } S \text{ or } V - S}$$

$$\boxed{\Phi_G = \min_{S \subset V} \Phi(S)}$$

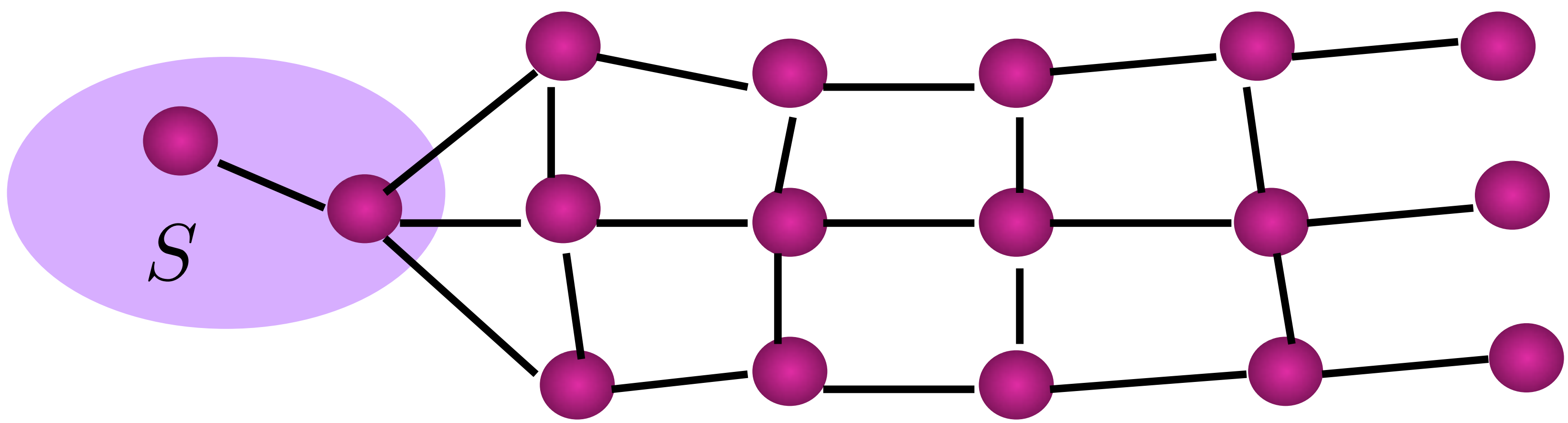
Conductance

$$\Phi(S) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } S}{\text{sum of degrees on smaller side}}$$



Conductance

$$\Phi(S) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } S}{\text{sum of degrees on smaller side}}$$

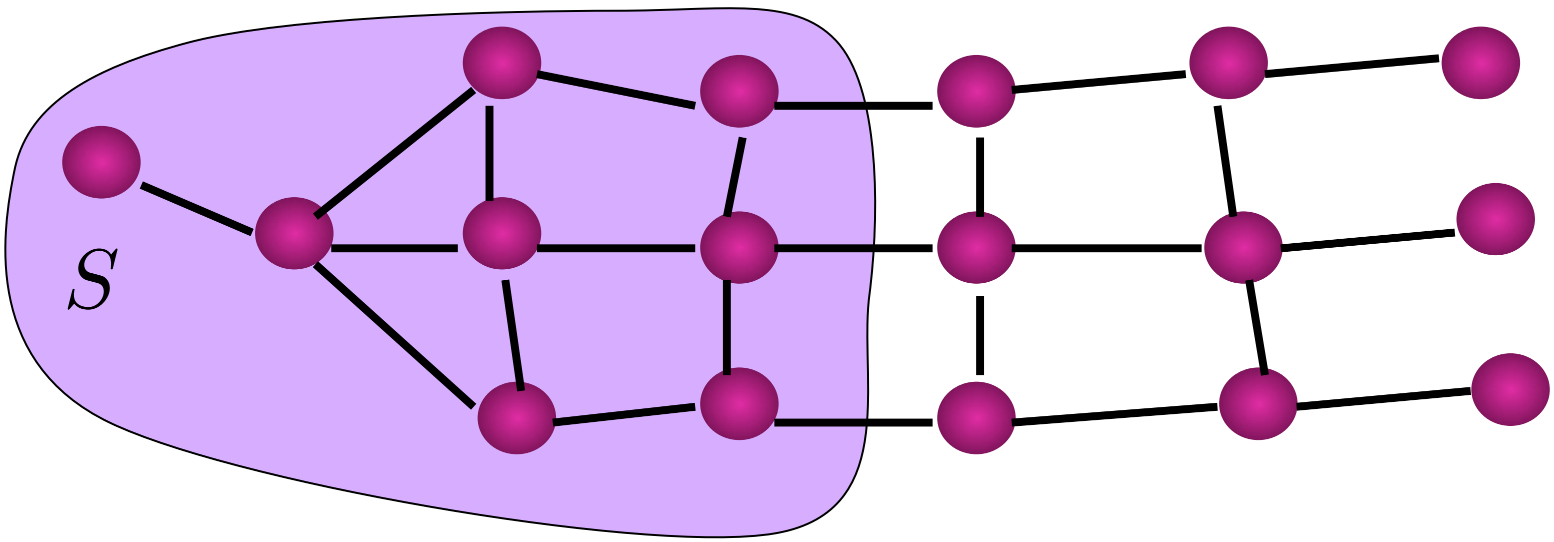


$$\Phi(S) = 3/5$$

Conductance

$$\Phi(S) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } S}{\text{sum of degrees on smaller side}}$$

$$\Phi_G \stackrel{\text{def}}{=} \min_S \Phi(S)$$



$$\Phi(S) = 3/\min(25, 23) = \Phi_G$$

Cheeger's Inequality and the Conjugate Gradient

Cheeger's inequality (degree-d unweighted case)

$$\boxed{\frac{1}{2}\frac{\lambda_2}{d} \leq \Phi_G \leq \sqrt{2\frac{\lambda_2}{d}}}$$

λ_2 = second-smallest eigenvalue of L_G
~ d/mixing time of random walk

near d for expanders and random graphs

Cheeger's Inequality and the Conjugate Gradient

Cheeger's inequality (degree-d unweighted case)

$$\boxed{\frac{1}{2}\frac{\lambda_2}{d} \leq \Phi_G \leq \sqrt{2\frac{\lambda_2}{d}}}$$

λ_2 = second-smallest eigenvalue of L_G
~ d/mixing time of random walk

Conjugate Gradient finds ϵ-approx solution to $L_G x = b$

in $O(\sqrt{d/\lambda_2}\log \epsilon^{-1})$ mults by L_G

is $O(m\Phi_G^{-1}\log \epsilon^{-1})$ ops

Fast solution of linear equations

Conjugate Gradient fast when conductance high.

Elimination fast when low for G and all subgraphs.

Fast solution of linear equations

Conjugate Gradient fast when conductance high.

Planar graphs

Elimination fast when low for G and all subgraphs.

Problems:

Want speed of extremes in the middle

Fast solution of linear equations

Conjugate Gradient fast when conductance high.

Planar graphs

Elimination fast when low for G and all subgraphs.

Problems:

Want speed of extremes in the middle

Not all graphs fit into these categories!

Preconditioned Conjugate Gradient

Solve $L_G x = b$ by

Approximating L_G by L_H (the preconditioner)

In each iteration
solve a system in L_H
multiply a vector by L_G

ϵ-approximate solution after
$O(\sqrt{\kappa(L_G, L_H)} \log \epsilon^{-1})$ iterations

condition number/approx quality

Inequalities and Approximation

$L_H \preccurlyeq L_G$ if $L_G - L_H$ is positive semi-definite,
i.e. for all x,

$$x^T L_H x \preccurlyeq x^T L_G x$$

Example: if H is a subgraph of G

$$\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{(u,v) \in E} w_{(u,v)} \left(\boldsymbol{x}(u) - \boldsymbol{x}(v) \right)^2$$

Inequalities and Approximation

$L_H \preccurlyeq L_G$ if $L_G - L_H$ is positive semi-definite,
i.e. for all x,

$$x^T L_H x \preccurlyeq x^T L_G x$$

$\kappa(L_G, L_H) \leq t$

if $\quad L_H \preccurlyeq L_G \preccurlyeq tL_H$

iff $\; cL_H \preccurlyeq L_G \preccurlyeq ctL_H$ for some c

Inequalities and Approximation

$L_H \preccurlyeq L_G$ if $L_G - L_H$ is positive semi-definite,
i.e. for all x,

$$x^T L_H x \preccurlyeq x^T L_G x$$

$\kappa(L_G, L_H) \leq t$

if $\quad L_H \preccurlyeq L_G \preccurlyeq t L_H$

iff $\; c L_H \preccurlyeq L_G \preccurlyeq ct L_H$ for some c

Call H a t-approx of G if $\kappa(L_G, L_H) \leq t$

Other definitions of relative condition number

$$\kappa(L_G, L_H) = \frac{\lambda_{max}(L_G L_H^+)}{\lambda_{min}(L_G L_H^+)}$$

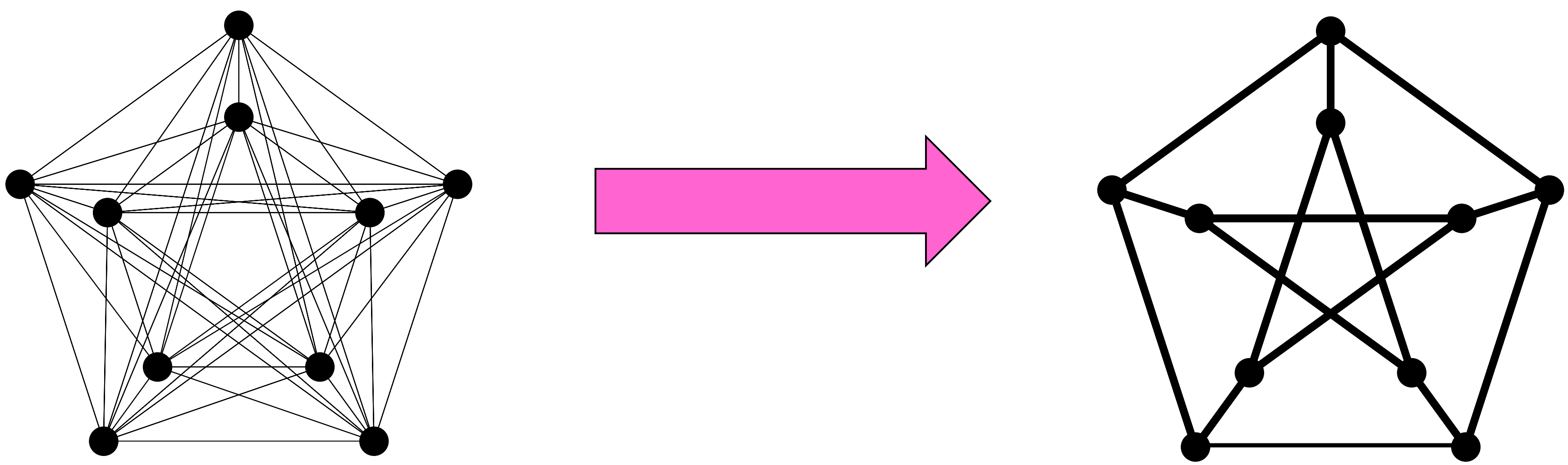
pseudo-inverse

min non-zero eigenvalue

Spectral Sparsification of Graphs [S-Teng]

For every graph G with n vertices
there is a sparse graph H such that

$$\kappa(L_G, L_H) \leq 1 + \epsilon$$



Spectral Sparsification of Graphs [S-Teng]

For every graph G with n vertices
there is a sparse graph H such that

$$\kappa(L_G, L_H) \leq 1 + \epsilon$$

Sparse: exists H with $4n/\epsilon^2$ edges
[Batson-S-Srivastava]

Can find H with $O(n \log n/\epsilon^2)$ edges
in nearly-linear time. [S-Srivastava]

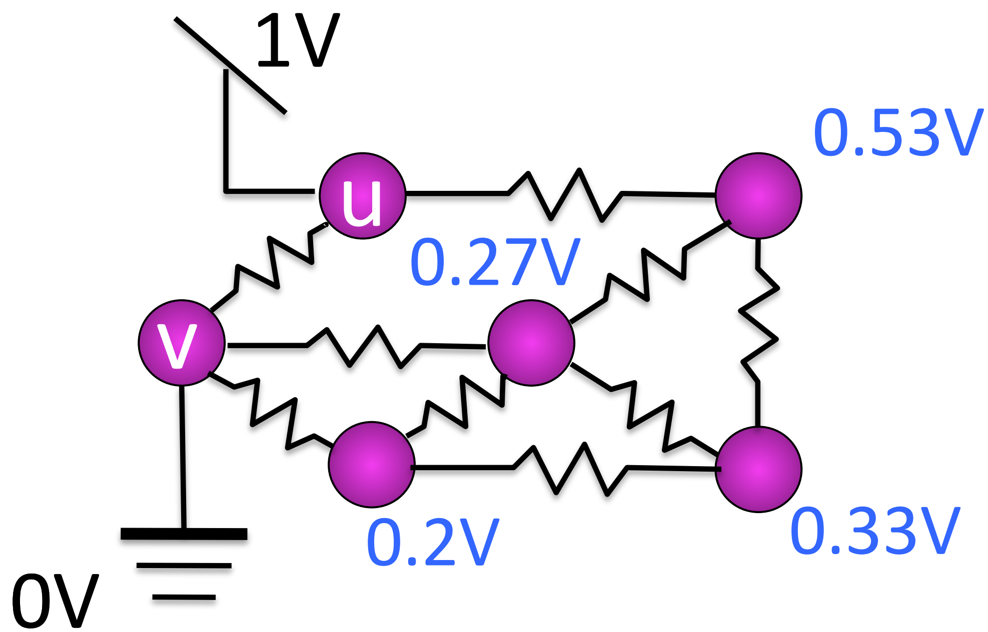
Sparsification by Random Sampling [S-Srivastava]

Include edge (u, v) with probability

$$p_{u,v} \sim w_{u,v} R_{\text{eff}}(u, v)$$

$R_{\text{eff}}(u, v)$ = effective resistance between u and v

= 1/(current flow at one volt)



Sparsification by Random Sampling [S-Srivastava]

Include edge (u, v) with probability

$$p_{u,v} \sim w_{u,v} R_{\mathrm{eff}}(u, v)$$

If include edge, give weight $w_{u,v}/p_{u,v}$

Can do all this in time $O(n \log^3 n)$

Spectral Sparsification of Graphs [S-Teng]

For every graph G with n vertices
there is a sparse graph H such that

$$\kappa(L_G, L_H) \leq 1 + \epsilon$$

Can solve $L_G x = b$ in time

$$O(n^2 \log n \log \epsilon^{-1})$$

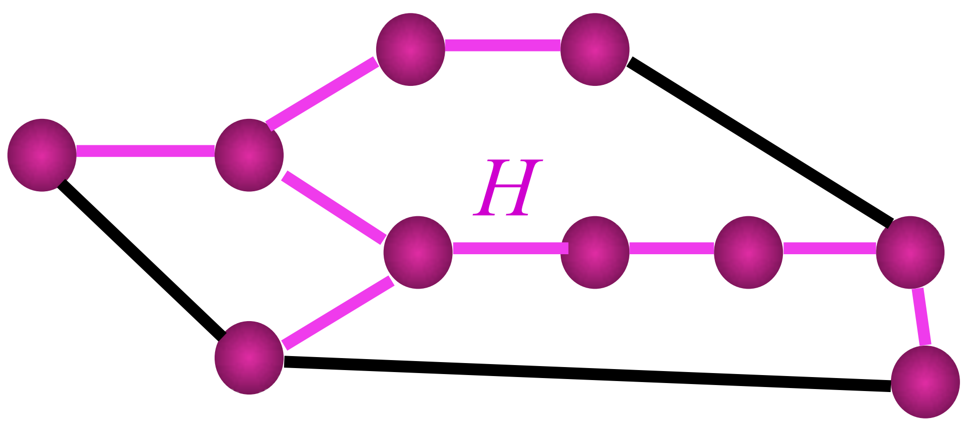
Using CG as direct solver for L_H

Vaidya's Subgraph Preconditioners

Precondition G by a subgraph H

$L_H \preccurlyeq L_G$ so just need t for which $L_G \preccurlyeq t L_H$

Easy to bound t if H is a spanning tree

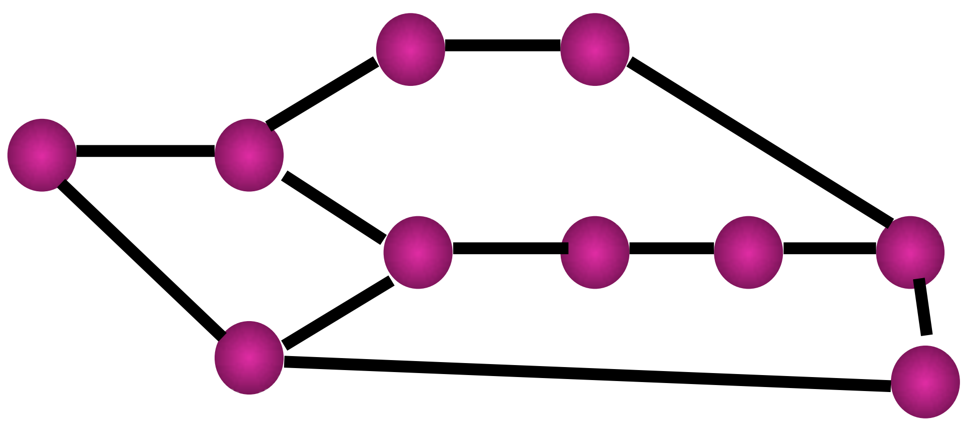


And, easy to solve equations in L_H by elimination

The Stretch of Spanning Trees

Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

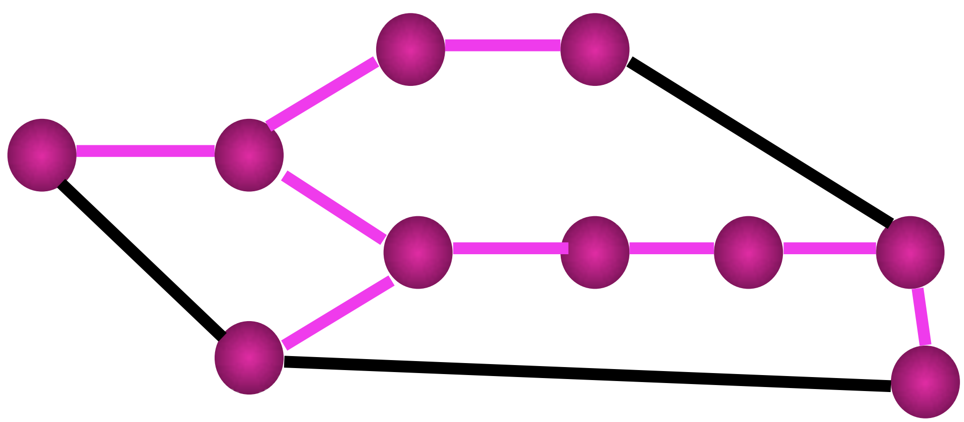
Where $\mathrm{st}_T(G) = \sum_{(u,v) \in E} \text{path-length}_T(u,v)$



The Stretch of Spanning Trees

Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

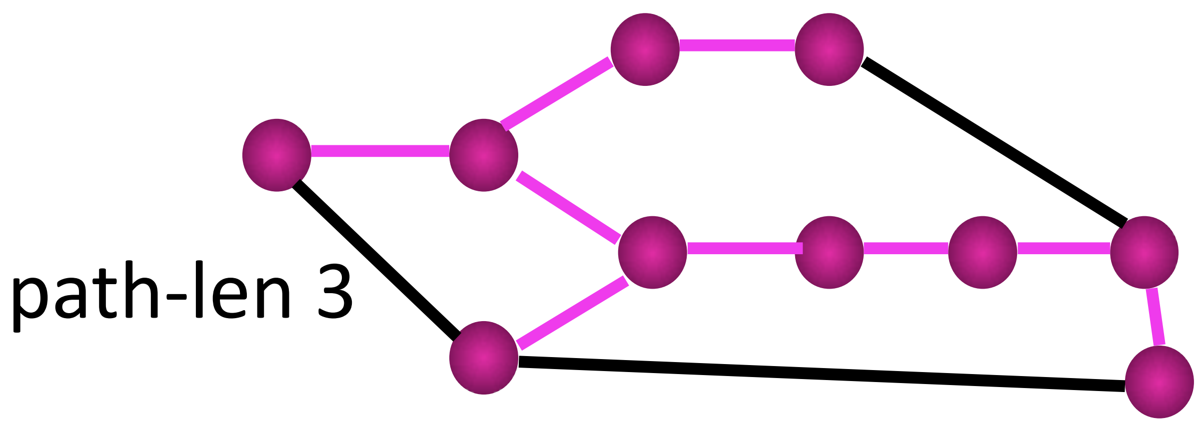
Where $\mathrm{st}_T(G) = \sum_{(u,v)\in E} \text{path-length}_T(u,v)$



The Stretch of Spanning Trees

Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

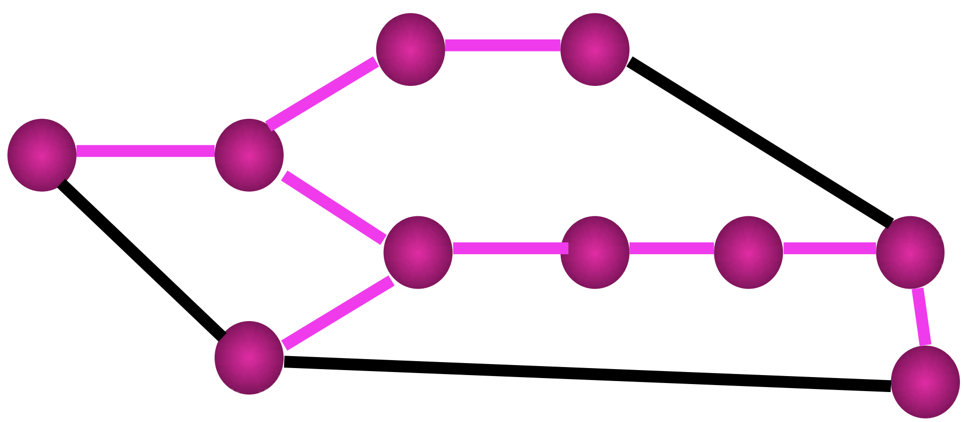
Where $\mathrm{st}_T(G) = \sum_{(u,v) \in E} \text{path-length}_T(u,v)$



The Stretch of Spanning Trees

Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

Where $\mathrm{st}_T(G) = \sum_{(u,v)\in E} \text{path-length}_T(u,v)$

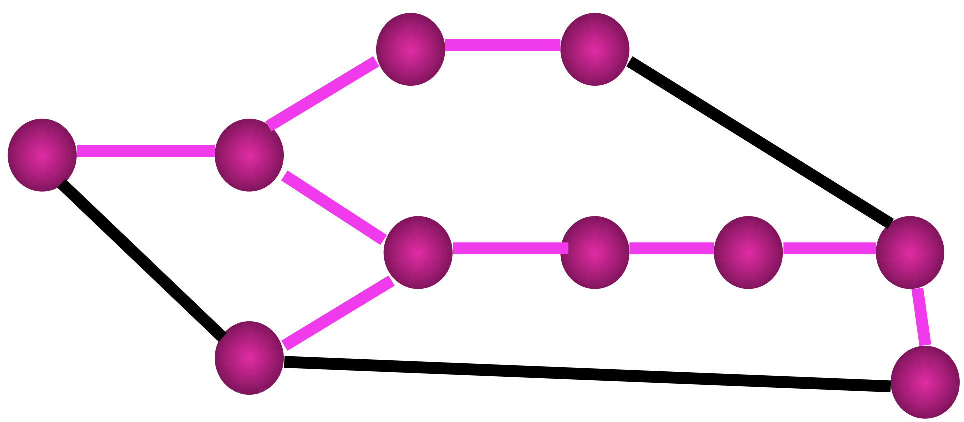


path-len 5

The Stretch of Spanning Trees

Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

Where $\mathrm{st}_T(G) = \sum_{(u,v)\in E} \text{path-length}_T(u,v)$

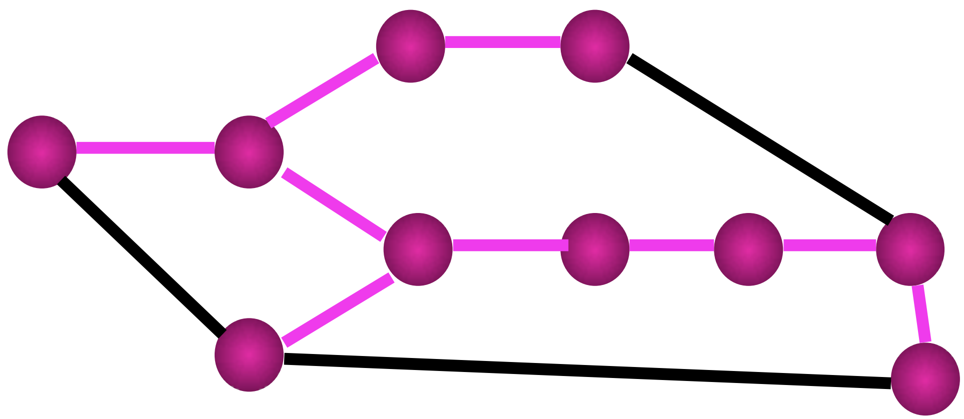


path-len 1

The Stretch of Spanning Trees

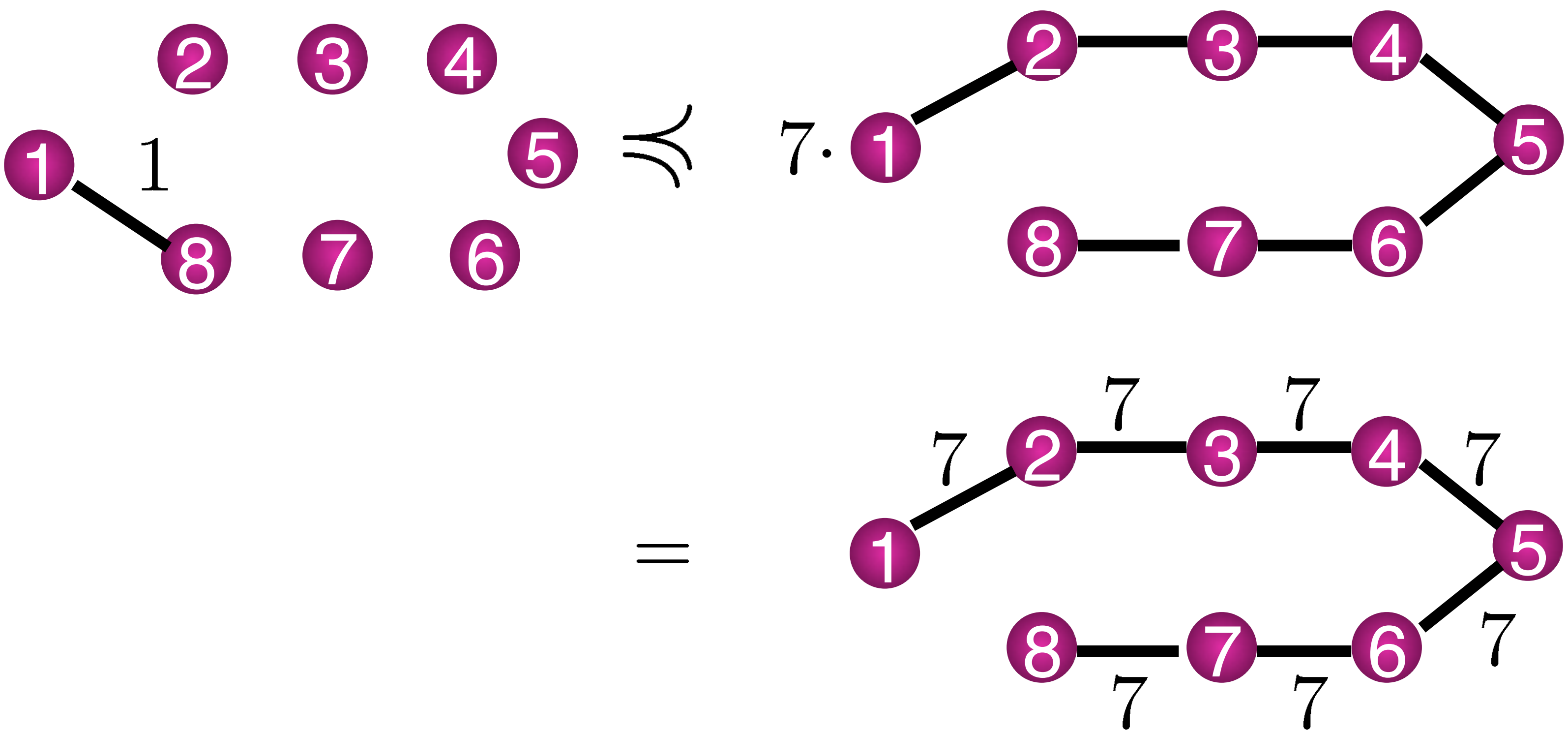
Boman-Hendrickson '01: $L_G \preccurlyeq \mathrm{st}_G(T) L_T$

Where $\mathrm{st}_T(G) = \sum_{(u,v)\in E} \text{path-length}_T(u,v)$



In weighted case, measure resistances of paths

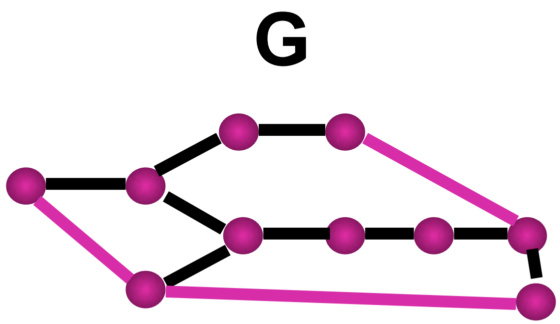
Fundamental Graphic Inequality



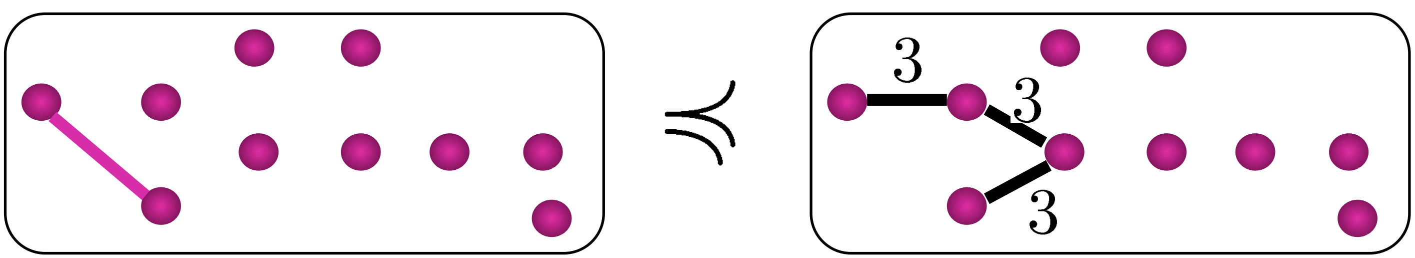
edge $\preccurlyeq$ k times path of length k

With weights, corresponds to resistors in serial
(Poincaré inequality)

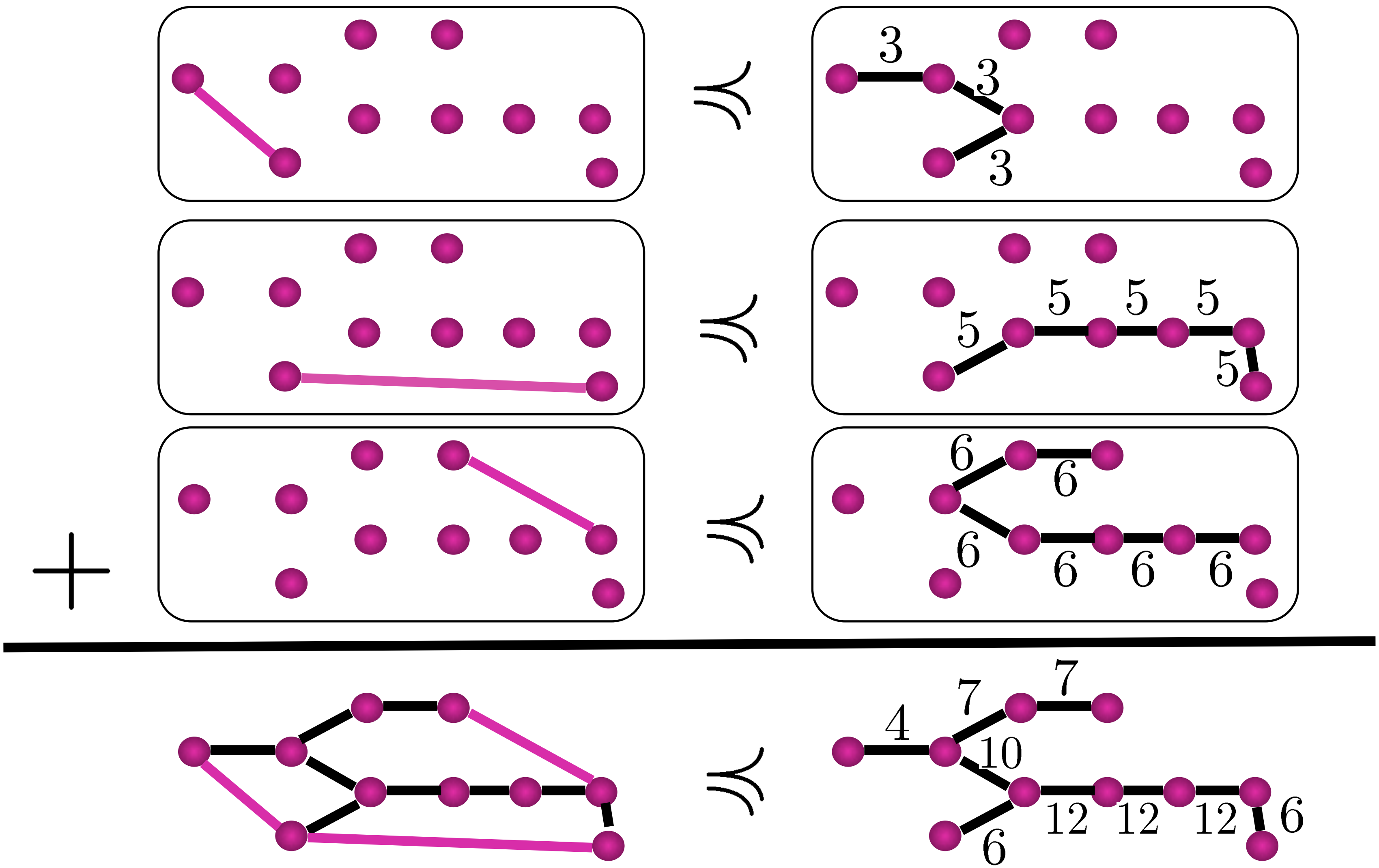
When T is a Spanning Tree



Every edge of G not in T has unique path in T



When T is a Spanning Tree



Low-Stretch Spanning Trees

For every G there is a T with

$$\mathrm{st}_T(G) \leq m^{1+o(1)}$$ where $m = |E|$

(Alon-Karp-Peleg-West '91)

$$\mathrm{st}_T(G) \leq O(m \log m \log^2 \log m)$$

(Elkin-Emek-S-Teng '04, Abraham-Bartal-Neiman '08)

Solve linear systems in time $O(m^{3/2} \log m)$

Low-Stretch Spanning Trees

For every G there is a T with

$$\mathrm{st}_T(G) \leq m^{1+o(1)}$$ where $m = |E|$

(Alon-Karp-Peleg-West '91)

$$\mathrm{st}_T(G) \leq O(m \log m \log^2 \log m)$$

(Elkin-Emek-S-Teng '04, Abraham-Bartal-Neiman '08)

Solve linear systems in time $O(m^{3/2} \log m)$

With sparsification, $O(m + n^{3/2} \log n)$

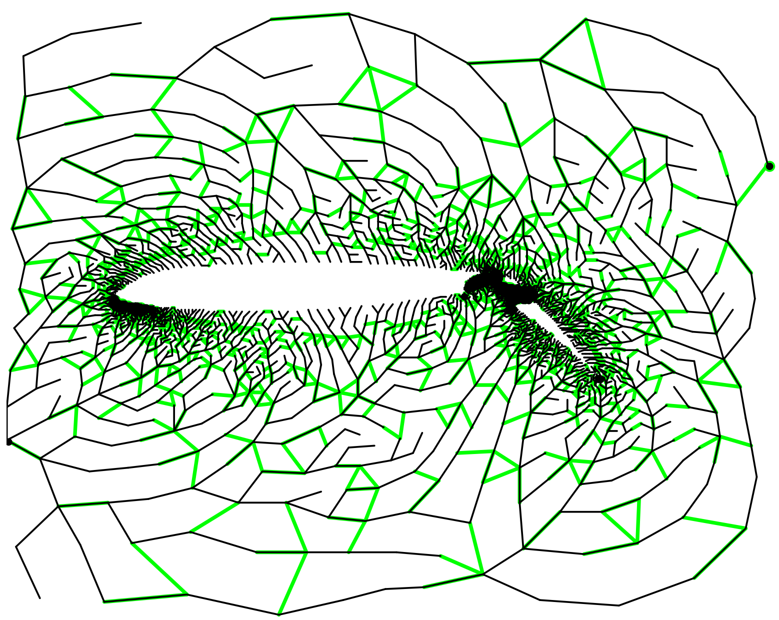
Sparsifiers

Low-Stretch Trees

Ultra-Sparsifiers [S-Teng]

Approximate G by a tree plus $n/\log^2 n$ edges

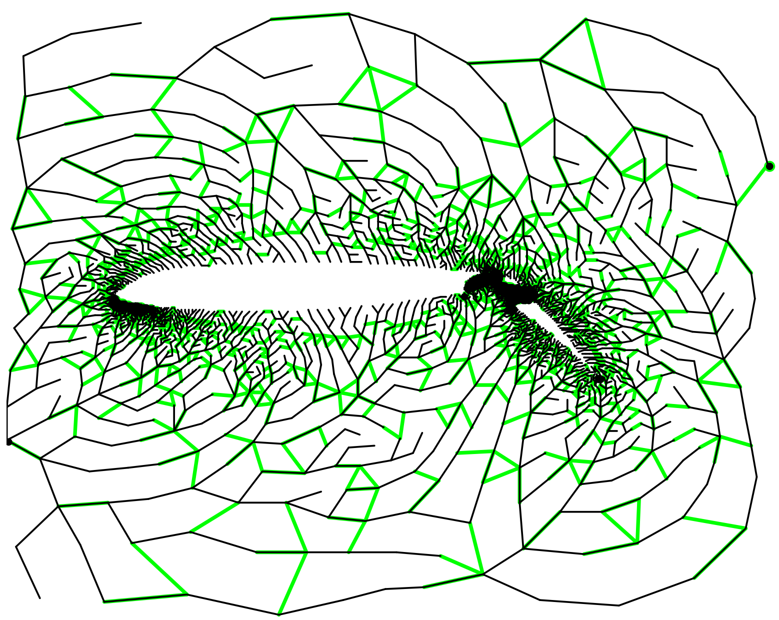
$$L_H \preccurlyeq L_G \preccurlyeq c\log^2 n \; L_H$$



Ultra-Sparsifiers

Solve systems in H by:

1. Cholesky eliminating degree 1 and 2 nodes
2. recursively solving reduced system



Time

$$O(m \log^c m)$$

Koutis-Miller-Peng '11

Solve in time $O(m \log n \ \log^2 \log n \ \log(1/\epsilon))$

Build Ultra-Sparsifier by:

1. Constructing low-stretch spanning tree
2. Adding other edges with probability

$$p_{u,v} \sim \text{path-length}_T(u, v)$$

Conclusions

Laplacian Solvers are a powerful primitive!

Faster Maxflow: Christiano-Kelner-Madry-S-Teng

Faster Random Spanning Trees: Kelner-Madry-Propp

All Effective Resistances: S-Srivastava

Can we solve all well-conditioned graph problems in nearly-linear time?

Don't fear large constants

Open Problems

Faster and better Low-Stretch Spanning Trees.

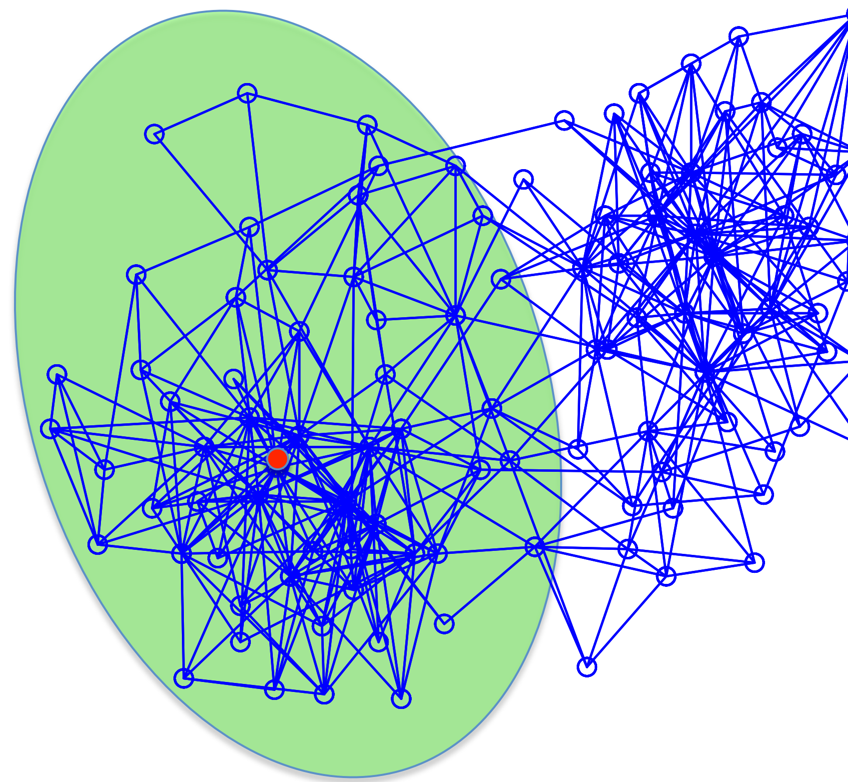
Faster high-quality sparsification.

Other families of linear systems.

From optimization, machine learning, etc.

Local Graph Clustering [S-Teng '04]

Given vertex of interest find nearby cluster S with small conductance in time O(|S|)



Local Graph Clustering [S-Teng '04]

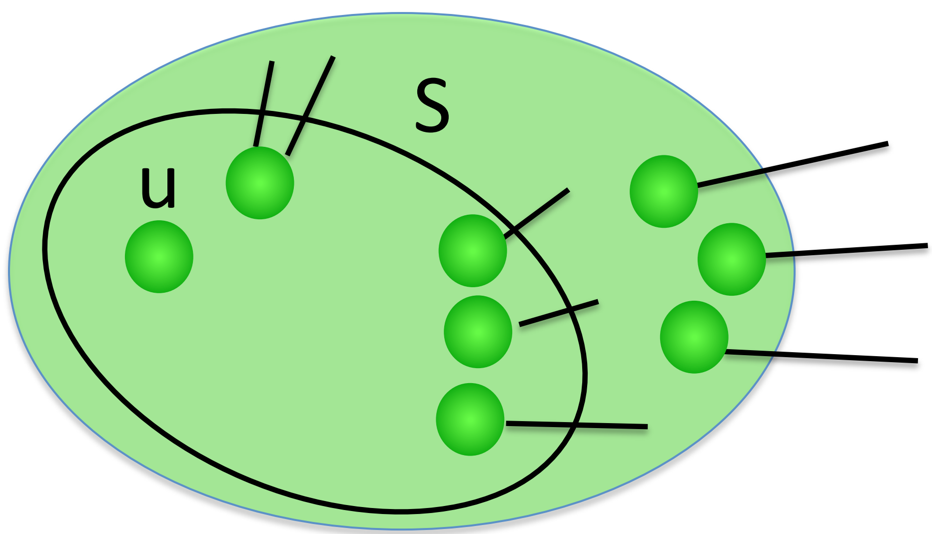
Prove: if S has small conductance ϕ

u is a random node in S

probably

find a set of small conductance, $\phi^{1/2} \log^c n$

in time $|S| \log^c n / \phi^2$



Using Approximate Personal PageRank Vectors

Jeh-Widom '03, Berkhin '06, Andersen-Chung-Lang '06

Spilling paint in a graph:
start at one node
at each step,
α fraction dries
of wet paint, half stays put, half to neighbors

Using Approximate Personal PageRank Vectors

Jeh-Widom '03, Berkhin '06, Andersen-Chung-Lang '06

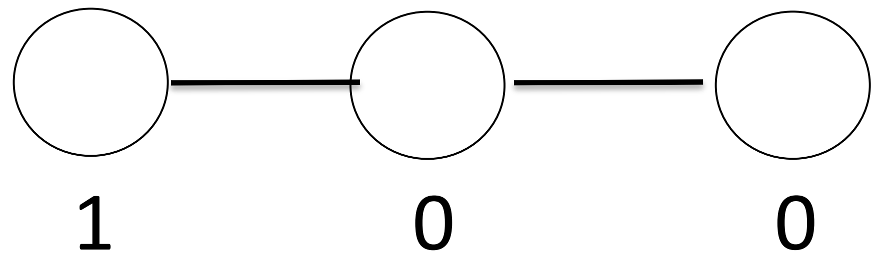
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

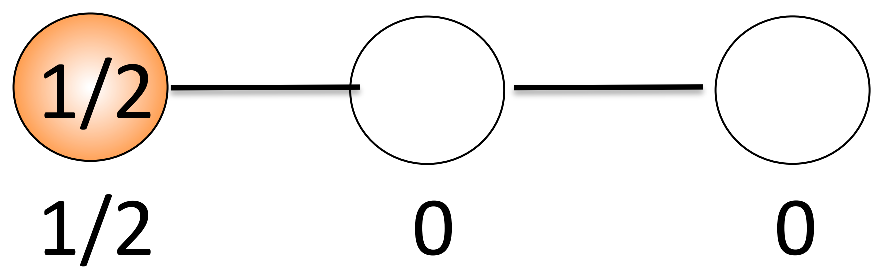
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



with $\alpha = 1/2$

Using Approximate Personal PageRank Vectors

Jeh-Widom '03, Berkhin '06, Andersen-Chung-Lang '06

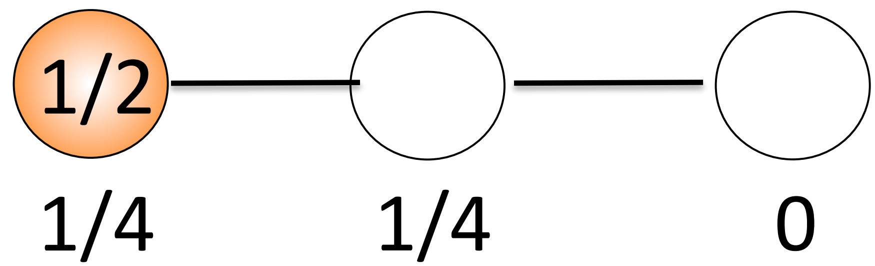
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



with $\alpha = 1/2$

Using Approximate Personal PageRank Vectors

Jeh-Widom '03, Berkhin '06, Andersen-Chung-Lang '06

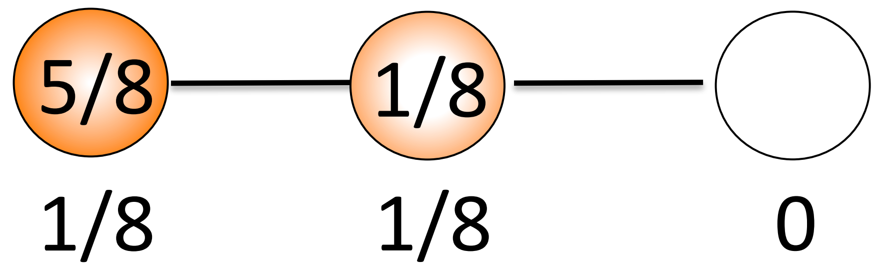
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



with $\alpha = 1/2$

Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

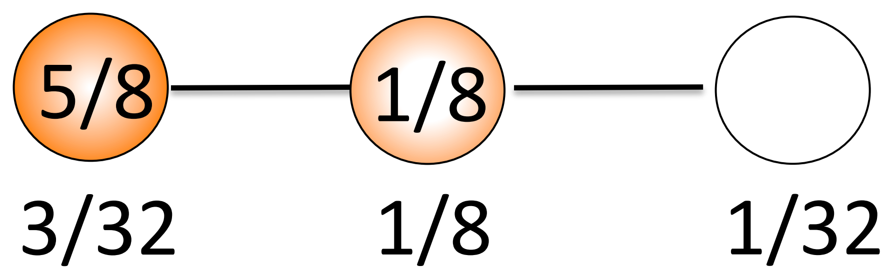
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



with $\alpha = 1/2$

Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors

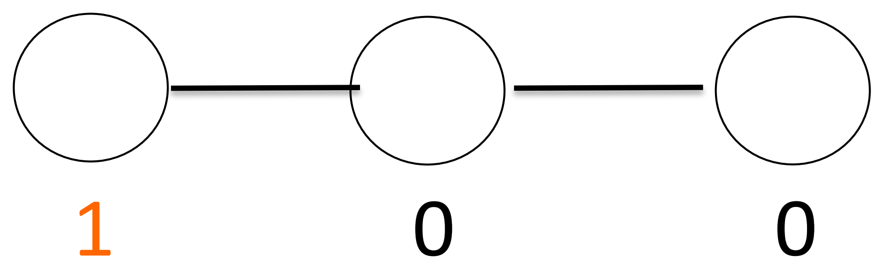
Time doesn’t matter, can push paint whenever

Approximate: only push when a lot of paint

Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

Spilling paint in a graph:
  start at one node
  at each step,
    α fraction dries
    of wet paint, half stays put, half to neighbors



Time doesn’t matter, can push paint whenever

Approximate: only push when a lot of paint

Using Approximate Personal PageRank Vectors

Jeh-Widom '03, Berkhin '06, Andersen-Chung-Lang '06

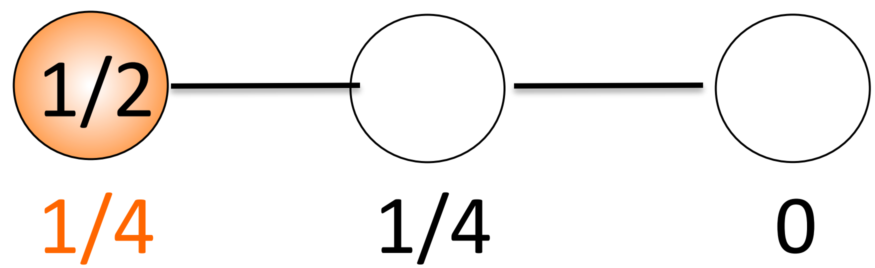
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



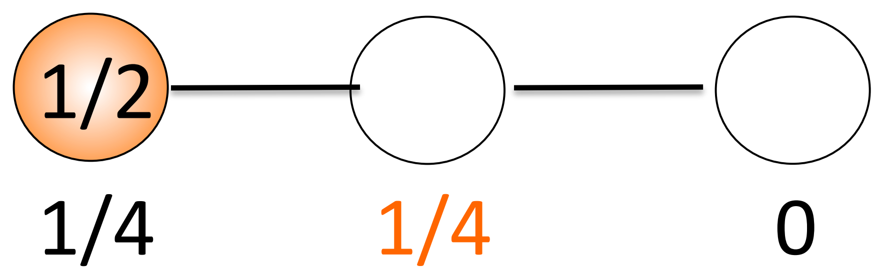
Time doesn't matter, can push paint whenever

Approximate: only push when a lot of paint

Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

Spilling paint in a graph:
- start at one node
- at each step,
  - α fraction dries
  - of wet paint, half stays put, half to neighbors



Time doesn’t matter, can push paint whenever

Approximate: only push when a lot of paint

Using Approximate Personal PageRank Vectors

Jeh-Widom ‘03, Berkhin ‘06, Andersen-Chung-Lang ’06

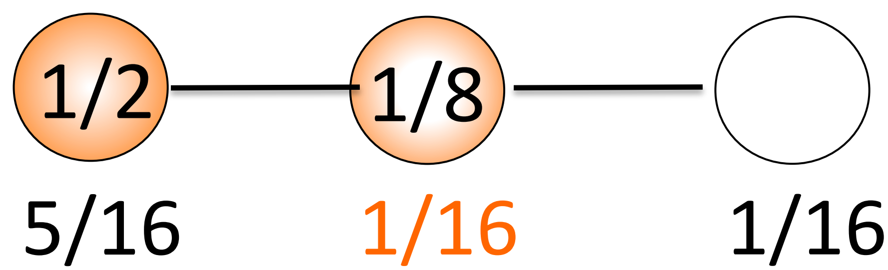
Spilling paint in a graph:

start at one node

at each step,

α fraction dries

of wet paint, half stays put, half to neighbors



Time doesn’t matter, can push paint whenever

Approximate: only push when a lot of paint

Volume-Biased Evolving Set Markov Chain

[Andersen-Peres '09]

Walk on sets of vertices
starts at one vertex, ends at V

Dual to random walk on graph

When start inside set of conductance ϕ
find set of conductance $\phi^{1/2} \log^{1/2} n$
with work $|S| \log^c n / \phi^{1/2}$

Volume-Biased Evolving Set Markov Chain

[Andersen-Peres '09]

Walk on sets of vertices
 starts at one vertex, ends at V

Dual to random walk on graph

When start inside set of conductance ϕ
 find set of conductance $\phi^{1/2} \log^{1/2} n$
 with work $|S| \log^c n / \underline{\phi^{1/2}}$

can we eliminate this?

Open Problems

Faster and better Low-Stretch Spanning Trees.

Faster high-quality sparsification.

Other families of linear systems.

Faster and better local clustering.