A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time

Lorenzo Orecchia, MIT Math

Joint work with Jonathan Kelner, Aaron Sidford and Zeyuan Allen Zhu

MIT MATHEMATICS

CSAIL

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$Ax = b$$

SQUARE SYSTEM OF LINEAR EQUATIONS

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

- Symmetry: $A^{\mathrm{T}} = A$

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

- Symmetry: $A^{\mathrm{T}} = A$
- Diagonal Dominance:
  for all rows (and columns) i, diagonal entry dominates

$$a_{ii} \geq \sum_{j \neq i} |a_{ij}|$$

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

- Symmetry: $A^{\mathrm{T}} = A$
- Diagonal Dominance: for all i, $a_{ii} \geq \sum_{j \neq i} |a_{ij}|$

WHY SDD ? Practically important subcase of positive semidefinite matrices

A **IS SDD** $\longrightarrow$ $A \succeq 0$

Easy-to-identify null space

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

GOAL: solve for x

Restriction: A is **S**ymmetric **D**iagonally-**D**ominant

- Symmetry: $A^{\mathrm{T}} = A$
- Diagonal Dominance: for all i, $a_{ii} \geq \sum_{j \neq i} |a_{ij}|$

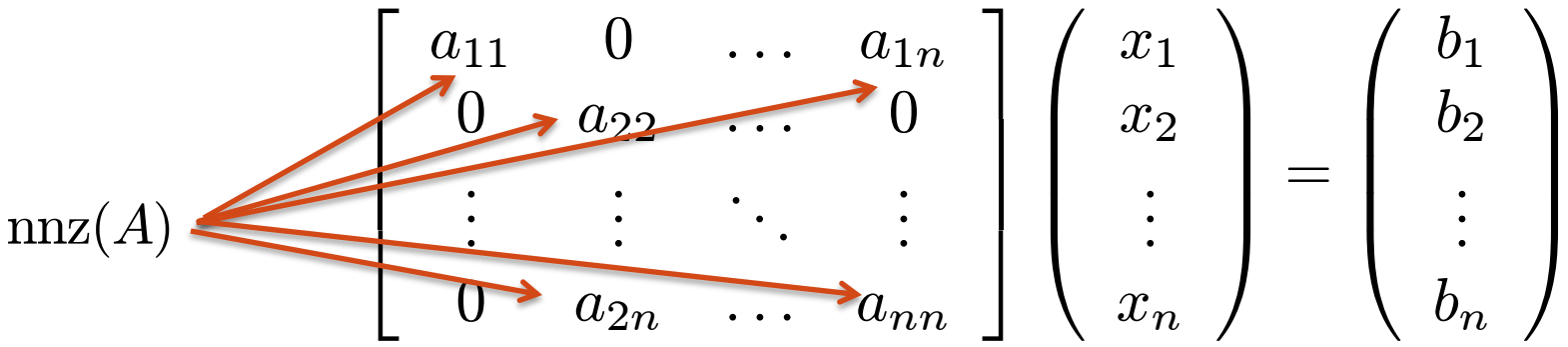
WHY SDD ? Practically important subcase of positive semidefinite matrices

A **IS SDD** $\longrightarrow$ $A \succeq 0$

Easy-to-identify null space

Problem Definition: SDD Systems

PROBLEM INPUT: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, b \in \mathrm{Im}(A)$



GOAL: solve for x

- approximately: $\|x - x^*\|_A \cdot \epsilon \|x^*\|_A$

- in nearly-linear time in the sparsity $\mathrm{nnz}(A)$ and $\log(1/\epsilon)$, i.e.

$$\text{RunTime} = O\left(\mathrm{nnz}(A) \cdot \mathrm{polylog}(n) \cdot \log(1/\epsilon)\right)$$

Reduction to Laplacian Systems

SDD SYSTEM:

$$Ax = b$$

[Gremban'96]

LAPLACIAN SYSTEM:

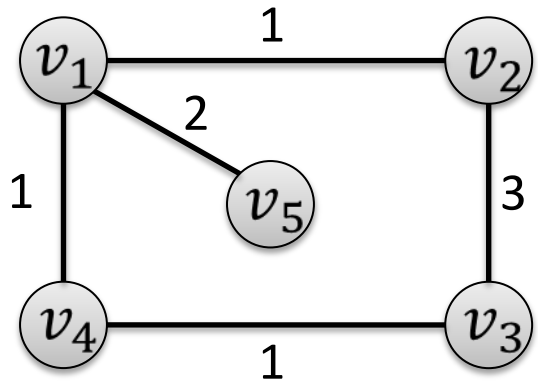
$$Lv = \chi$$

Reduction preserves approximation and sparsity

Graph Laplacian

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

GRAPH G



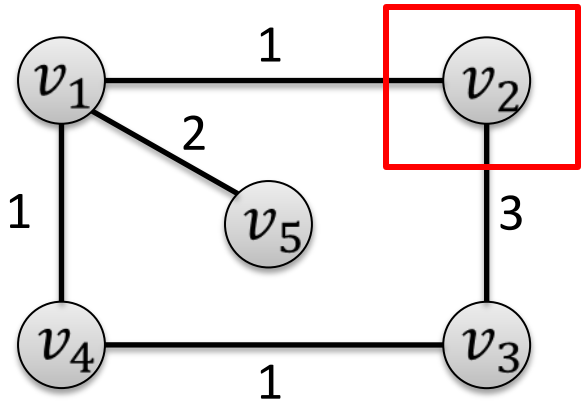
GRAPH LAPLACIAN $L(G)$

$$L(G) = \begin{bmatrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 4 & -3 & 0 & 0 \\ 0 & -3 & 4 & -1 & 0 \\ -1 & 0 & -1 & 2 & 0 \\ -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Graph Laplacian

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

GRAPH G



GRAPH LAPLACIAN $L(G)$

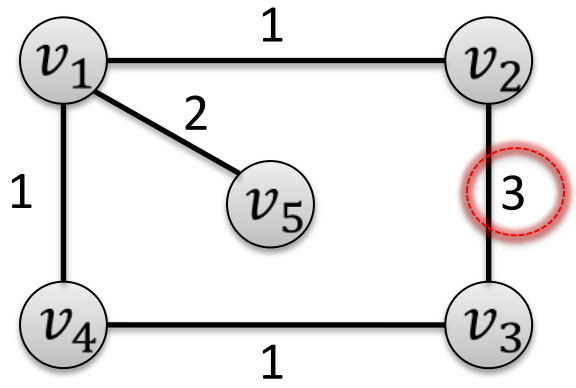
$$L(G) = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & & & \\ 0 & -3 & & & \\ -1 & 0 & -1 & 2 & 0 \\ -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Degree of vertex 2

Graph Laplacian

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

GRAPH G



GRAPH LAPLACIAN $L(G)$

$$L(G) = \begin{bmatrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 4 & -3 & & \\ 0 & -3 & 4 & & \\ -1 & 0 & -1 & & \\ -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

- Weight of edge {2,3}

Laplacian as Sum of Edges

$$L(G) = \sum_{e=\{i,j\}\in E} w_e \begin{array}{c} \\ \\ \\ i \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \\ \end{array} \\ \left(\begin{array}{ccccccc} 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & \dots & 1 & \dots & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & \dots & -1 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 0 \end{array}\right) \end{array}$$

Laplacian as Sum of Edges

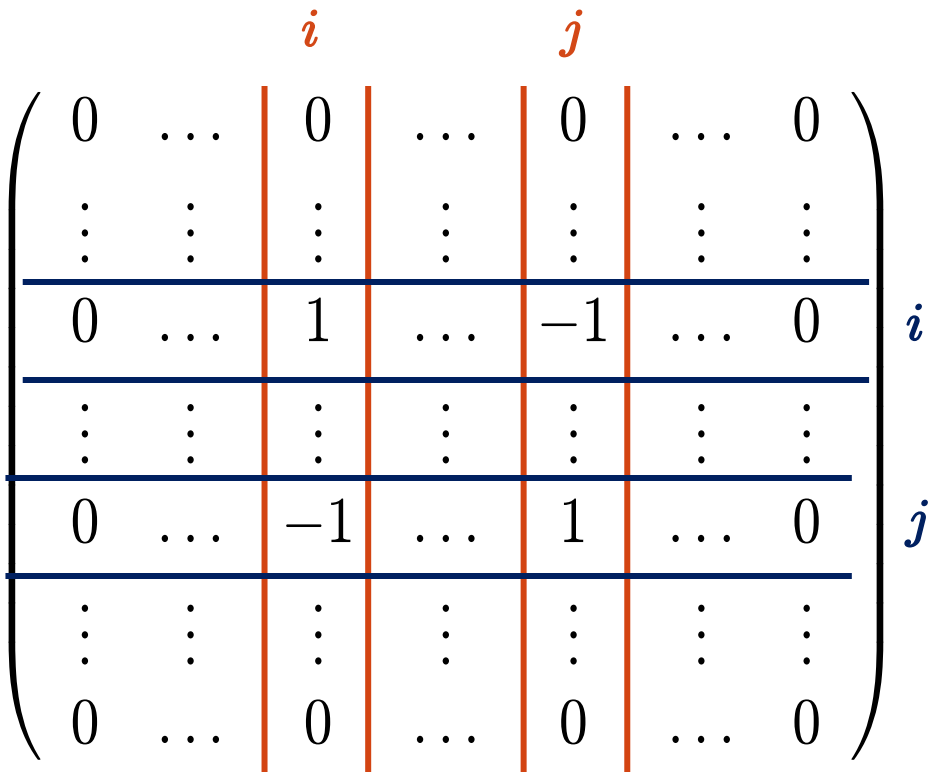
$$L(G) = \sum_{e=\{i,j\}\in E} w_e$$

$$\begin{array}{c} \\ \\ i \\ \\ j \\ \\ \end{array}
\begin{array}{c}
\begin{array}{ccccccc} \quad & \quad & i & \quad & j & \quad & \quad \end{array} \\
\left(\begin{array}{ccccccc}
0 & \dots & 0 & \dots & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
0 & \dots & 1 & \dots & -1 & \dots & 0 \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
0 & \dots & -1 & \dots & 1 & \dots & 0 \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \dots & 0 & \dots & 0 & \dots & 0
\end{array}\right)
\end{array}
\begin{array}{l} \\ \\ i \\ \\ j \\ \\ \end{array}$$

Edge Matrix L_e

Laplacian as Sum of Edges

$$L(G) = \sum_{e=\{i,j\}\in E} w_e$$

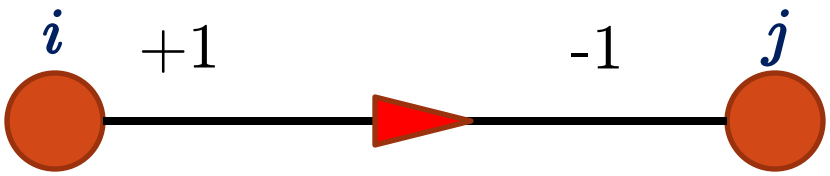


Edge Matrix L_e has rank 1

Laplacian as Sum of Edges

$$L(G) = \sum_{e=\{i,j\}\in E} w_e \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ i \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ j \\ \\ \end{matrix} \begin{pmatrix} 0 & \cdots & 1 & \cdots & -1 & \cdots & 0 \end{pmatrix}^{\mathrm{T}}$$

$$\chi_e \qquad\qquad \chi_e^{\mathrm{T}}$$



ARBITRARY ORIENTATION OF EDGES: WILL BE USEFUL WHEN DISCUSSING FLOWS

More Graph Matrices: Incidence Matrix

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

$$B^{\mathrm{T}} = \begin{pmatrix} & \\ \text{——} & \chi_{e_1} & \text{——} \\ \text{——} & \chi_{e_2} & \text{——} \\ & \vdots & \\ \text{——} & \chi_{e_i} & \text{——} \\ & \vdots & \\ \text{——} & \chi_{e_m} & \text{——} \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \quad \begin{matrix} n \\ \\ \\ \\ \\ \\ \\ m \end{matrix}$$

More Graph Matrices: Incidence Matrix

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

$$B^{\mathrm{T}} = \overbrace{\begin{pmatrix} \text{———} & \chi_{e_1} & \text{———} \\ \text{———} & \chi_{e_2} & \text{———} \\ & \vdots & \\ \text{———} & \chi_{e_i} & \text{———} \\ & \vdots & \\ \text{———} & \chi_{e_m} & \text{———} \end{pmatrix}}^{n} \left.\vphantom{\begin{pmatrix} a \\ a \\ a \\ a \\ a \\ a \end{pmatrix}}\right\} m$$

$$L = \sum_{e=\{i,j\}\in E} w_e \chi_e \chi_e^{\mathrm{T}} = B^T \underset{\downarrow \\ \mathrm{diag}(w)}{W} B$$

More Graph Matrices: Incidence Matrix

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges

$$B^{\mathrm{T}} = \begin{pmatrix} \text{——} & \chi_{e_1} & \text{——} \\ \text{——} & \chi_{e_2} & \text{——} \\ & \vdots & \\ \text{——} & \chi_{e_i} & \text{——} \\ & \vdots & \\ \text{——} & \chi_{e_m} & \text{——} \end{pmatrix} \quad \begin{matrix} \leftarrow n \rightarrow \\ \\ \\ \\ \\ \\ \\ \end{matrix} \quad m$$

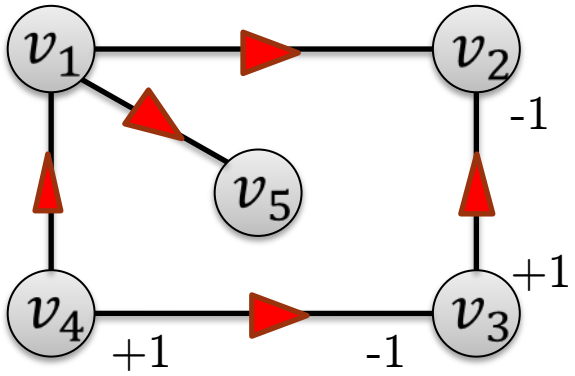
$$L = \sum_{e=\{} w_e \chi_e \chi_e^{\mathrm{T}} = B^T W B$$

$W^{1/2} B$ is square root of L

$\mathrm{diag}(w)$

Action of Incidence Matrix

$G = (V, E, w)$ weighted undirected graph with n vertices and m edges



Action of B^T on a vector $f \in \mathrm{R}^m$:

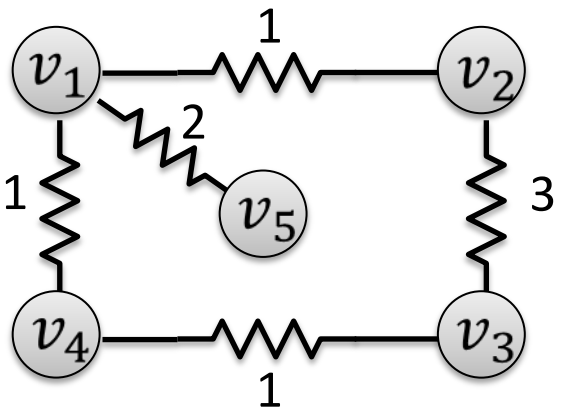
$$(B^T f)_i = \text{flow out of i} - \text{flow into of i} = \text{net flow into graph at i}$$

Action of B on a vector $v \in \mathrm{R}^n$:

$$(Bv)_{e=(i,j)} = v_i - v_j = \text{change in } v \text{ along } (i,j)$$

Graphs as Electrical Circuits

Edge conductances w_e

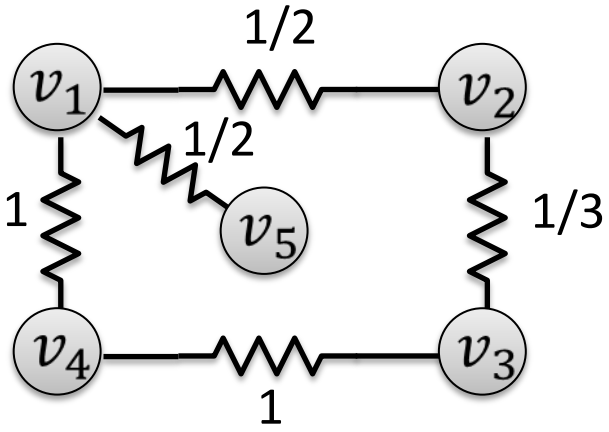


Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$$r_e = 1/w_e$$

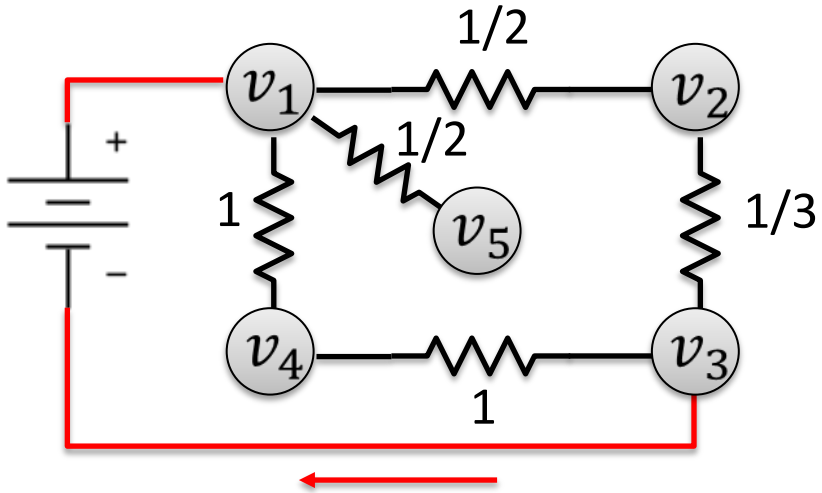


Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



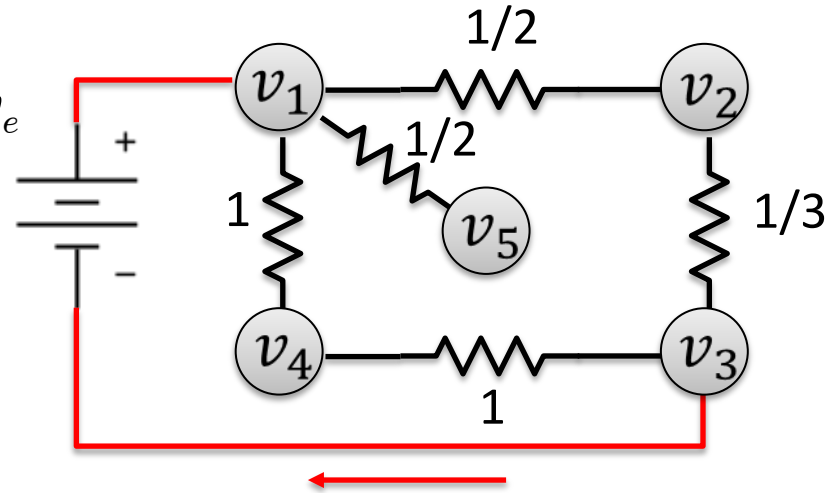
one unit of electrical flow
Into vertex 1 and out of 3

Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



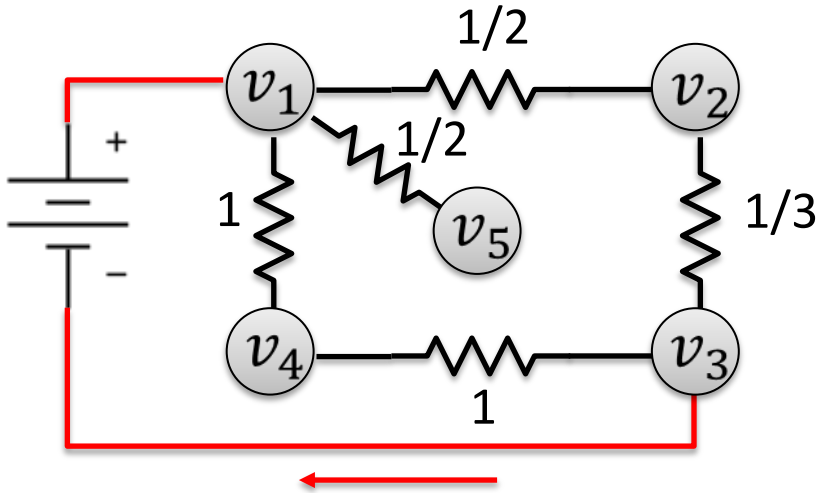
Q: What is resulting electrical flow on the graph?

Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



1. Ohm's Law: For every edge $e = (i, j) \in E$:

$$f_{(i,j)} = \frac{v_i - v_j}{r_{ij}}$$

2. Kirchoff's Conservation Law: For every vertex $i \in V$:

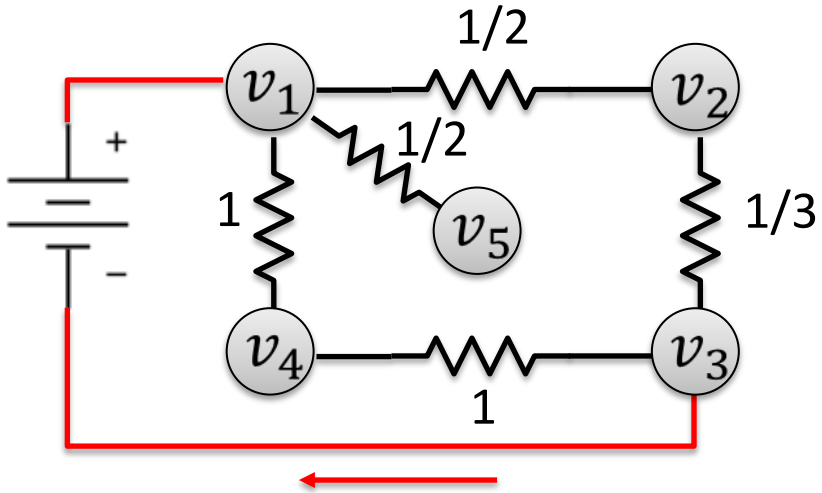
$$\text{flow out of } i - \text{flow into } i = \text{net flow into network at } i$$

Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



one unit of electrical flow
Into vertex 1 and out of 3

1. Ohm's Law: For every edge $e = (i, j) \in E$:

$$f_{(i,j)} = \frac{v_i - v_j}{r_{ij}} \longrightarrow f = R^{-1}Bv = WBv$$

2. Kirchoff's Conservation Law: For every vertex $i \in V$:

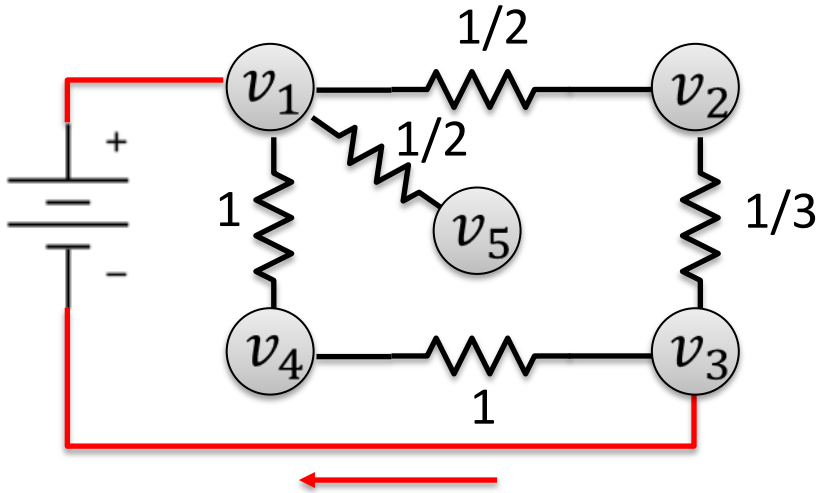
$$\text{flow out of } i - \text{flow into } i = \text{net flow into network at } i$$

Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



1. Ohm's Law:

$$f = R^{-1}Bv = WBv$$

2. Kirchoff's Conservation Law:

$$B^T f = e_s - e_t$$

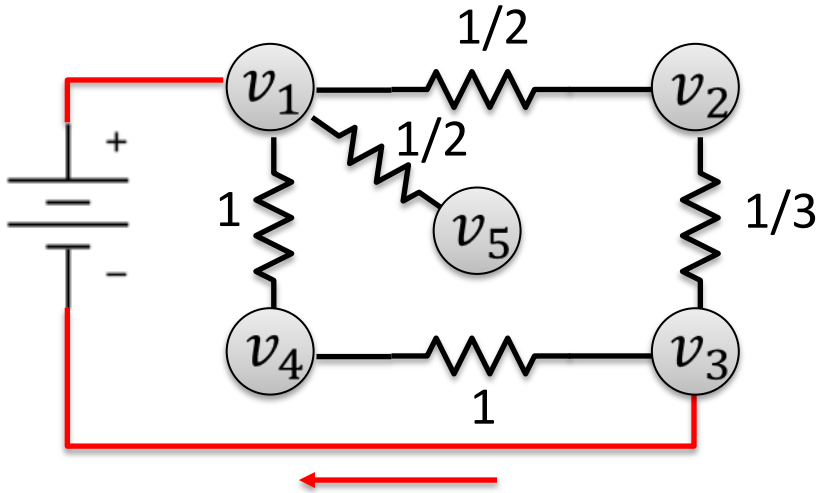
Example above: Current source s is vertex 1. Current sink t is vertex 3

Graphs as Electrical Circuits

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



one unit of electrical flow
Into vertex 1 and out of 3

1. Ohm's Law:

$$f = R^{-1}Bv = WBv$$

2. Kirchoff's Conservation Law:

$$B^T f = \chi$$

General net flow vector χ allowed:

$$\chi^{\mathrm{T}}\vec{1} = 0$$

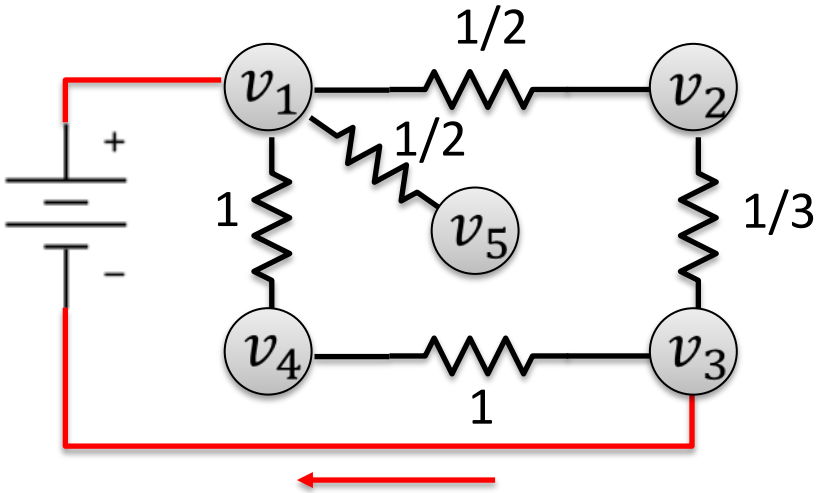
Example above: Current source s is vertex 1. Current sink t is vertex 3

Laplacian Systems as Electrical Problems

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



General net flow vector χ :

$$\chi^{\mathrm{T}}\vec{1} = 0$$

Q: What is resulting electrical flow on the graph?

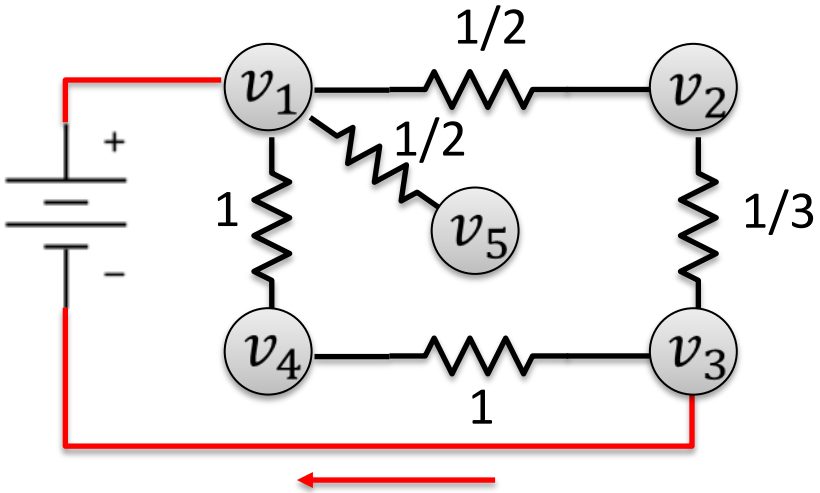
$$\left.\begin{array}{c} f = R^{-1}Bv = WBv, \\ B^{\mathrm{T}}f = \chi \end{array}\right\} \rightarrow B^T W B v = Lv = \chi$$

Laplacian Systems as Electrical Problems

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



General net flow vector χ :

$$\chi^{\mathrm{T}}\vec{1} = 0$$

Q: What is resulting electrical flow on the graph?

$$Lv = \chi$$

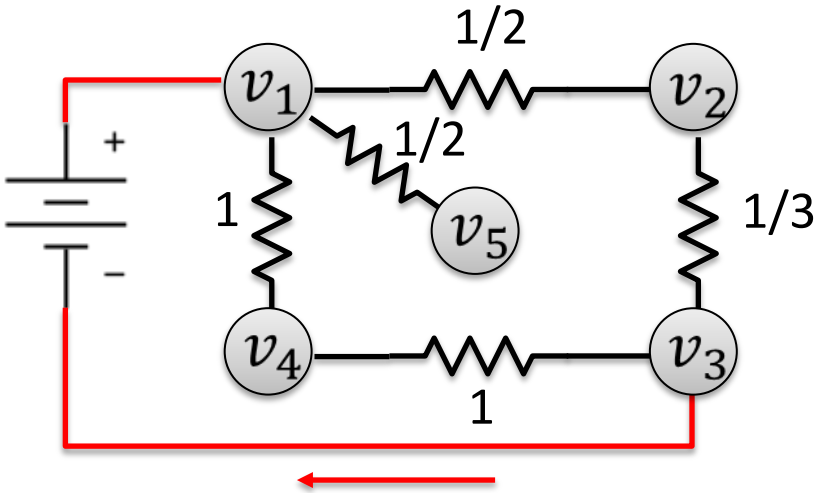
Given input currents χ, finding voltage is equivalent to solving Laplacian system of linear equations

Laplacian Systems as Electrical Problems

Edge conductances w_e

Edge resistances r_e

$r_e = 1/w_e$



General net flow vector χ :

$$\chi^{\mathrm{T}}\vec{1} = 0$$

Q: What is resulting electrical flow on the graph?

Pseudo-inverse

$$Lv = \chi \longrightarrow v = L^{+}\chi$$

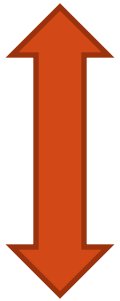
Given input currents χ, finding voltage is equivalent to solving Laplacian system of linear equations

Energy Intepretation

$$Lv = \chi$$

Optimality condition

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi$$



Equivalent up to scaling

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot x^{\mathrm{T}} L x$$

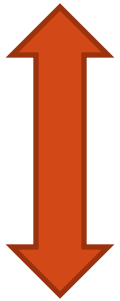
$$\text{s.t.} \quad x^{\mathrm{T}} \chi = 1$$

Energy Intepretation

$$Lv = \chi$$

Optimality condition

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi$$



Equivalent up to scaling

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} L x = \sum_{(i,j) \in E} \frac{(x_i - x_j)^2}{r_{ij}}$$

$$\text{s.t.} \qquad x^{\mathrm{T}} \chi = 1$$

Minimize energy for a fixed voltage gap

Why it matters

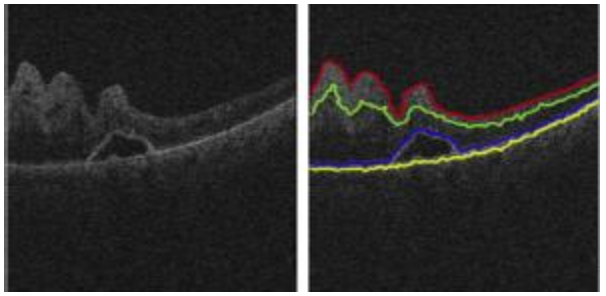
- Direct Applications
  - Modeling electrical networks
  - Simulating random walks
  - PageRank

$$L^{+}\chi = \sum_{t=0}^{\infty} \left(\frac{I+W}{2}\right)^{t} \chi$$

Aggregate behavior of lazy random walk started at χ

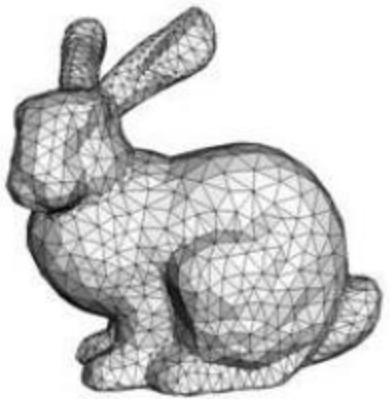
Why it matters

- **Direct Applications**
  - Modeling electrical networks
  - Simulating random walks
  - PageRank



Numerical Applications

- Finite element [BHV08]
- Matrix exponential [OSV12]
- Largest eigenvalue [ST12]
- Image Smoothing



Why it matters: Faster Graph Algorithms

"The Laplacian Paradigm"

- Maximum flow [CKM+11,LRS13,KOLS12]
- Multicommodity flow [KMP12, KOLS12]
- Random spanning trees [KM09]
- Graph sparsification [SS11]
- Lossy flow, min-cost flow [DS08]
- Balanced partitioning [OSV12]
- Oblivious routing [KOLS12]
- … and more

Highlights of Previous Work

Direct solvers for general matrices

- Gaussian elimination $O(n^3)$
- Strassen alg $O(n^{2.807})$
- …
- Williams alg $O(n^{2.373})$

Iterative methods for PSD matrices

- Conjugate Gradient
- Chebyshev Method
- $O(nm)$

On various subclasses

- Multigrid
- Many many others…

For SDD/ Laplacians

- $O(m^{1.75})$ started by [Vaidya'90]
- …
- Spielman-Teng O(m polylog n)
- Sped up by [KMP'12] to $\tilde{O}(m \log n)$

Our Result

Solve $Lx = b$ in time $\tilde{O}\left(m \log^2 n \log \frac{1}{\epsilon}\right)$

- Very different approach
- Simple and intuitive algorithm
- Proof fits on a single blackboard
- Easily shown to be numerically stable

Our Result

Solve $Lx = b$ in time $\tilde{O}\left(m \log^2 n \log \frac{1}{\epsilon}\right)$

- Very different approach
- Simple and intuitive algorithm
- Proof fits on a single blackboard
- Easily shown to be numerically stable

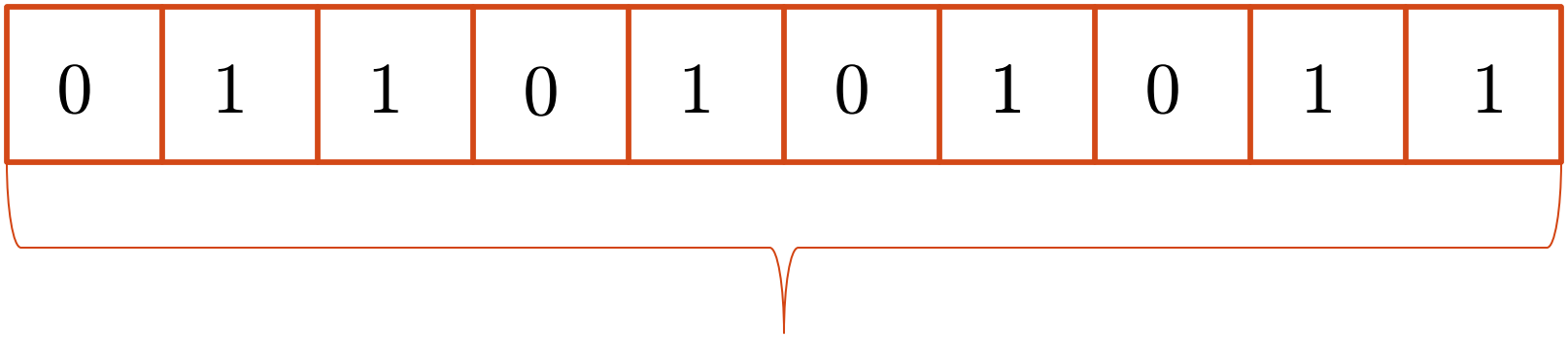
Previous methods are not as stable, need $\log^c n$-bit precision (although $\tilde{O}(m \log n)$ is achieved, an extra $\log^c n$ factor is hidden)

In unit-cost RAM model, our algorithm is faster.

Which computational model?

Q: How do we account for running time of arithmetic operations?

A: Constant time for operations on word-size sequence of bits.



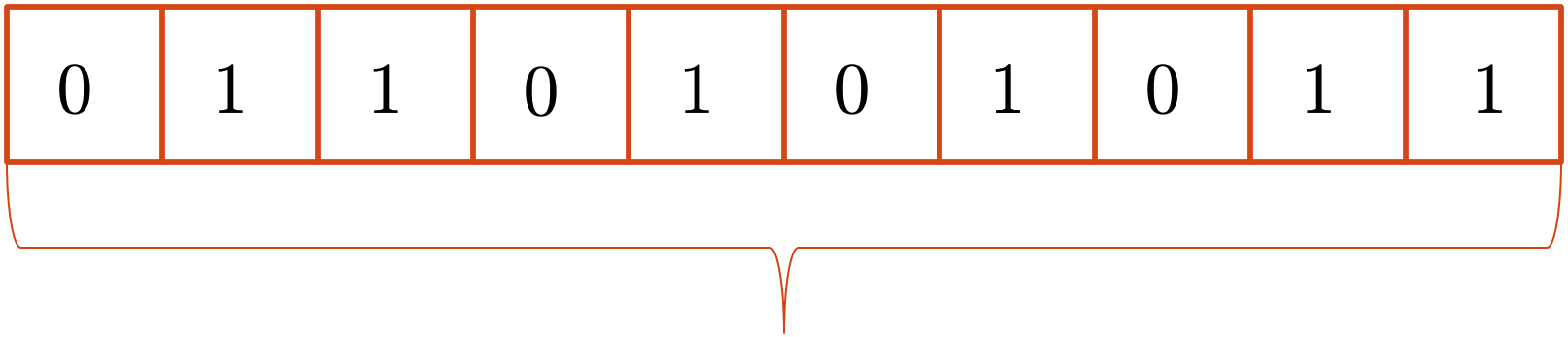
Previous works: $O(\log^c n)$

Size of Word: Polynomial in size of input

Which computational model?

Q: How do we account for running time of arithmetic operations?

A: Constant time for operations on word-size sequence of bits.



Previous works: $O(\log^c n)$

Size of Word: Polynomial in size of input

Our work: $O(\log n)$

UNIT-COST RAM MODEL:
MORE REALISTIC MODEL

Size of Word: Linear in size of input

The Philosophy of This Talk

PROBLEM REPRESENTATION

ITERATIVE METHOD

The Philosophy of This Talk

PROBLEM REPRESENTATION

Not all representations created equally:
e.g. eigenvalue decompostion

Good representation often requires **combinatorial insight**

ITERATIVE METHOD

The Philosophy of This Talk

PROBLEM REPRESENTATION

Not all representations created equally:
e.g. eigenvalue decompostion

Good representation often requires **combinatorial insight**

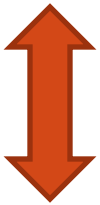
ITERATIVE METHOD

Leverage large body of techniques in **continuous optimization**

- Regularization
- Gradient Descent
- Accelerated Gradient Descent
- Randomized Coordinate Descent

The Philosophy of This Talk

PROBLEM REPRESENTATION



ITERATIVE METHOD

Not all representations created equally:
e.g. eigenvalue decompostion

Good representation often requires **combinatorial insight**

Leverage large body of techniques in **continuous optimization**

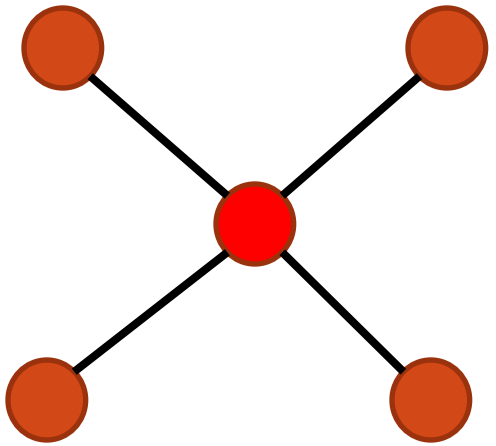
- Regularization
- Gradient Descent
- Accelerated Gradient Descent
- Randomized Coordinate Descent

Efficiency and simplicity of algorithm relies on **combining these two tools** in the right way

Techniques and Challenges in the Solution of Laplacian Systems

Changing Representation: Gaussian Elimination

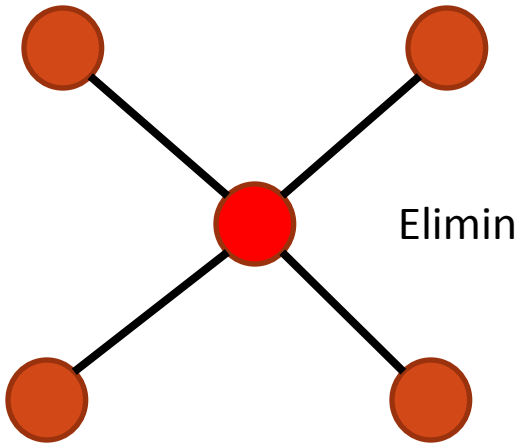
Simple Graphic Interpretation:



$$L = \begin{pmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Changing Representation: Gaussian Elimination

Simple Graphic Interpretation:

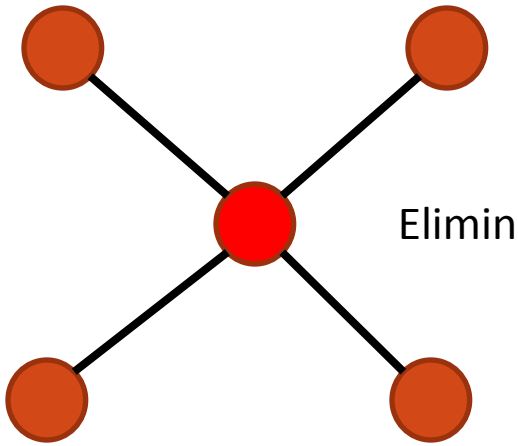


Eliminate (pivot) this node

$$\begin{pmatrix} \boxed{4} & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Changing Representation: Gaussian Elimination

Simple Graphic Interpretation:

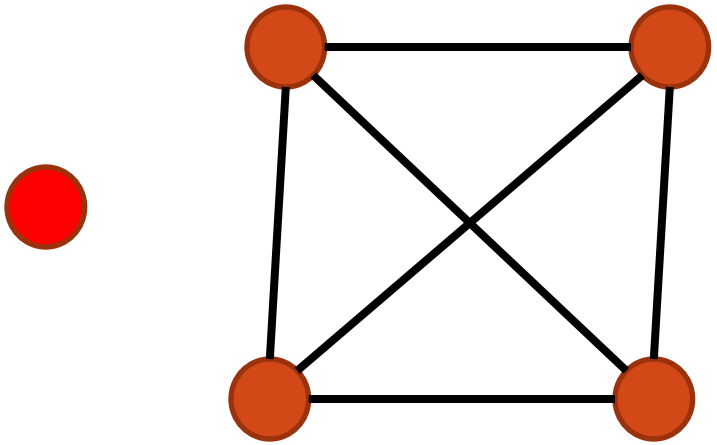


Eliminate (pivot) this node

$$\begin{pmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Changing Representation: Gaussian Elimination

Simple Graphic Interpretation:

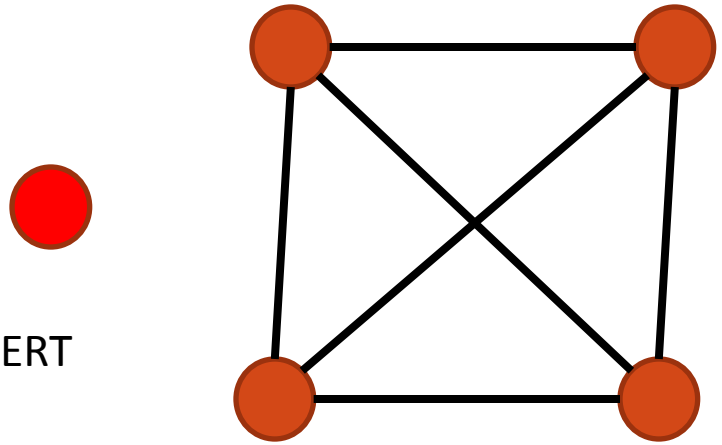


$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & -1 & \frac{3}{4} \end{pmatrix}$$

→ Laplacian on n-1 vertices

Cholesky Decomposition

Simple Graphic Interpretation:



EASY TO INVERT

EASY TO INVERT

$$L = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & -1 & \frac{3}{4} \end{pmatrix} \begin{pmatrix} 4 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Changing Representation: Gaussian Elimination

Advantages:

- Gives exact algorithm
- Computes inverse matrix (can then be used on any b)
- Can choose elimination order

Changing Representation: Gaussian Elimination

Advantages:

- Gives exact algorithm
- Computes implicit representation of inverse matrix (can then be used on any b)
- Can choose elimination order

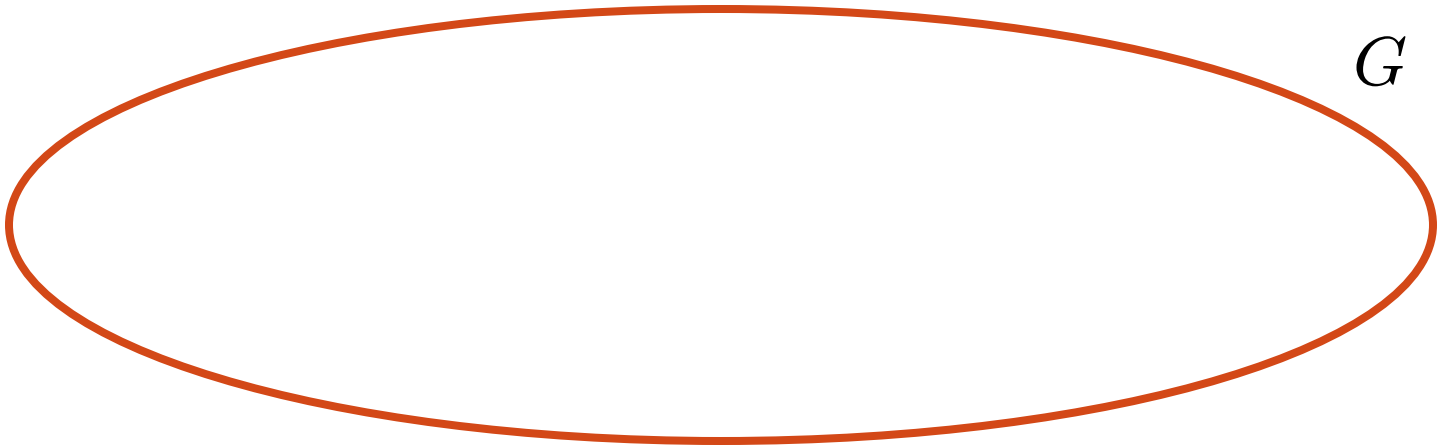
Disadvantages:

- Intermediate Laplacians can be very dense

SPARSE **DENSE**

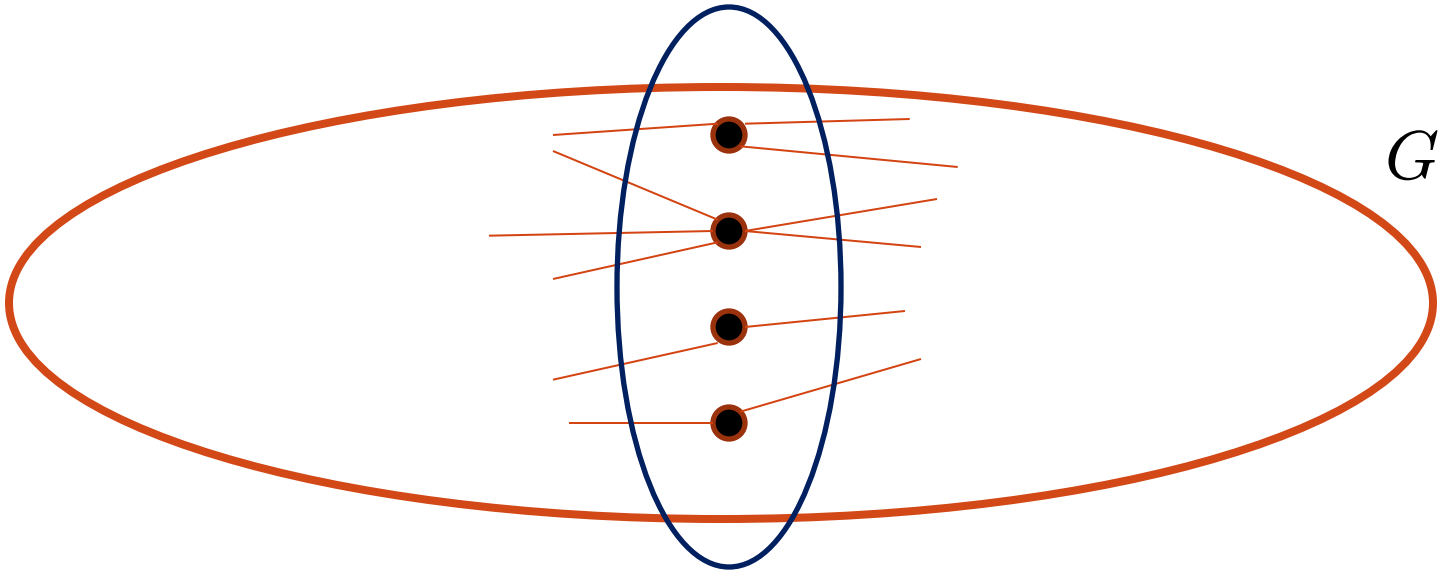
Gaussian Elimination and Nested Dissection

Combinatorial arguments can help preserve sparsity by selecting good order:



Gaussian Elimination and Nested Dissection

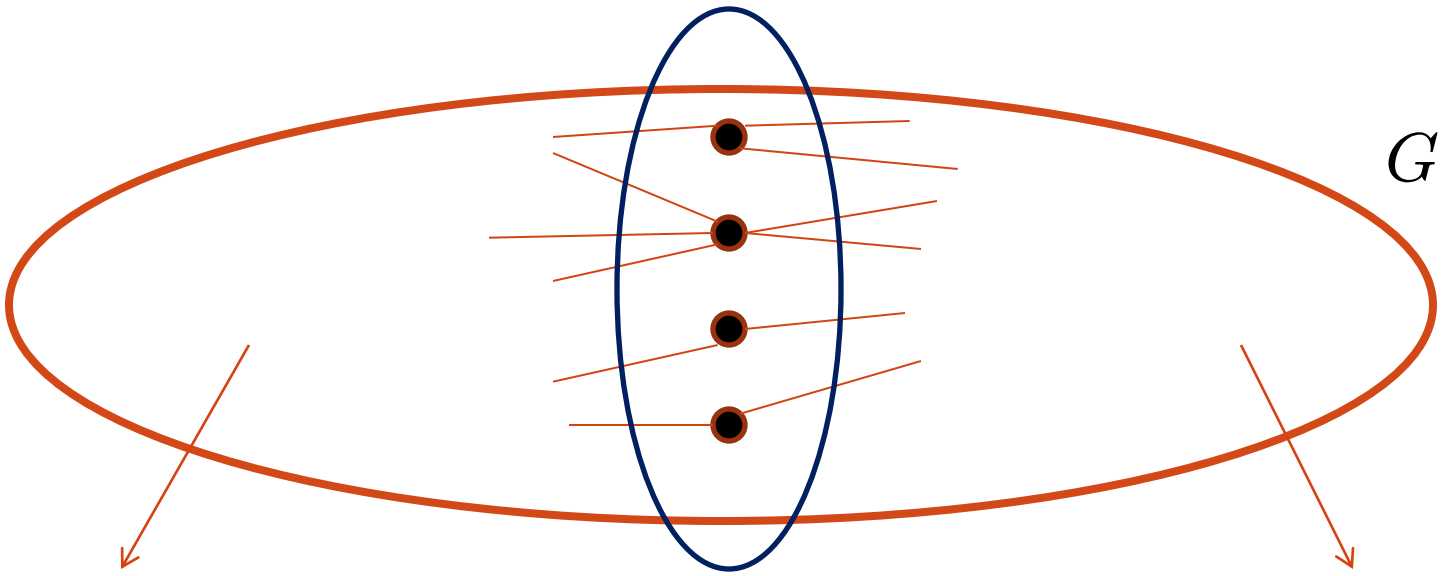
Combinatorial arguments can help preserve sparsity by selecting good order:



Small Vertex Separator

Gaussian Elimination and Nested Dissection

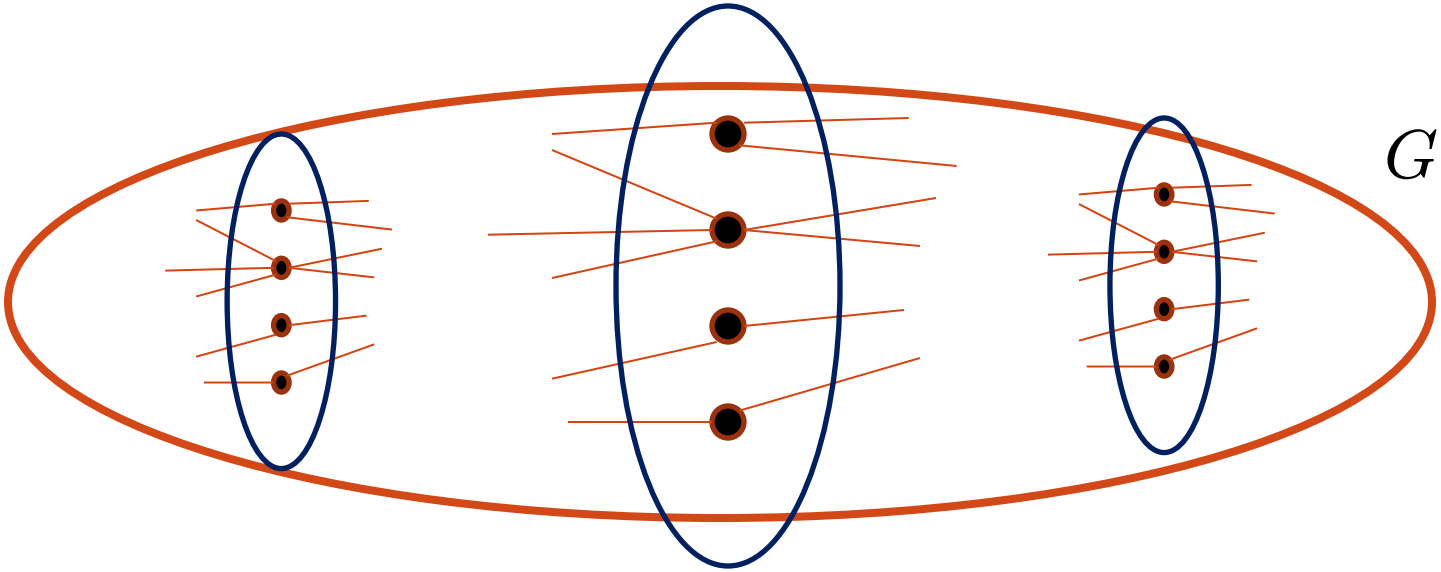
Combinatorial arguments can help preserve sparsity by selecting good order:



Eliminate recursively | Small Vertex Separator | Eliminate recursively

Gaussian Elimination and Nested Dissection

Combinatorial arguments can help preserve sparsity by selecting good order:

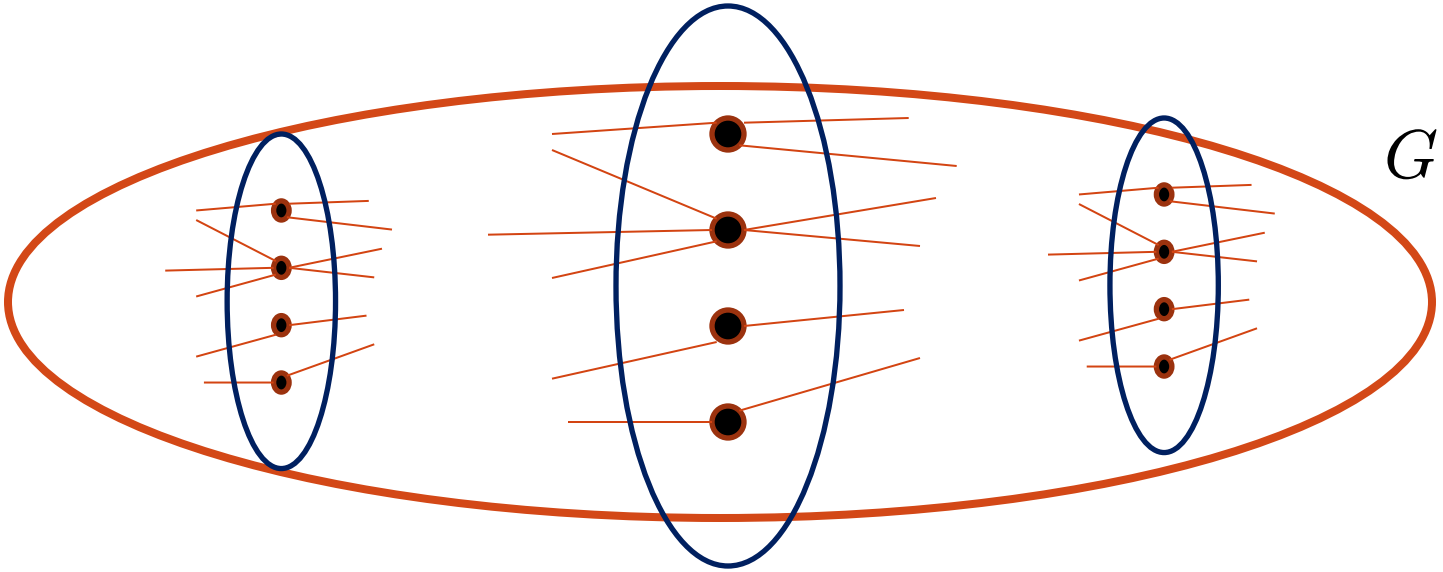


Size of existing separators bounds sparsity

Works well if small separators always exist (and are easy to find)

Gaussian Elimination and Nested Dissection

Combinatorial arguments can help preserve sparsity by selecting good order:

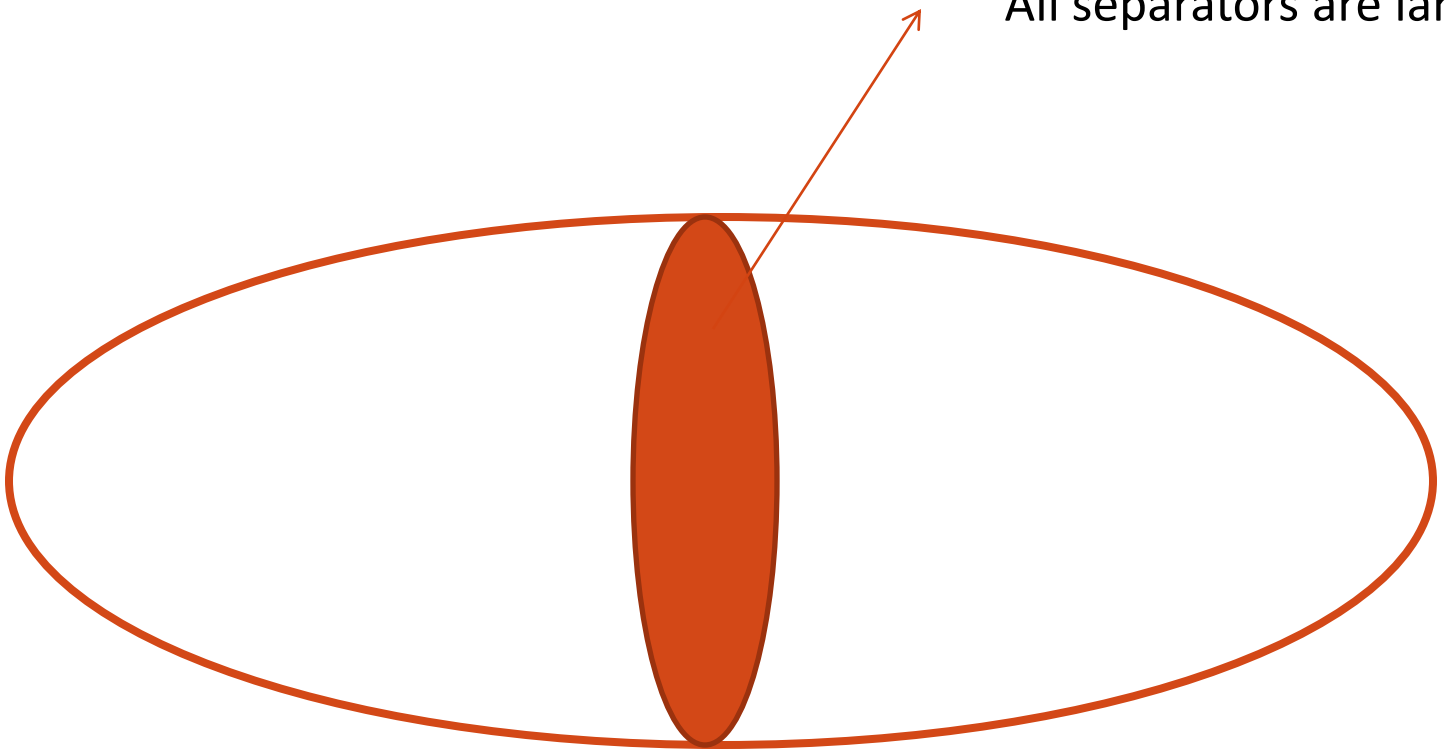


Size of existing separators bounds sparsity
Works well if small separators always exist (and are easy to find)
E.G.: Planar Graphs

Gaussian Elimination: the bad case

G sparse expander

All separators are large



Any elimination order requires producing a dense graph.

Changing Representation: Gaussian Elimination

Advantages:

- Gives exact algorithm
- Computes implicit representation of inverse matrix (can then be used on any b)
- Can choose elimination order to minimize running time

Disadvantages:

- Intermediate Laplacians can be very dense
- **Very slow** in worst case:

$$O(n^3) \longrightarrow O(n^w)$$

FAR FROM NEARLY LINEAR!

Gaussian Elimination: the good case

Easily linear time for some graphs:

PATHS

Gaussian Elimination: the good case

Easily linear time for some graphs:

Eliminate in time $\mathrm{O}(1)$

PATHS

Gaussian Elimination: the good case

Easily linear time for some graphs:

PATHS $O(n)$ time

Gaussian Elimination: the good case

Easily linear time for some graphs:

PATHS $O(n)$ time

This works more in general for trees
by **recursively eliminating a leaf.**

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \underbrace{\boxed{\frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\nabla^2 f(x) = L$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \quad \underbrace{\boxed{\frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

Use degree norm $\| \cdot \|_D$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \quad \underbrace{\boxed{\tfrac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Construct iterative solutions: $x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(t)}, \ldots$

$$x^{(t+1)} = x^{(t)} - hD^{-1}\nabla f(x^{(t)})$$

Step length

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \underbrace{\boxed{\frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Construct iterative solutions: $x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(t)}, \ldots$

$$x^{(t+1)} = x^{(t)} - \frac{1}{2} D^{-1} \nabla f\left(x^{(t)}\right)$$

By standard gradient descent analysis $h = 1/2$
For quadratic function, it can be optimized at every step.

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \underbrace{\boxed{\tfrac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Construct iterative solutions: $x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(t)}, \ldots$

$$x^{(t+1)} = \sum_{j=0}^{t} \left(\frac{I+W}{2} \right)^j \chi$$

.

UNRAVEL RECURSION TO OBTAIN TRUNCATED SERIES: t steps of random walk

Iterative Methods: Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \underbrace{\frac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi}_{f(x)}$$

$$\nabla f(x) = Lx - \chi$$

$$\lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Construct iterative solutions: $x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(t)}, \ldots$

$$x^{(t+1)} = \sum_{j=0}^{t} \left(\frac{I+W}{2} \right)^j \chi$$

Iterations necessary to converge to ϵ-approximate solution:

$$T = O\left(\frac{2}{\lambda_2} \log\left(\frac{n}{\epsilon} \right) \right)$$

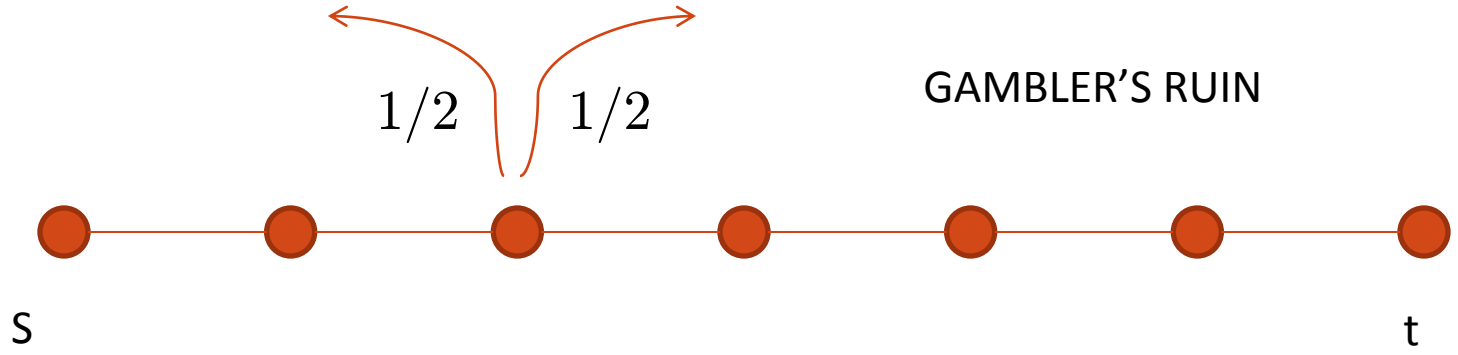
Bad Example for Gradient Descent

PATH: $\lambda_2 = \frac{1}{n^2}$

Iterations necessary to converge to ϵ-approximate solution:

$$T = O\left(\frac{2}{\lambda_2}\log\left(\frac{n}{\epsilon}\right)\right) = O\left(n^2\log\left(\frac{n}{\epsilon}\right)\right)$$

Bad Example for Gradient Descent



PATH: $\lambda_2 = \frac{1}{n^2}$

Iterations necessary to converge to ϵ-approximate solution:

$$T = O\left(\frac{2}{\lambda_2}\log\left(\frac{n}{\epsilon}\right)\right) = O\left(n^2\log\left(\frac{n}{\epsilon}\right)\right)$$

Each Iteration is a matrix-vector multiplication by $\left(\frac{I+W}{2}\right)$, requiring time $\mathrm{O}(m)$

$$\mathrm{RunTime} = O\left(mn^2\log\left(\frac{n}{\epsilon}\right)\right)$$

ESSENTIALLY TIGHT

Bad Example for Gradient Descent

PATH: $\lambda_2 = \frac{1}{n^2}$

Iterations necessary to converge to ϵ-approximate solution:

$$T = O\left(\frac{2}{\lambda_2}\log\left(\frac{n}{\epsilon}\right)\right) = O\left(n^2\log\left(\frac{n}{\epsilon}\right)\right)$$

Each Iteration is a matrix-vector multiplication by $\left(\frac{I+W}{2}\right)$, requiring time $\mathrm{O}(m)$

$$\mathrm{RunTime} = O\left(mn^2\log\left(\frac{n}{\epsilon}\right)\right)$$

Accelerated Gradient Descent

Consider energy interpretation:

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot x^{\mathrm{T}} L x - x^{\mathrm{T}} \chi \qquad \nabla f(x) = Lx - \chi \qquad \lambda_2 D \preceq \nabla^2 f(x) \preceq 2D$$

This is a **convex optimization problem** on which we can apply gradient descent techniques.

Accelerated gradient techniques achieve better convergence:

CHEBYSHEV'S ITERATION **CONJUGATE GRADIENT**

Improved iteration count:

$$T = O\left(\sqrt{\frac{2}{\lambda_2}} \log\left(\frac{n}{\epsilon}\right)\right)$$

Still no luck, but …

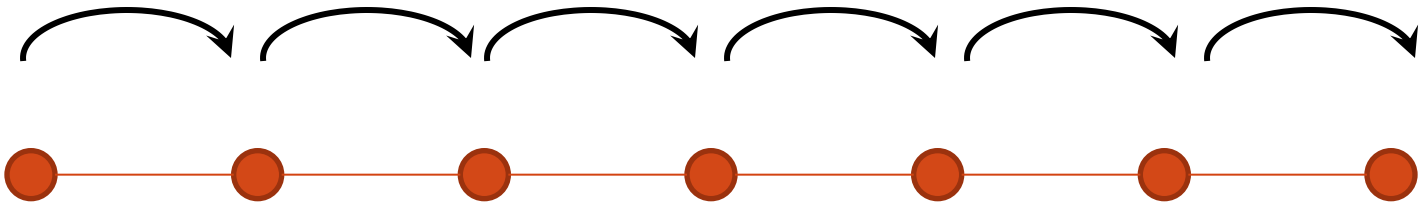
PATH: $\lambda_2 = \frac{1}{n^2}$ $\qquad T = O\left(n \log\left(\frac{n}{\epsilon}\right)\right)$

$$\text{RunTime} = O\left(mn \log\left(\tfrac{n}{\epsilon}\right)\right)$$

BEST POSSIBLE USING GRADIENT APPROACH

Still no luck, but …

IT TAKES n STEPS FOR CHARGE TO TRAVEL ACROSS



PATH: $\lambda_2 = \frac{1}{n^2}$ $\qquad T = O\left(n \log\left(\frac{n}{\epsilon}\right)\right)$

$$\text{RunTime} = O\left(mn \log\left(\tfrac{n}{\epsilon}\right)\right)$$

BEST POSSIBLE USING GRADIENT APPROACH

Combining Representation and Iteration

Gaussian elimination and gradient methods seem **complementary**

- Gaussian elimination is nearly-linear on **paths** (and trees),
  but slow on **expanders**.

- Gradient methods are nearly-linear time on **expanders**,
  but slow on **paths**.

MAIN APPROACH TO FAST SOLVERS:

Combine Gaussian elimination and gradient methods
to obtain **best of both worlds**

Combining Representation and Iteration: Combinatorial Preconditioning

$$Lv = \chi$$

Gradient methods fail when condition number is large.

IDEA: modify system to improve condition number

$$L_H^+ Lv = L_H^+ \chi$$

where H is a **preconditioner** graph.

DESIRED PROPERTIES OF H:

1. New matrix is well conditioned: $L_H^+ L$

2. Linear systems in L_H can be solved quickly by Gaussian elimination

Combining Representation and Iteration: Combinatorial Preconditioning

$$Lv = \chi$$

Gradient methods fail when condition number is large.

IDEA: modify system to improve condition number

$$L_H^+ L v = L_H^+ \chi$$

where H is a **preconditioner** graph.

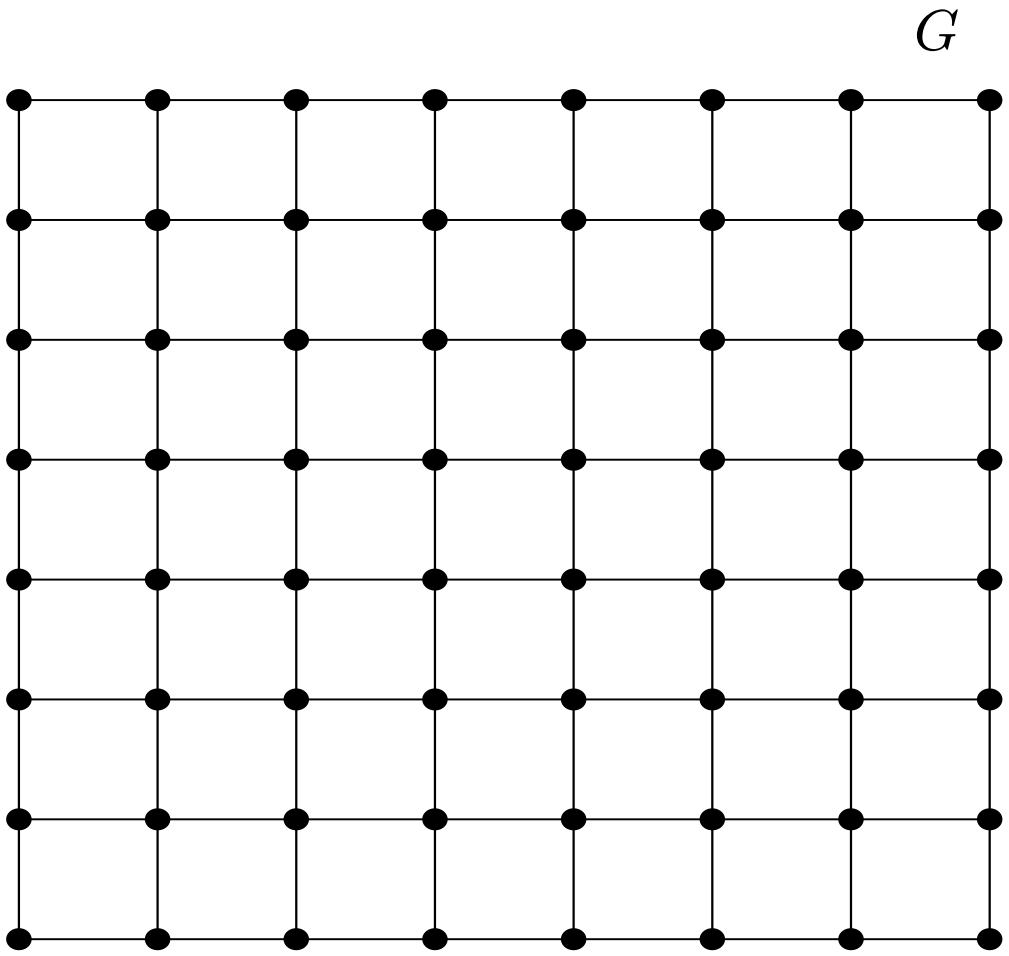
DESIRED PROPERTIES OF H:

1. New matrix is well conditioned: $L_H^+ L$ **IMPROVES ITERATION COUNT**

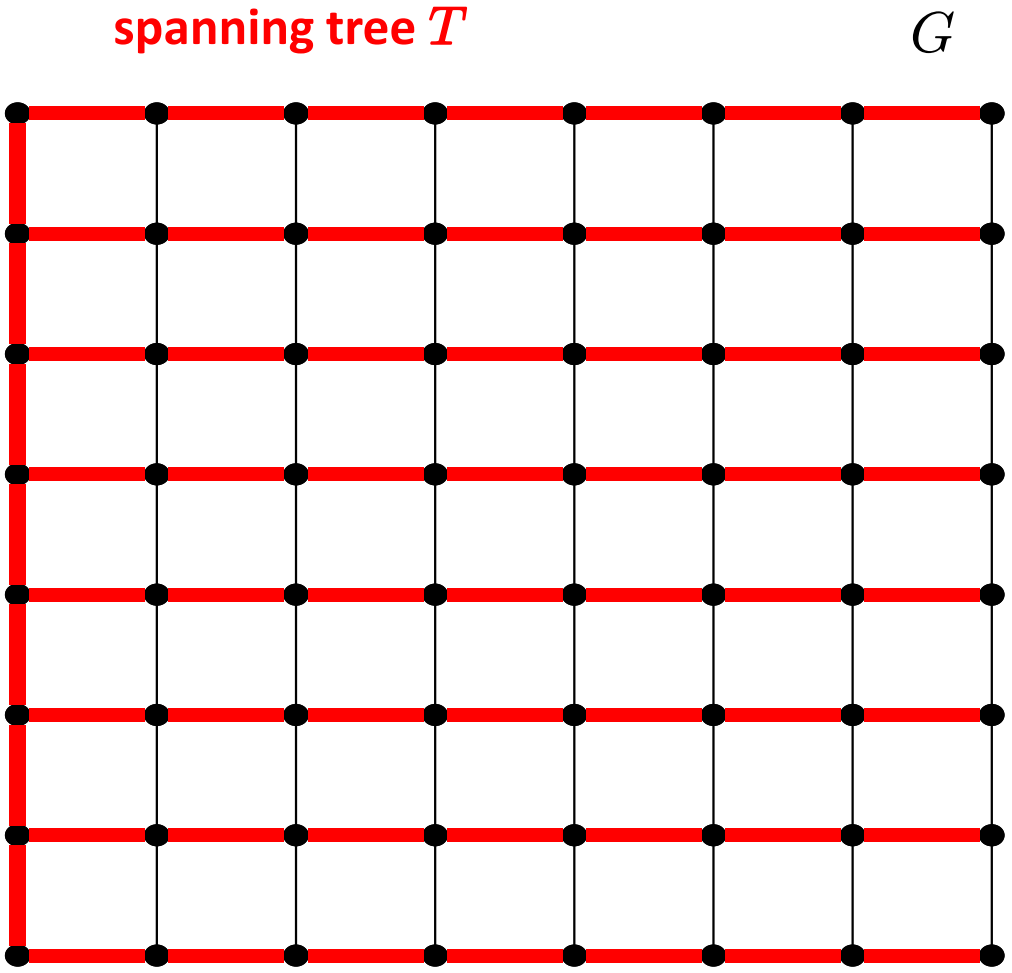
2. Linear systems in L_H can be solved quickly by Gaussian elimination

KEEPS ITERATIONS LINEAR TIME

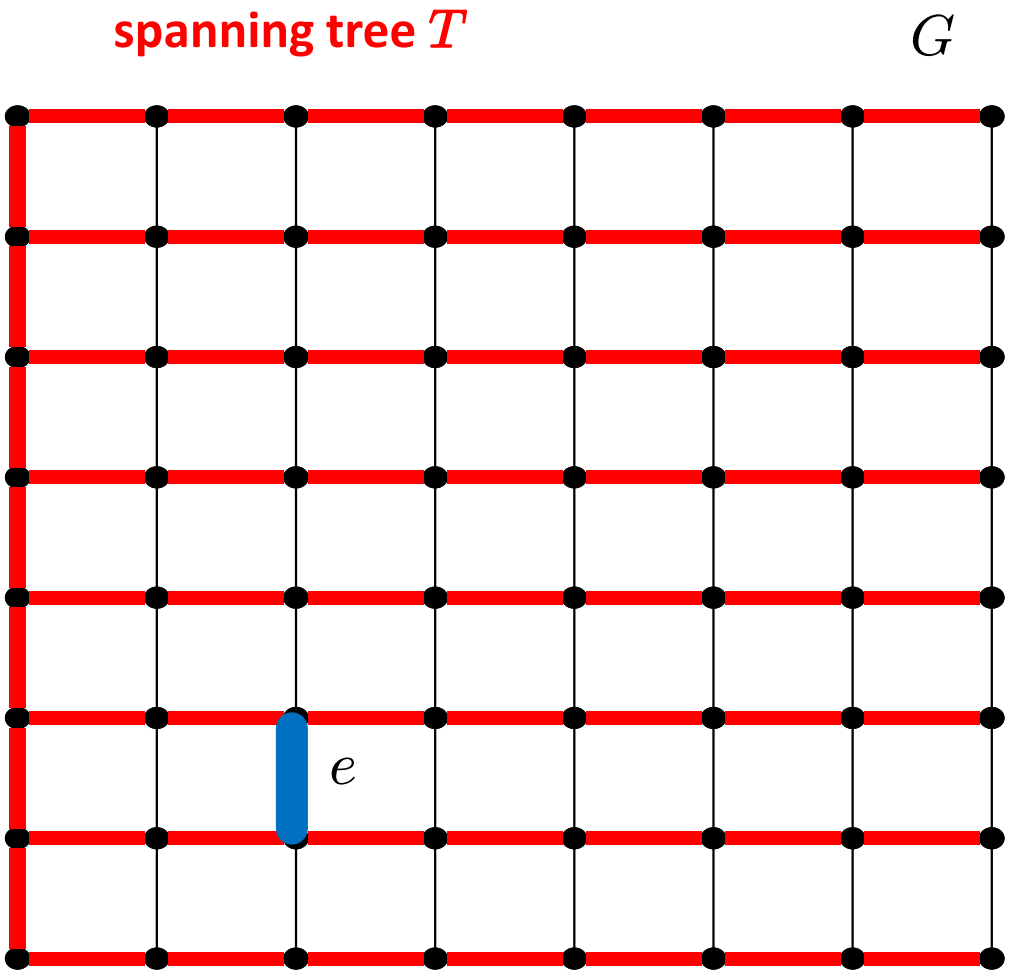
Preconditioning: Low-Stretch Trees



Preconditioning: Low-Stretch Trees



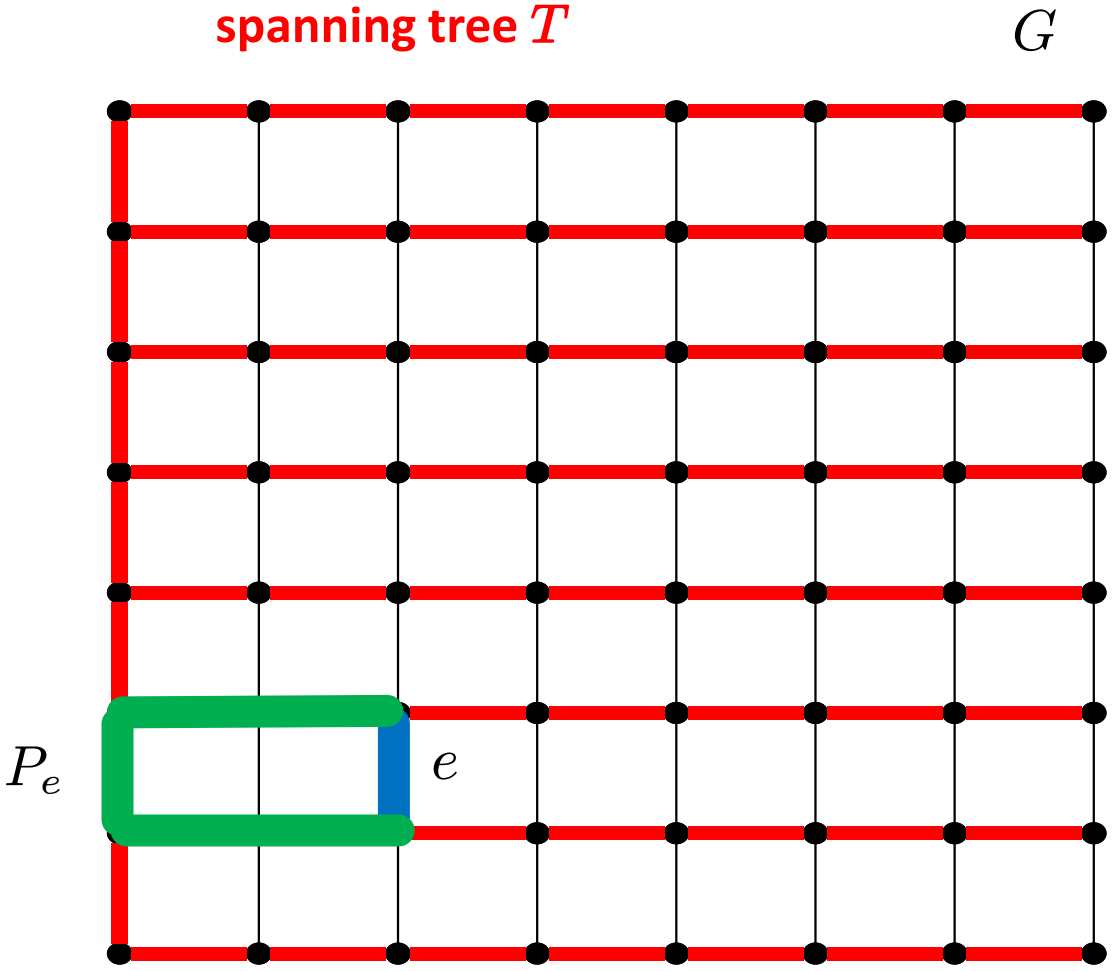
Preconditioning: Low-Stretch Trees



Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

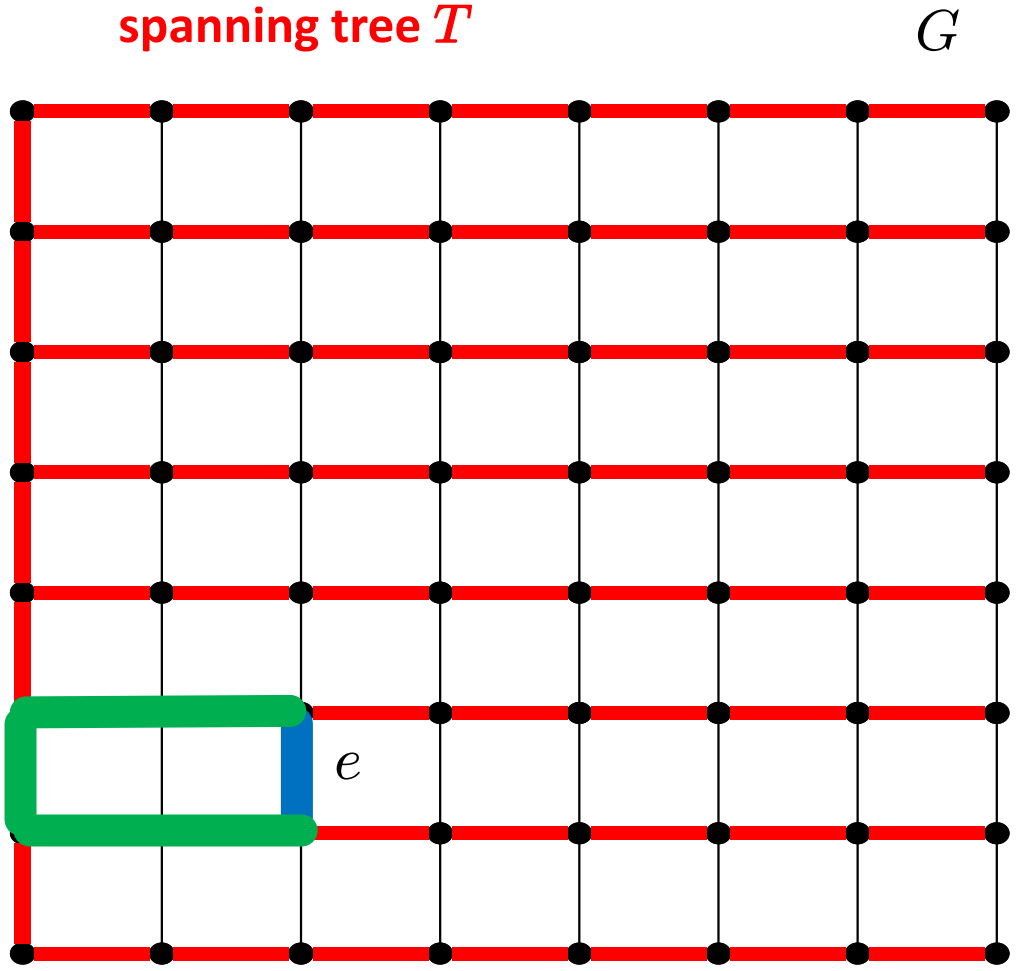
Stretch of e



Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

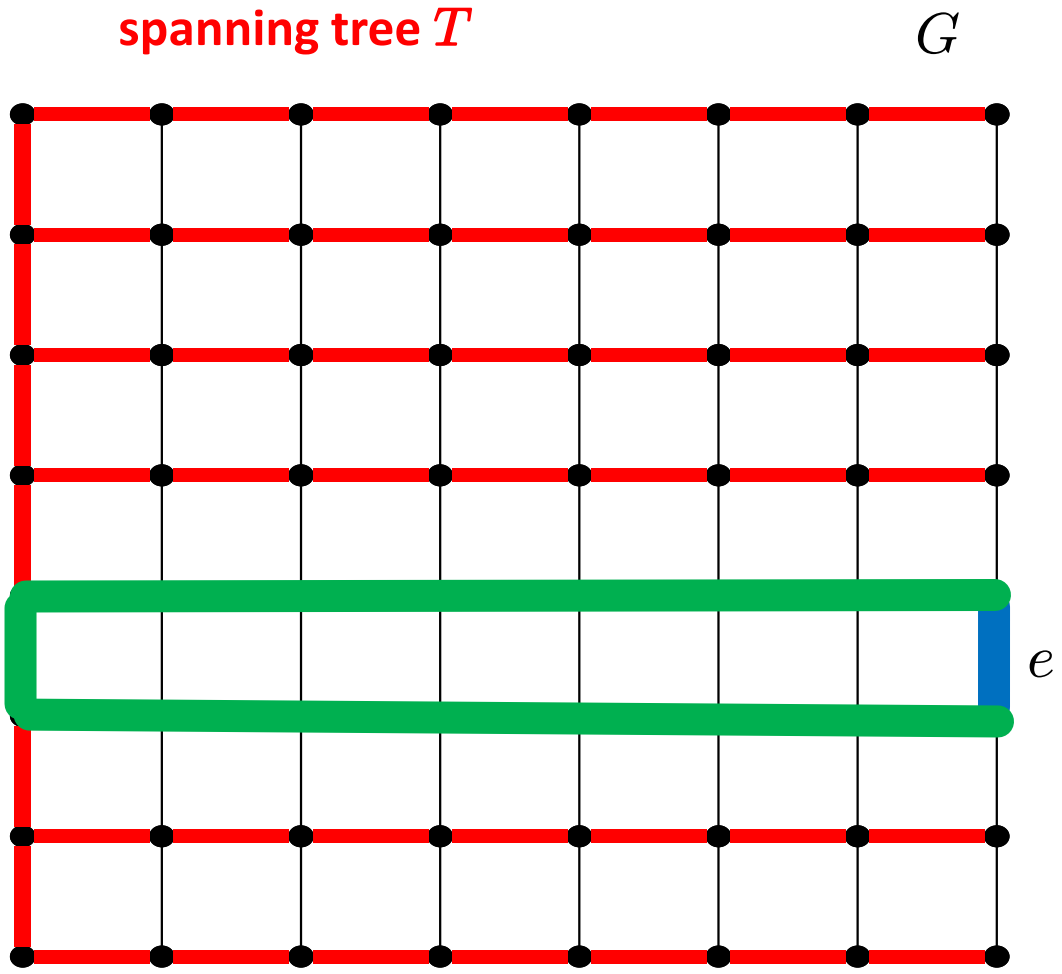
Stretch of e



Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

Stretch of e



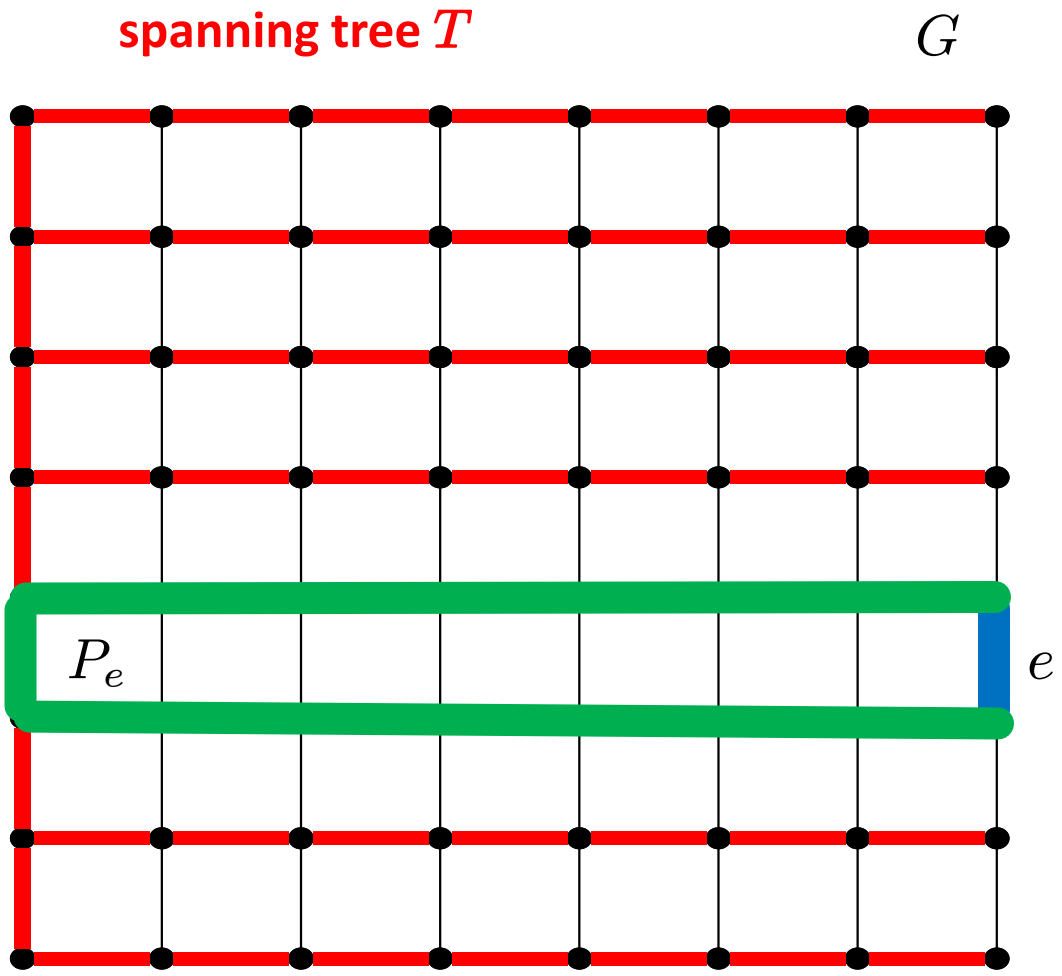
Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

Stretch of e

$$\mathrm{st}(T) = \sum_{e \in E} \mathrm{st}(e)$$

Total Stretch of e



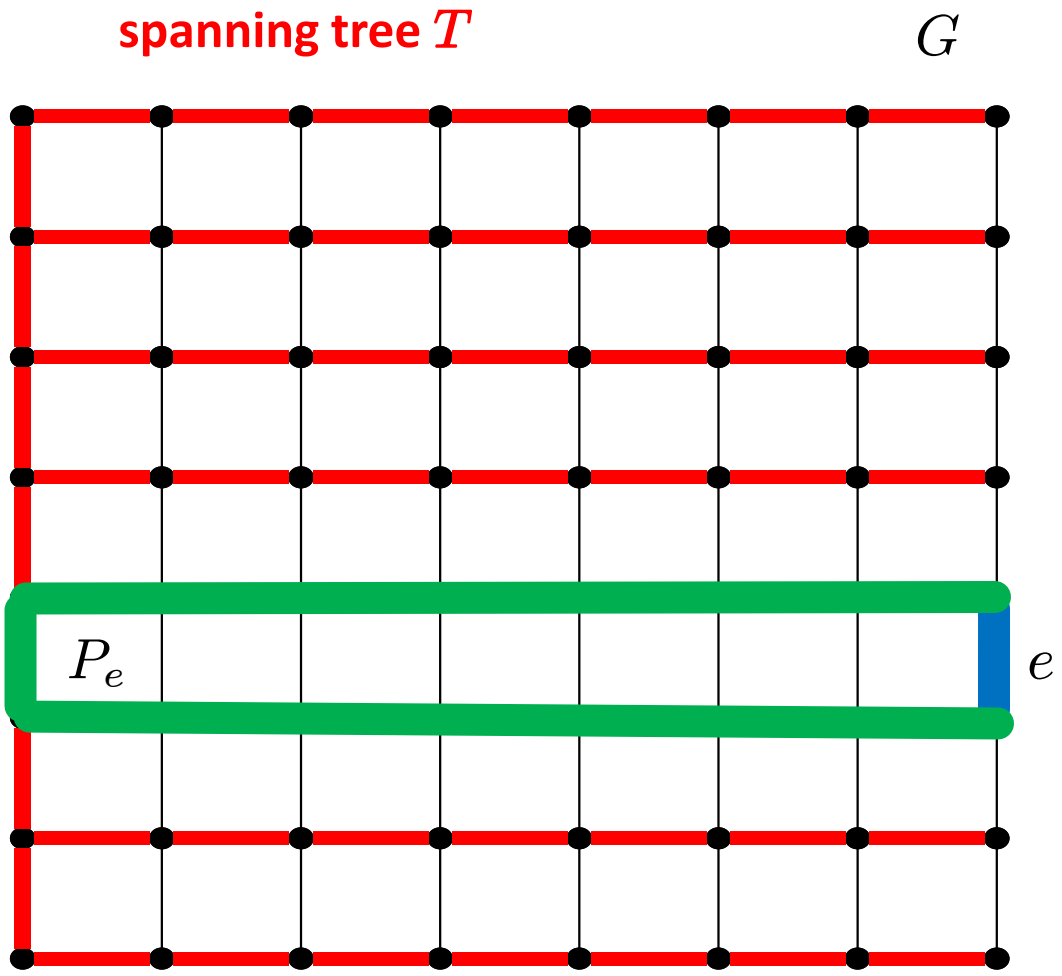
Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

Stretch of e

$$\mathrm{st}(T) = \sum_{e \in E} \mathrm{st}(e)$$

Total Stretch of e



$$\mathrm{st}(T) = O(n^{1.5})$$

Preconditioning: Low-Stretch Trees

$$\mathrm{st}(e) = \frac{1}{r_e} \sum_{e' \in P_e} r_{e'}$$

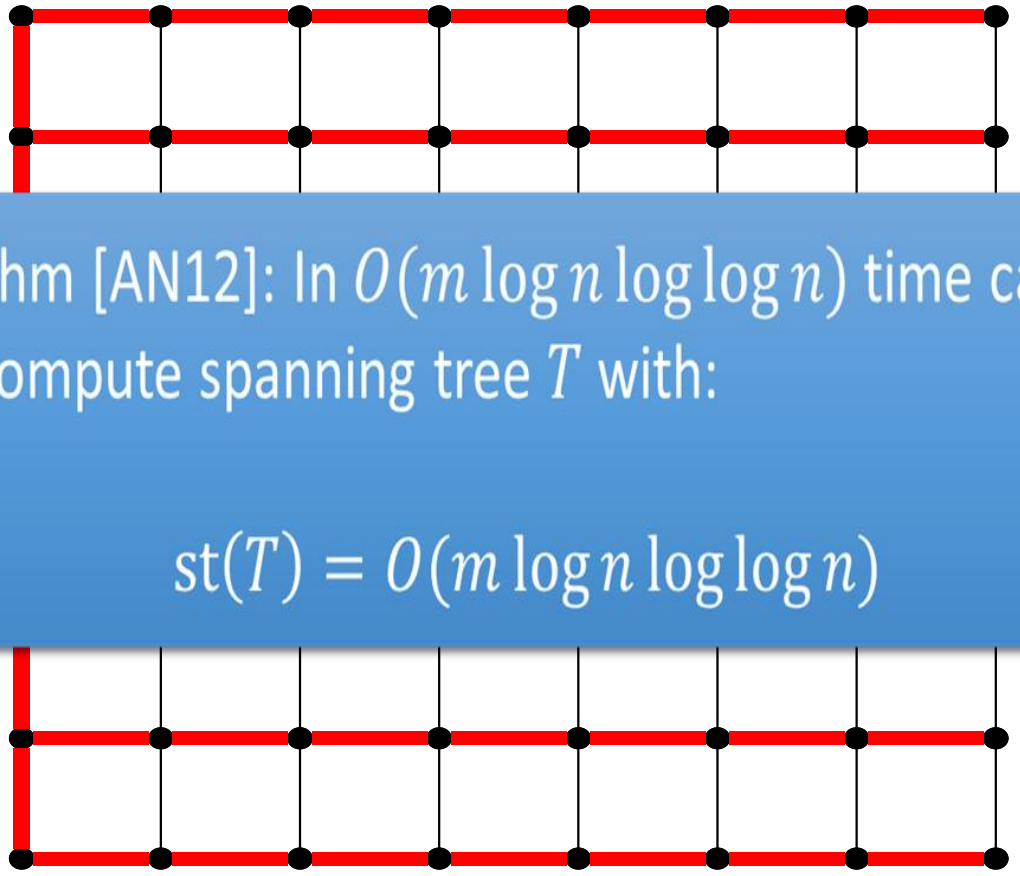
Stretch of e

$$\mathrm{st}(T) = \sum_{e \in E} \mathrm{st}(e)$$

Total Stretch of e

spanning tree T

G



Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

$$\mathrm{st}(T) = O(n^{1.5})$$

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

A: Can help to bound condition number of system preconditioned by T

$$1 \cdot I \preceq L_T^{+} L_G \preceq \mathrm{st}(T) \cdot I$$

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

A: Can help to bound condition number of system preconditioned by T

$$1 \cdot I \preceq L_T^{+} L_G \preceq \mathrm{st}(T) \cdot I \qquad \text{[SW09]}$$

Spanning tree property

Low-stretch-tree property

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

A: Can help to bound condition number of system preconditioned by T

$$1 \cdot I \preceq L_T^+ L_G \preceq \mathrm{st}(T) \cdot I \qquad \text{[SW09]}$$

EASY PROOF:

$$\lambda_{\max}\left(L_T^+ L_G\right) \cdot \ \mathrm{Tr}\left(L_T^+ L_G\right) = \sum_{e \in E} \chi_e^{\mathrm{T}} L_T^+ \chi_e = \mathrm{st}(T)$$

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

A: Can help to bound condition number of system preconditioned by T

$$1 \cdot I \preceq L_T^+ L_G \preceq \mathrm{st}(T) \cdot I \qquad \text{[SW09]}$$

EASY PROOF:

$$\lambda_{\max}\left(L_T^+ L_G\right) \cdot \ \mathrm{Tr}\left(L_T^+ L_G\right) = \sum_{e \in E} \chi_e^{\mathrm{T}} L_T^+ \chi_e = \mathrm{st}(T)$$

Trace bounds eigenvalue

Preconditioning: Low-Stretch Trees

Thm [AN12]: In $O(m \log n \log\log n)$ time can compute spanning tree T with:

$$\mathrm{st}(T) = O(m \log n \log\log n)$$

Q: How does this help?

A: Can help to bound condition number of system preconditioned by T

$$1 \cdot I \preceq L_T^+ L_G \preceq \mathrm{st}(T) \cdot I \qquad \text{[BH01]}$$

EASY PROOF:

$$\lambda_{\max}\left(L_T^+ L_G\right) \cdot \ \mathrm{Tr}\left(L_T^+ L_G\right) = \sum_{e \in E} \chi_e^{\mathrm{T}} L_T^+ \chi_e = \mathrm{st}(T)$$

Tree resistance is path length

Preconditioning: Partial Results

1. Preconditioning by low-stretch spanning trees[BH01]

$$\text{RunTime} = \tilde{O}(\underbrace{\sqrt{m \log n} \log\left(\tfrac{n}{\epsilon}\right)}) \cdot [O(m) + O(n)]$$

Condition number

Preconditioning: Partial Results

1. Preconditioning by low-stretch spanning trees[BH01]

$$\text{RunTime} = \tilde{O}(\underline{\sqrt{m \log n} \log\left(\tfrac{n}{\epsilon}\right)}) \cdot \underline{[O(m) + O(n)]}$$

Condition number

Multiplication by $L_T^+ L$

Preconditioning: Partial Results

1. Preconditioning by low-stretch spanning trees[BH01]

$$\text{RunTime} = \tilde{O}(\underline{\sqrt{m \log n}} \log\left(\tfrac{n}{\epsilon}\right)) \cdot \underline{[O(m) + O(n)]}$$

Condition number — Multiplication by $L_T^+ L$

2. Improved analysis using Conjugate Gradient and Trace bound [SW09]

$$\text{RunTime} = \tilde{O}(m^{4/3} \text{polylog } n)$$

NB: Low-stretch is a trace bound, stronger than eigenvalue bound!

Recursive Preconditioning: Spielman-Teng and Koutis-Miller-Peng

Nearly-linear-time algorithms at last: [ST04], [KMP10],[KMP11]

IDEA: Low-stretch trees do not provide good enough condition number.
Use better preconditioner graphs

Recursive Preconditioning: Spielman-Teng and Koutis-Miller-Peng

Nearly-linear-time algorithms at last: [ST04], [KMP10],[KMP11]

-

IDEA: Low-stretch trees do not provide good enough condition number.
Use better preconditioner graphs

$$L_{G_1}^{+} L v = L_{G_1}^{+} \chi$$

PROBLEM: Hard to solve system for G_1.

SOLUTION: Solve recursively.

Recursive Preconditioning: Spielman-Teng and Koutis-Miller-Peng

Nearly-linear-time algorithms at last: [ST04], [KMP10],[KMP11]

-

IDEA: Low-stretch trees do not provide good enough condition number.
Use better preconditioner graphs

$$L_{G_1}^{+} L v = L_{G_1}^{+} \chi$$

PROBLEM: Hard to solve system for G_1.

SOLUTION: Solve recursively.

$$L_{G_2}^{+} L_{G_1} x = L_{G_2}^{+} b$$

MAIN IDEA: At every recursive level, $G_i$becomes smaller as some low-degree vertices are eliminated via **Gaussian elimination**.

Our Algorithm

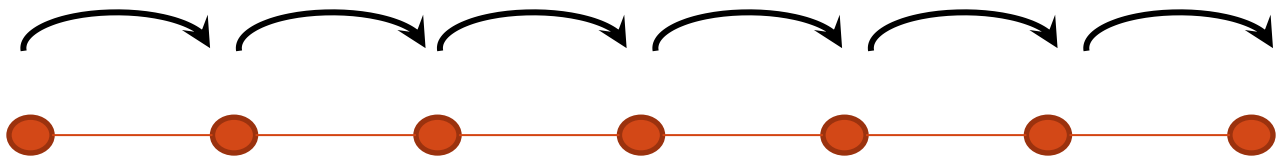
Our Algorithm
(at last)

Choosing the Right Representation
Solve for the Flow

All algorithms discussed so far aim to solve voltage problem .

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot v^{\mathrm{T}} L v - v^{\mathrm{T}} \chi$$

This is particularly problematic for gradient-based methods:



CURSE OF LONG PATHS

Choosing the Right Representation
Solve for the Flow

All algorithms discussed so far aim to solve voltage problem .

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot v^{\mathrm{T}} L v - v^{\mathrm{T}} \chi$$

Our algorithm targets the minimum-energy flow problem:

$$\begin{aligned} \min \quad & f^T R f \\ \text{s.t} \quad & B^T f = \chi \end{aligned}$$

Choosing the Right Representation Solve for Electrical Flow

All algorithms discussed so far aim to solve voltage problem .

$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot v^{\mathrm{T}} L v - v^{\mathrm{T}} \chi$$

Our algorithm targets the minimum-energy flow problem:

$$\min \quad f^T R f \qquad \text{Minimize energy}$$

$$\text{s.t} \quad B^T f = \chi \qquad \text{Route correct net flow}$$

Choosing the Right Representation
Solve for Electrical Flow

All algorithms discussed so far aim to solve voltage problem .

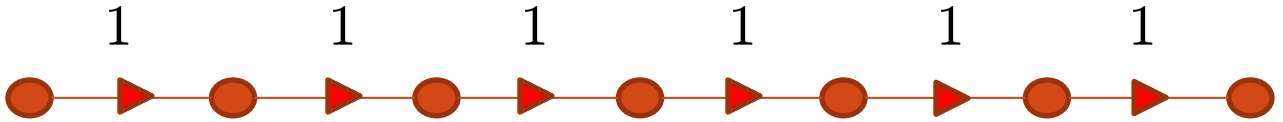
$$\min_{x \perp \vec{1}} \quad \tfrac{1}{2} \cdot v^{\mathrm{T}} L v - v^{\mathrm{T}} \chi$$

Our algorithm targets the minimum-energy flow problem:

$$\min \quad f^T R f \qquad \text{Minimize energy}$$

$$\text{s.t} \quad B^T f = \chi \qquad \text{Route correct net flow}$$

This choice opens up different representational questions:



Could this be a flow path in our basic representation?

Optimality Conditions for Flow Problem

1. Ohm's Law:

$$\exists v: \qquad f = R^{-1}Bv$$

2. Kirchoff's Conservation Law:

$$B^T f = \chi$$

Optimality Conditions for Flow Problem

1. Ohm's Law:

$$\exists v: \qquad f = R^{-1}Bv$$

2. Kirchoff's Conservation Law:

$$B^T f = \chi$$

We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

$$\sum_{e \in C} r_e f_e = 0$$

Flow-induced voltage drop along cycle is 0.

Optimality Conditions for Flow Problem

1. Ohm's Law:

$$\exists v: \qquad f = R^{-1}Bv$$

2. Kirchoff's Conservation Law:

$$B^T f = \chi$$

We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

$$\sum_{e \in C} r_e f_e = 0$$

Fact: Ohm's Law $\leftrightarrow$ Kirchoff's Cycle Law

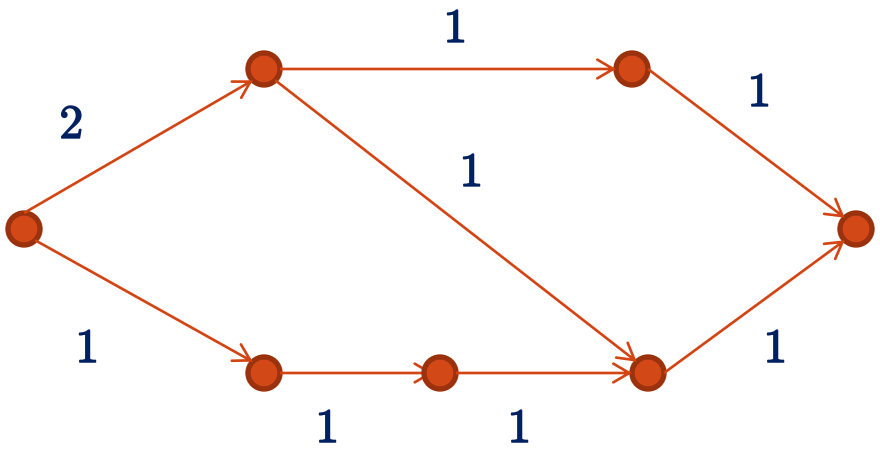
Optimality Conditions for Flow Problem

We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

$$\sum_{e \in C} r_e f_e = 0$$

Fact: Ohm's Law $\leftrightarrow$ Kirchoff's Cycle Law

Flow values
Unit resistances



Optimality Conditions for Flow Problem

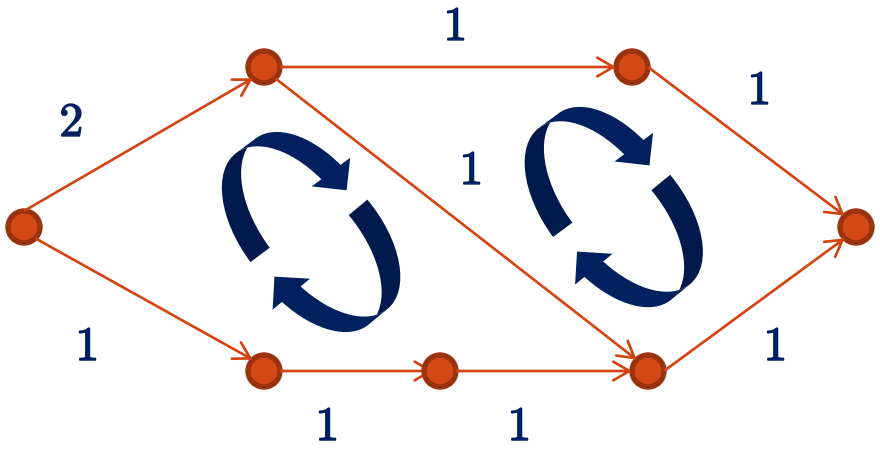
We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

$$\sum_{e \in C} r_e f_e = 0$$

Fact: Ohm's Law $\leftrightarrow$ Kirchoff's Cycle Law

Flow values
Unit resistances

Obeys KCL



Optimality Conditions for Flow Problem

We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

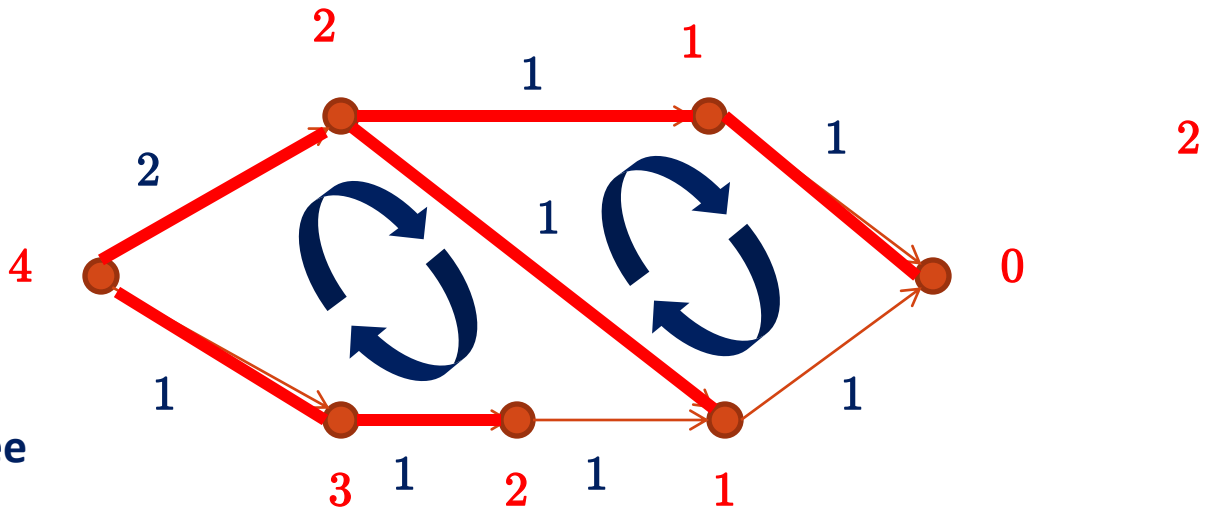
$$\sum_{e \in C} r_e f_e = 0$$

Fact: Ohm's Law $\leftrightarrow$ Kirchoff's Cycle Law

Flow values
Unit resistances

Obeys KCL

Set voltages using spanning tree



Optimality Conditions for Flow Problem

We eliminate dependence on voltages in Ohm's Law:
Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

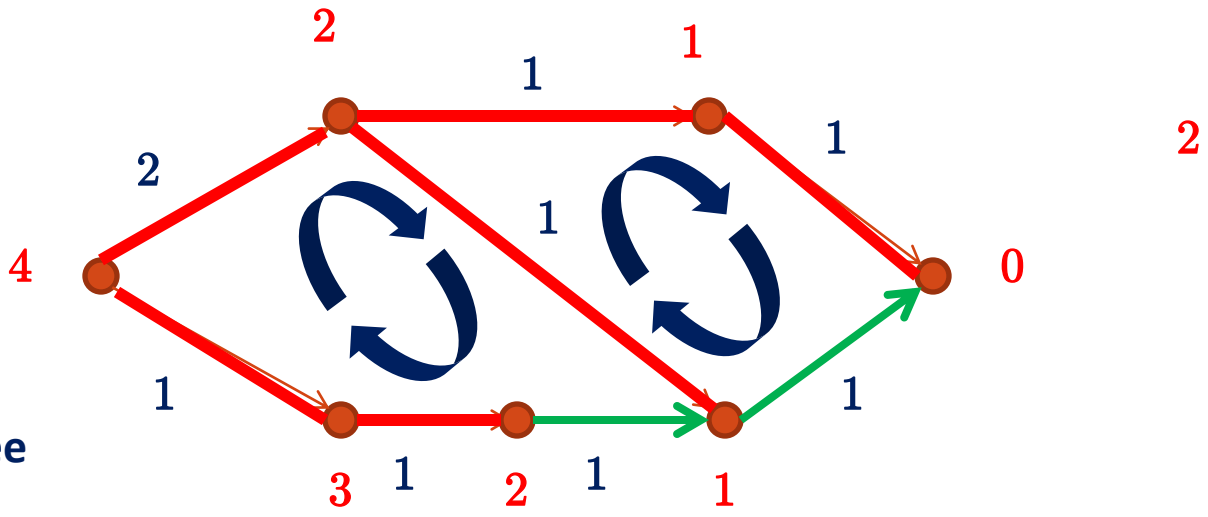
$$\sum_{e \in C} r_e f_e = 0$$

Fact: Ohm's Law $\leftrightarrow$ Kirchoff's Cycle Law

Flow values
Unit resistances

Obeys KCL

Set voltages using spanning tree



KCL ensures that off-tree edges respect Ohm's Law

Algorithmic Approach

1. Kirchoff's Cycle Law: For any cycle C in G, the optimal electrical flow has

$$\sum_{e \in C} r_e f_e = 0$$

ITERATIVELY FIX BY ADDING/REMOVING FLOW ALONG CYCLES

2. Kirchoff's Conservation Law:

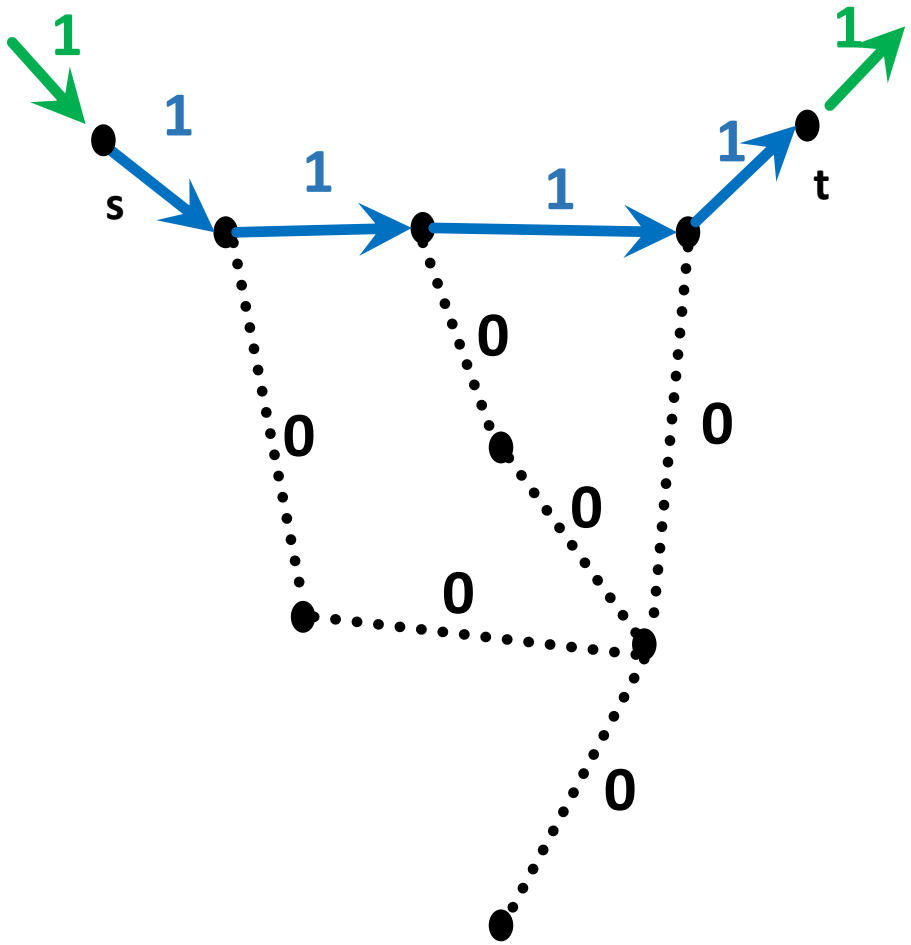
$$B^T f = \chi$$

MAINTAIN SATISFIED

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

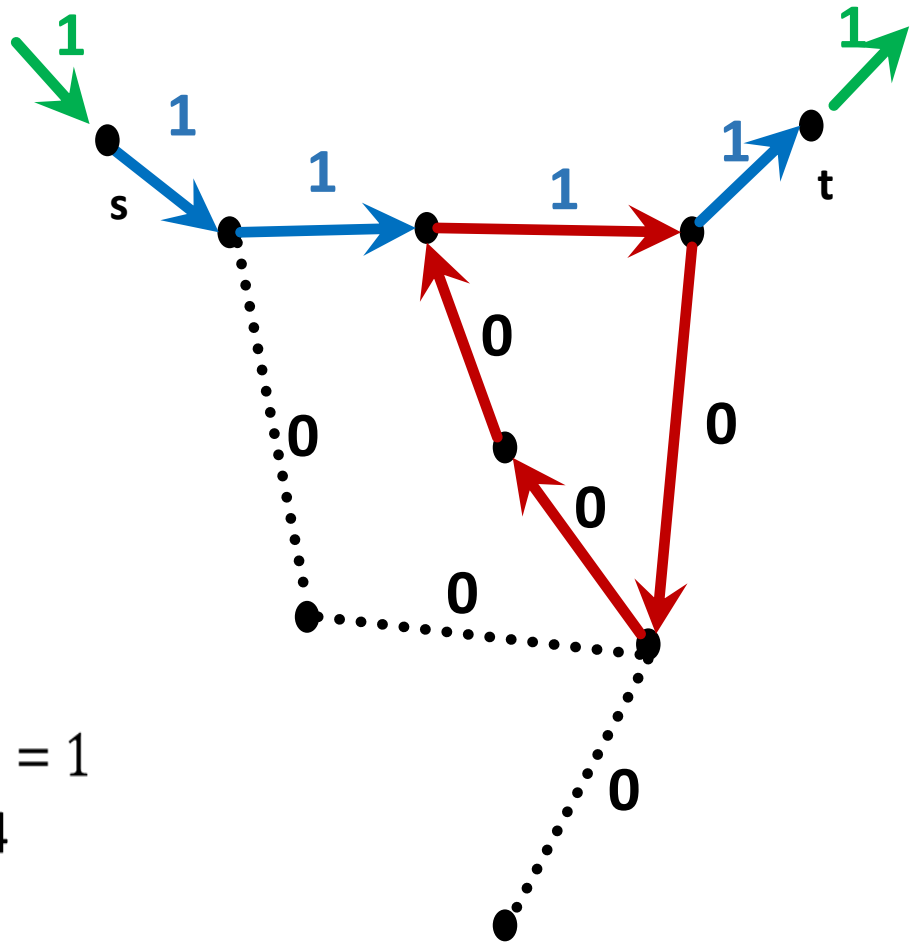
Improve:
- Pick a cycle



Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

Improve:
- Pick a cycle
- Fix it

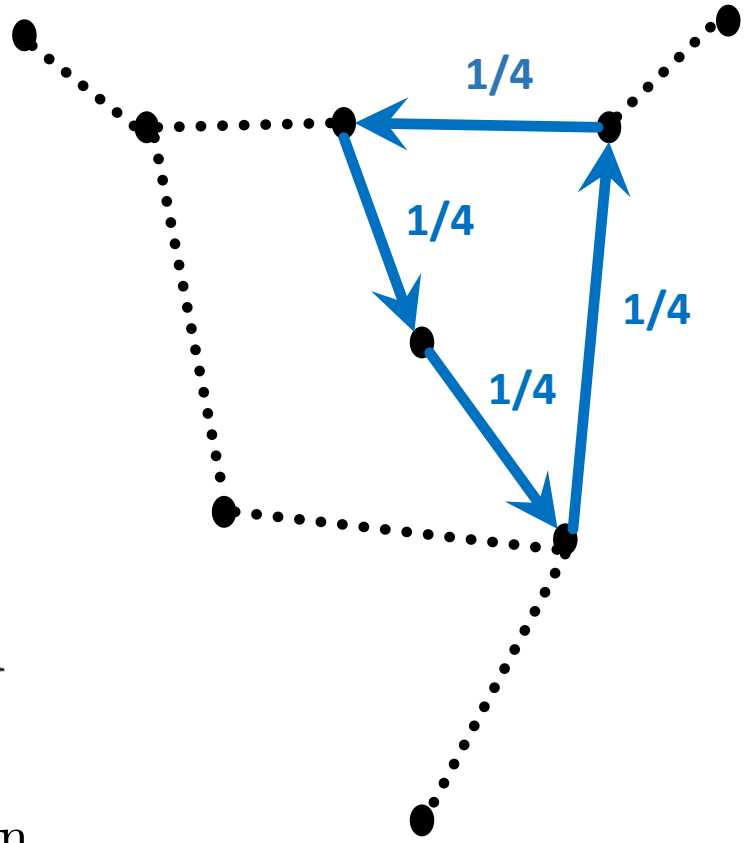


- Potential Drop: $\Delta = \sum_{e \in C} r_e \vec{f}(e) = 1$
- Cycle Resistance: $R = \sum_{e \in C} r_e = 4$

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

Improve:
- Pick a cycle
- Fix it



- Potential Drop: $\Delta = \sum_{e \in C} r_e \vec{f}(e) = 1$
- Cycle Resistance: $R = \sum_{e \in C} r_e = 4$

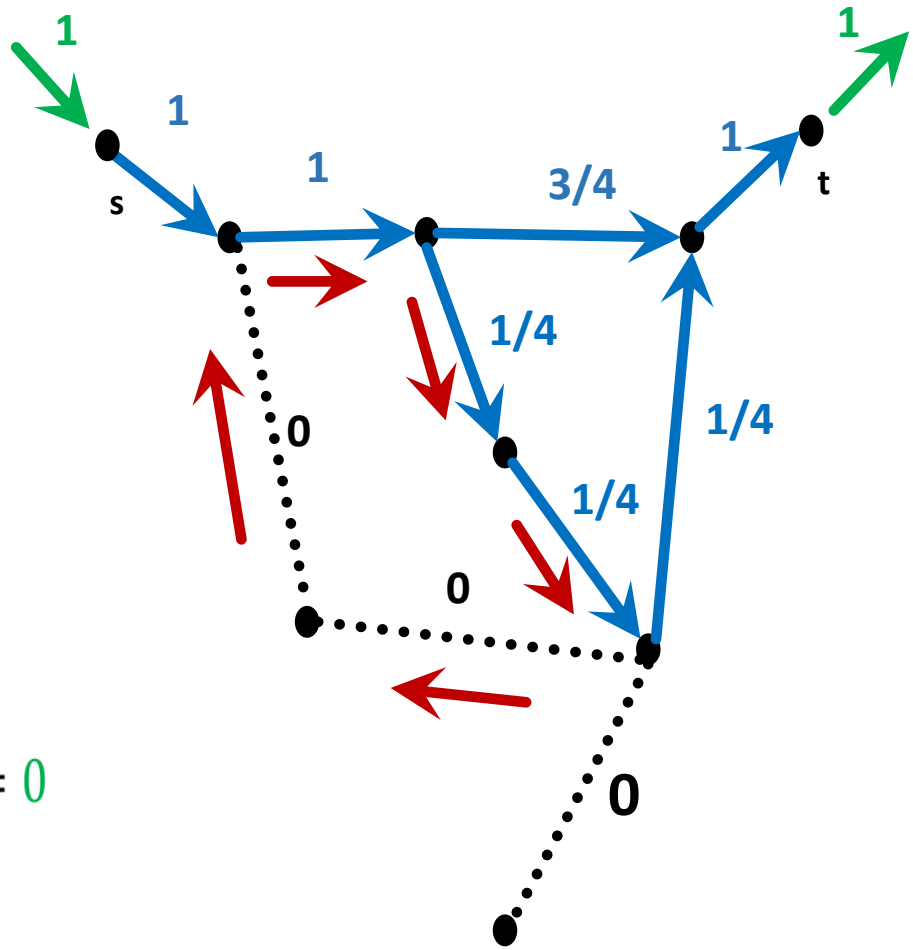
Send $\frac{\Delta}{R}$ flow in opposite direction

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

Improve:

- Pick a cycle
- Fix it
- Repeat



- Potential Drop: $\Delta = \sum_{e \in C} r_e \vec{f}(e) = 0$
- Cycle Resistance: $R = \sum_{e \in C} r_e = 4$

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

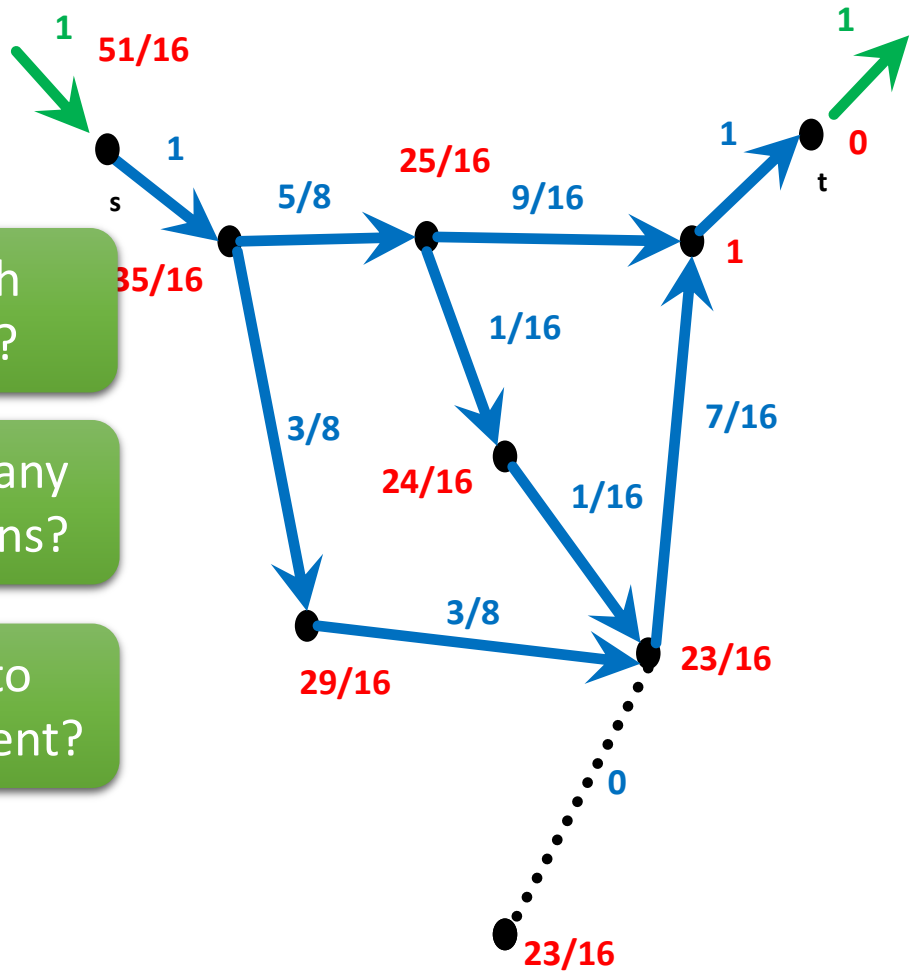
Improve:
- Pick a cycle
- Fix it
- Repeat

Which cycle?

How many iterations?

How to implement?

Output: Round to voltages v



Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

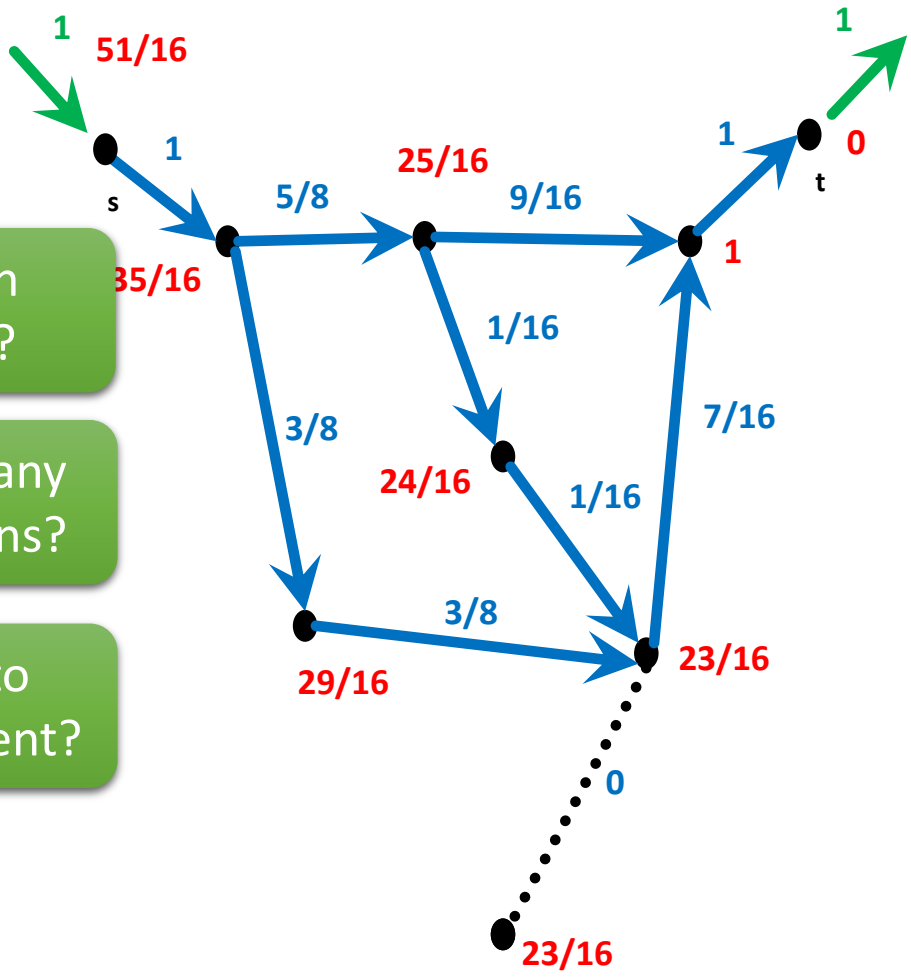
Improve:
- Pick a cycle
- Fix it
- Repeat

Which cycle?

How many iterations?

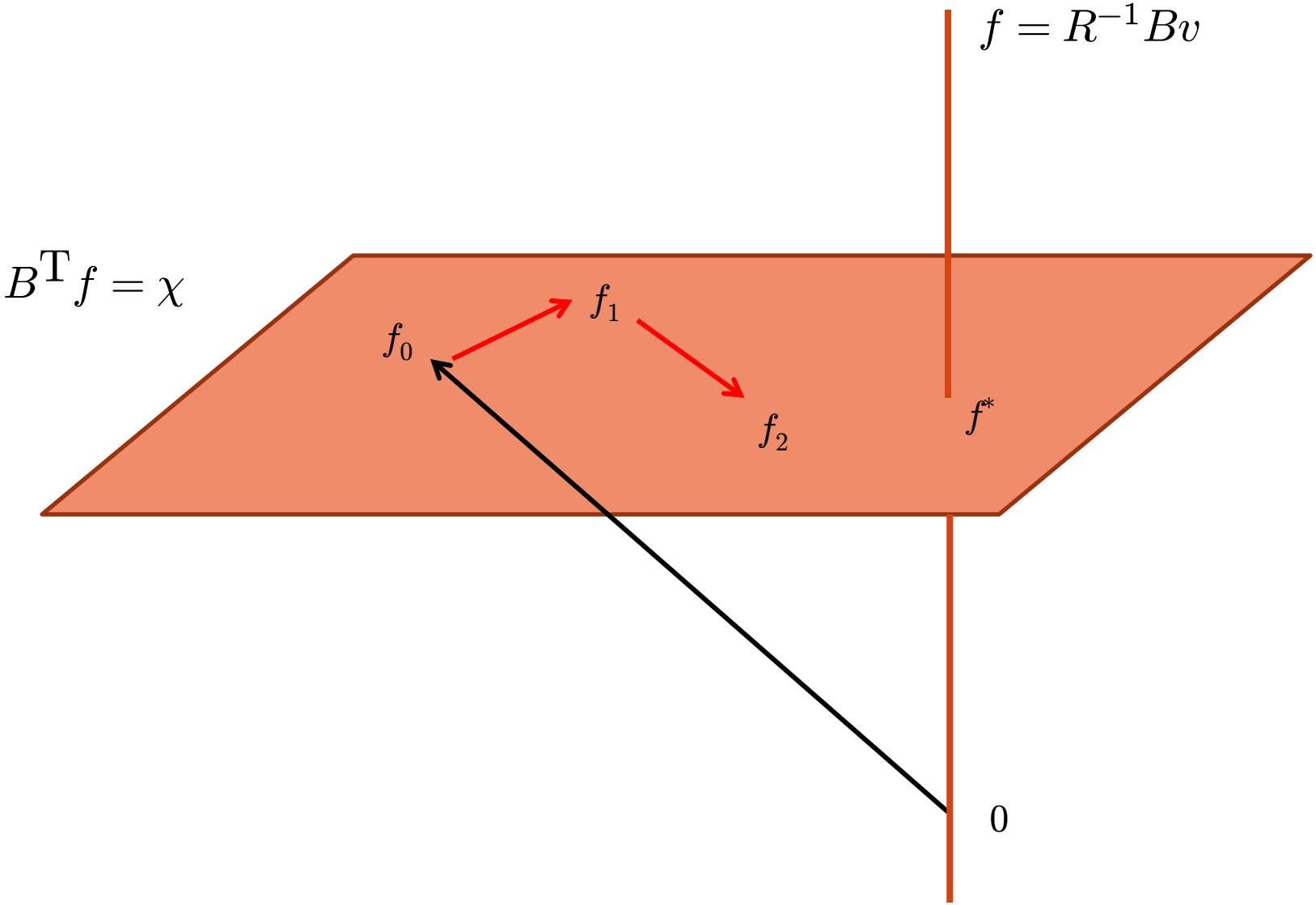
How to implement?

Output: Round to voltages v



IDEA: A nearly-linear number of cheap (i.e. $\mathrm{O}(\log n)$ iterations!

Choosing the Right Representation: Cycle Space



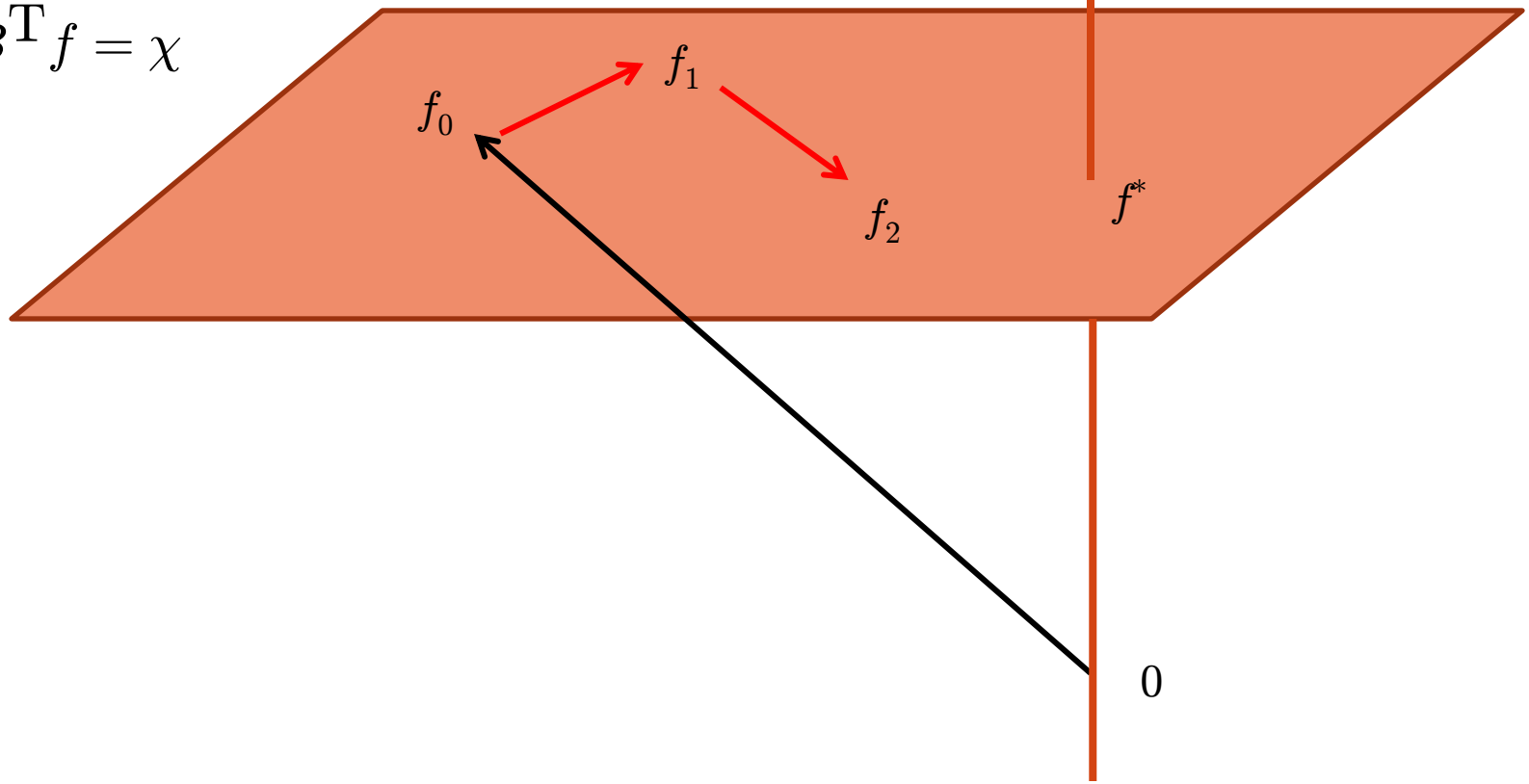
Choosing the Right Representation: Cycle Space

An affine translation of span of cycles (cycle space C_G)

$$C_G = \{c : B^{\mathrm{T}} c = 0\}$$

$$B^{\mathrm{T}} f = \chi$$

$$f = R^{-1} B v$$



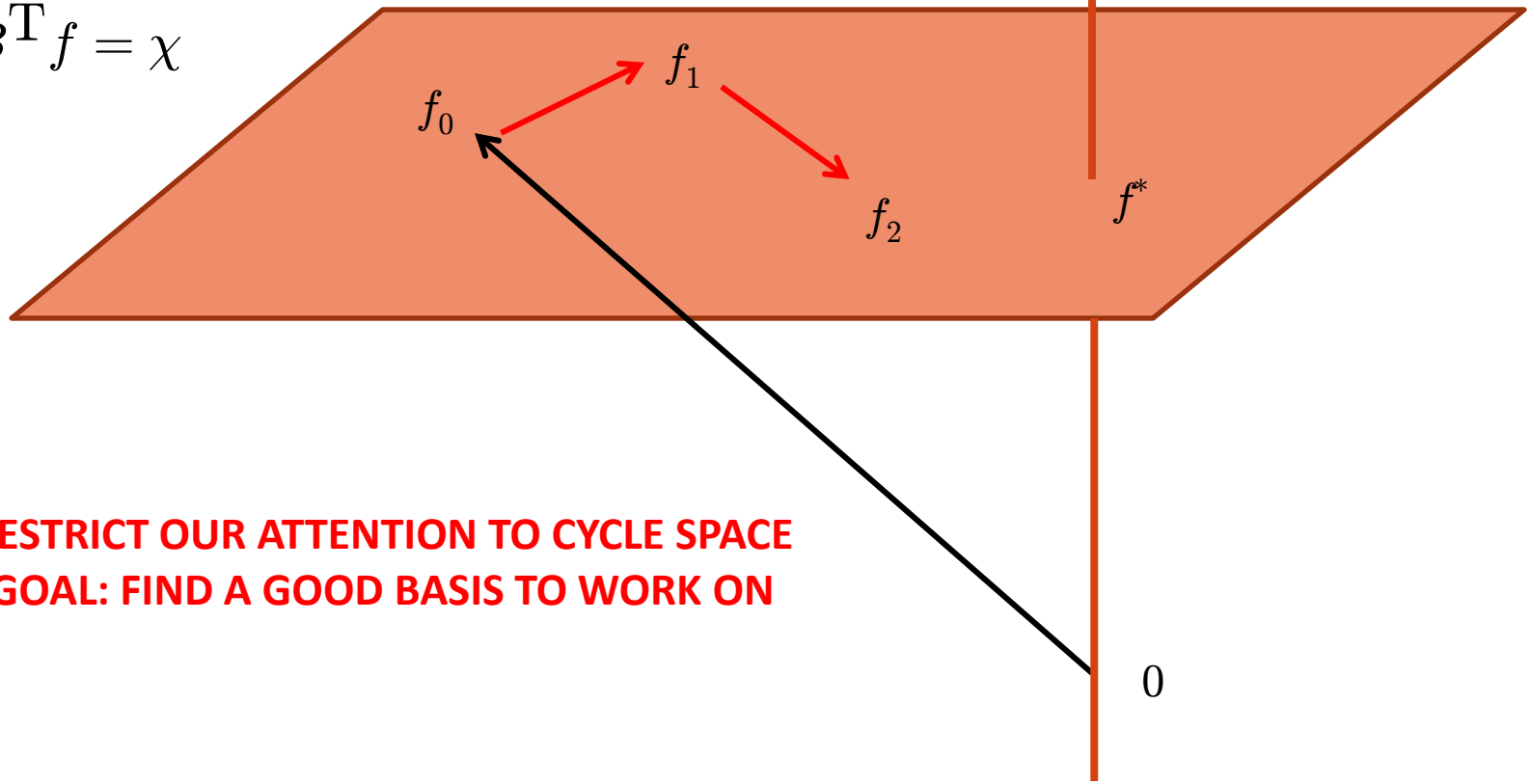
Choosing the Right Representation: Cycle Space

An affine translation of span of cycles (cycle space C_G)

$$C_G = \{c : B^{\mathrm{T}} c = 0\}$$

$$B^{\mathrm{T}} f = \chi$$

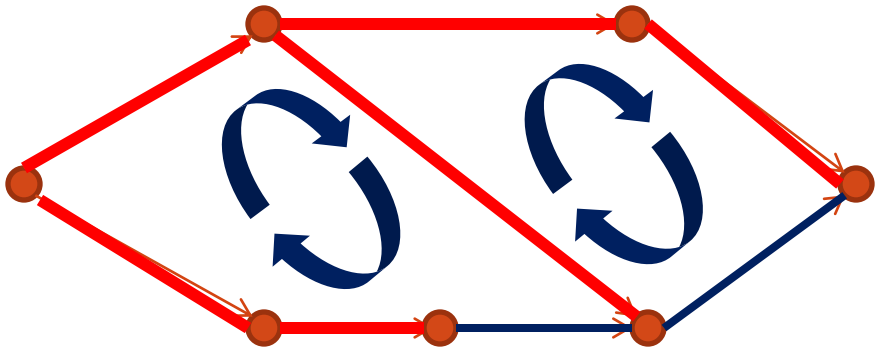
$$f = R^{-1} Bv$$



RESTRICT OUR ATTENTION TO CYCLE SPACE
GOAL: FIND A GOOD BASIS TO WORK ON

Choosing the Right Representation: Basis of Cycle Space

Fact:
Fundamental cycles of spanning tree give a basis of cycle space

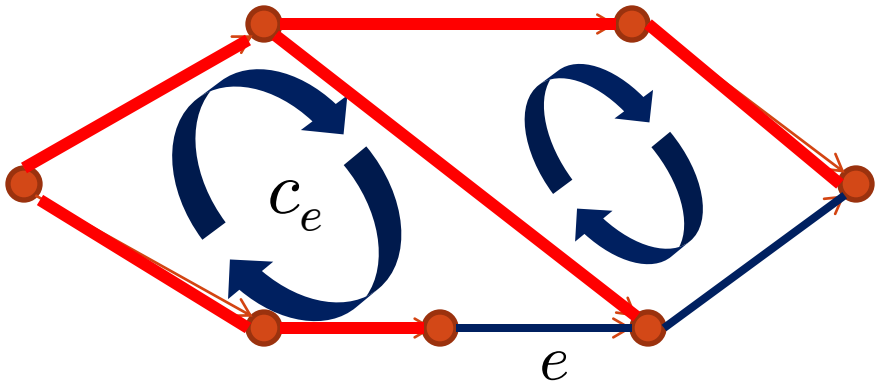


Fix spanning tree **T**

Choosing the Right Representation: Basis of Cycle Space

Fact:

Fundamental cycles of spanning tree give a basis of cycle space



Fix spanning tree **T**

Our New Representation

ORIGINAl PROBLEM

$$\forall c \in C_G : \quad c^T R(f_0 + f) = 0$$

$$B^T f = 0$$

Use spanning tree T
basis for C_G

$$\forall e \in E \setminus T : \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

Our New Representation

ORIGINAl PROBLEM

$$\forall c \in C_G : \quad c^T R(f_0 + f) = 0$$
$$B^T f = 0$$

Use spanning tree T
basis for C_G

$$\forall e \in E \setminus T : \quad c_e^T R(f_0 + f) = 0 \qquad m - n + 1 \text{ constraints}$$
$$f = \sum_{e \in E \setminus T} f_e c_e \qquad m - n + 1 \text{ variables}$$

Our New Representation

ORIGINAl PROBLEM

$$\forall c \in C_G : \quad c^T R(f_0 + f) = 0$$
$$B^T f = 0$$

Use spanning tree T basis for C_G

$$\forall e \in E \setminus T : \quad c_e^T R(f_0 + f) = 0 \qquad m - n + 1 \text{ constraints}$$
$$f = \sum_{e \in E \setminus T} f_e c_e \qquad m - n + 1 \text{ variables}$$

Iteratively fix fundamental cycles of T

Our New Representation

ORIGINAl PROBLEM

$$\forall c \in C_G : \quad c^T R(f_0 + f) = 0$$
$$B^T f = 0$$

Use spanning tree T basis for C_G

$$\forall e \in E \setminus T : \quad c_e^T R(f_0 + f) = 0 \qquad m - n + 1 \text{ constraints}$$
$$f = \sum_{e \in E \setminus T} f_e c_e \qquad m - n + 1 \text{ variables}$$

WHAT ITERATIVE METHOD IS THIS? Iteratively fix fundamental cycles of T

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

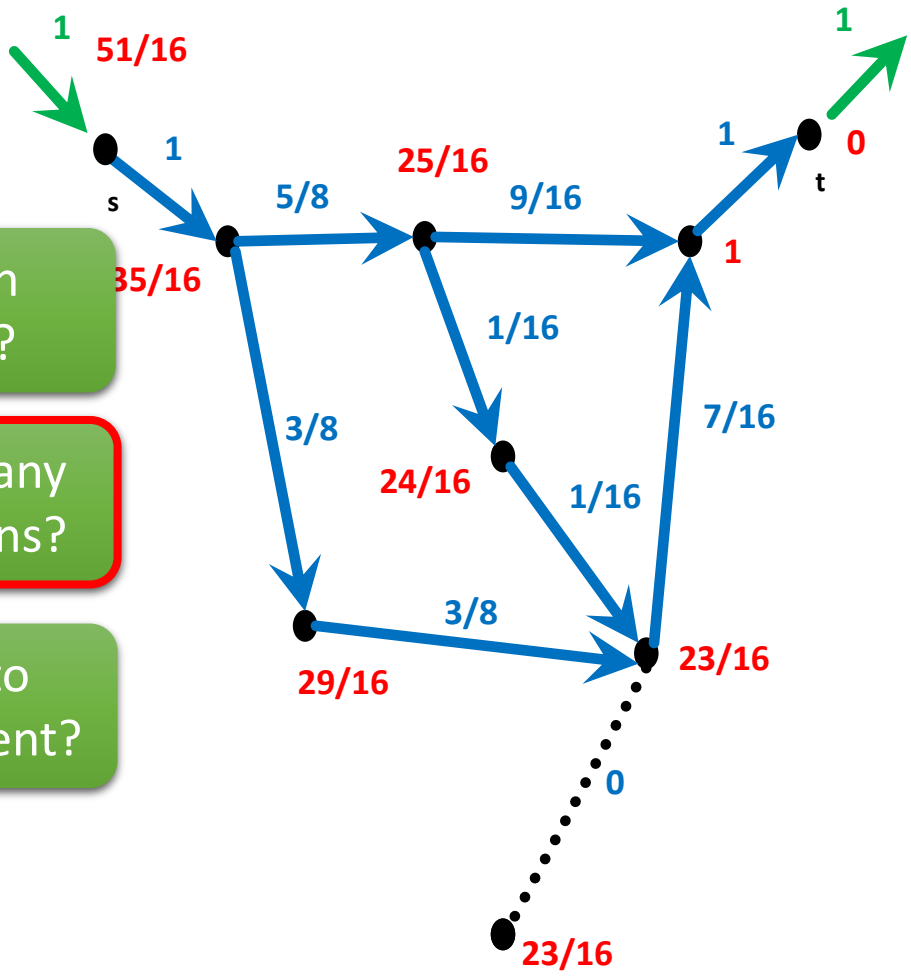
Improve:
- Pick a cycle
- Fix it
- Repeat

Which cycle?

How many iterations?

How to implement?

Output: Round to voltages v



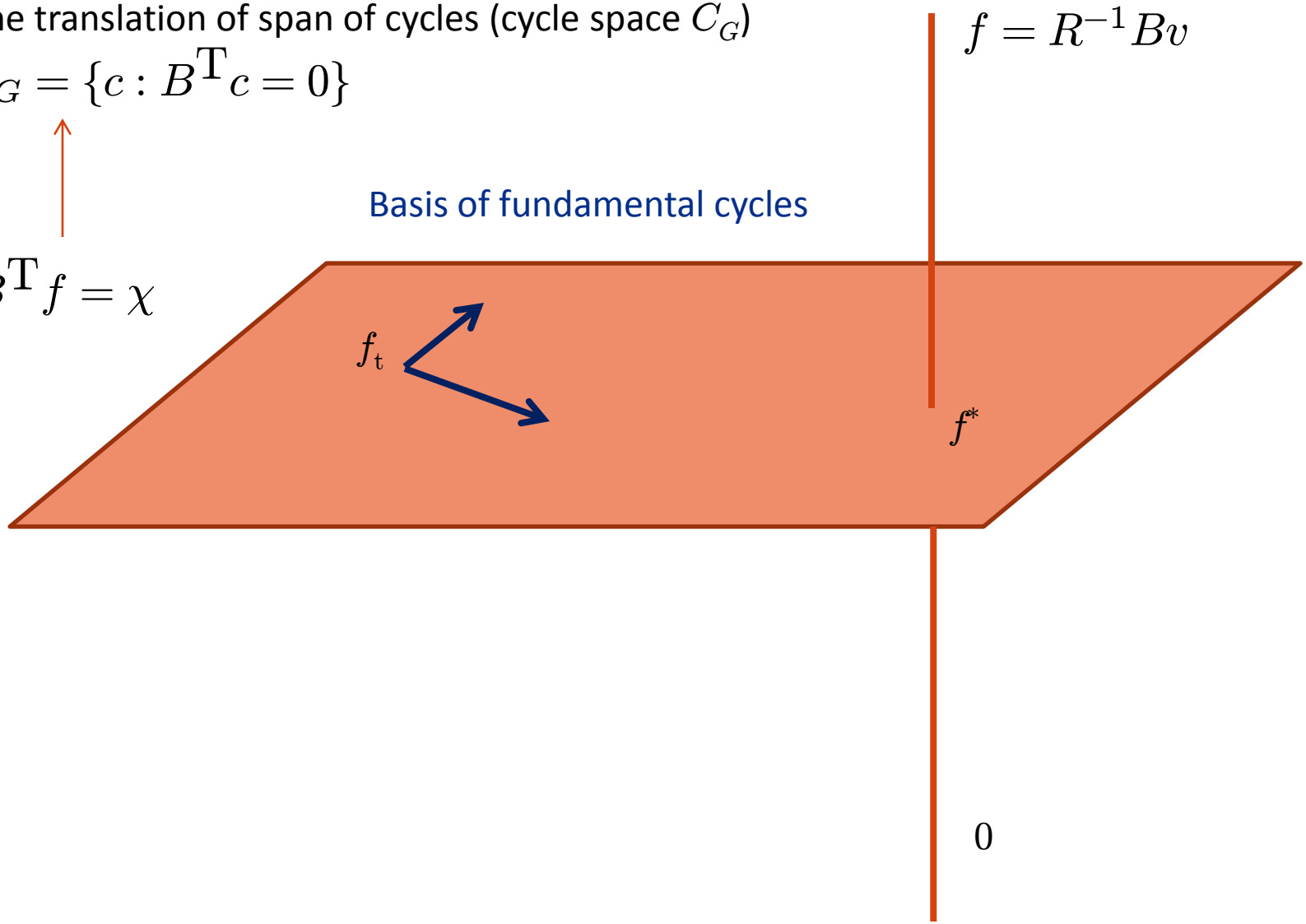
IDEA: A nearly-linear number of cheap (i.e. $\mathrm{O}(\log n)$ iterations!

Choosing the Right Iteration Gradient Descent?

An affine translation of span of cycles (cycle space C_G)

$$C_G = \{c : B^{\mathrm{T}} c = 0\}$$

$$B^{\mathrm{T}} f = \chi$$



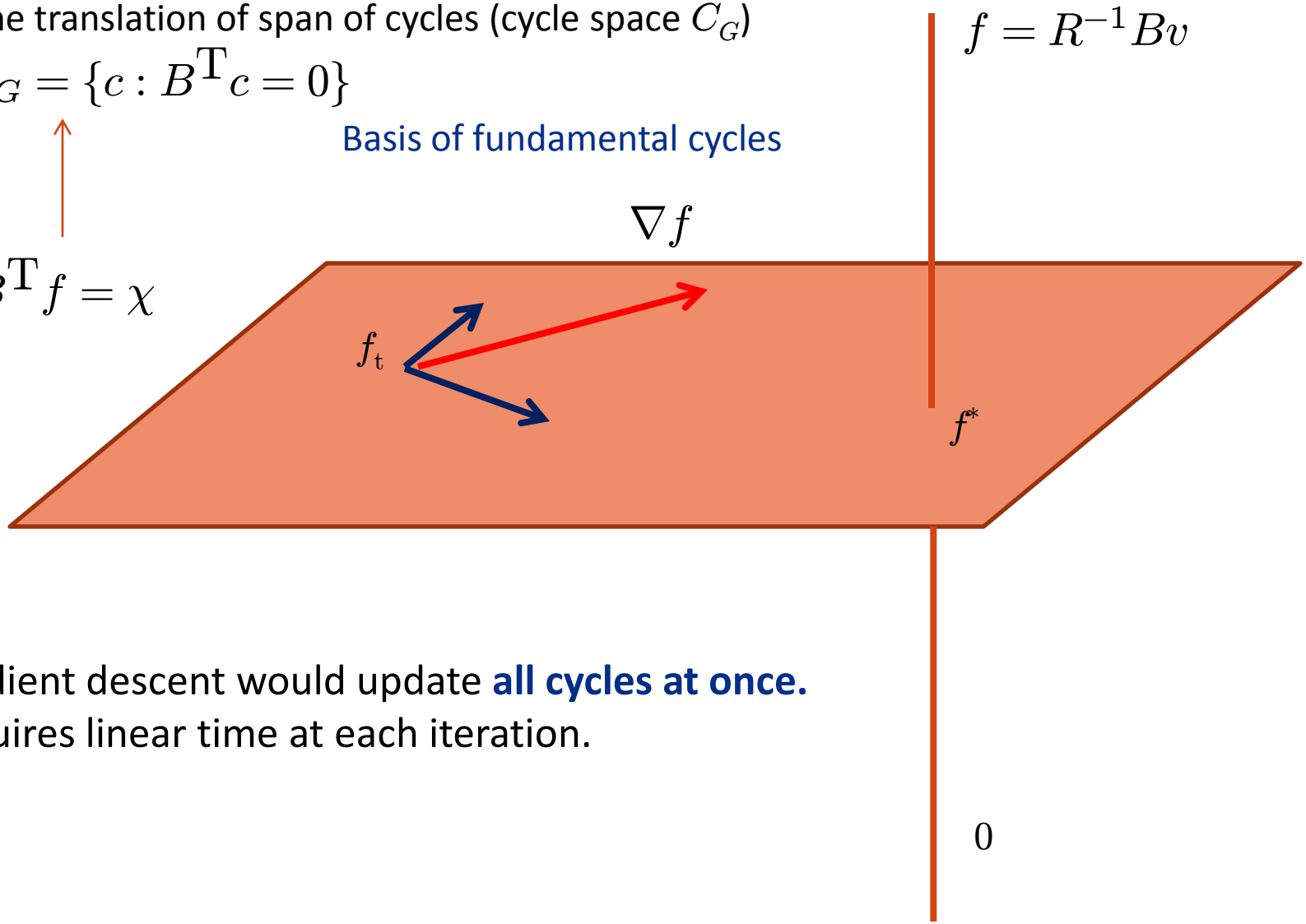
Choosing the Right Iteration Gradient Descent?

An affine translation of span of cycles (cycle space C_G)

$$C_G = \{c : B^{\mathrm{T}} c = 0\}$$

Basis of fundamental cycles

$$B^{\mathrm{T}} f = \chi$$



Gradient descent would update **all cycles at once.**
Requires linear time at each iteration.

Choosing the Right Iteration
Randomized Cordinate Descent!

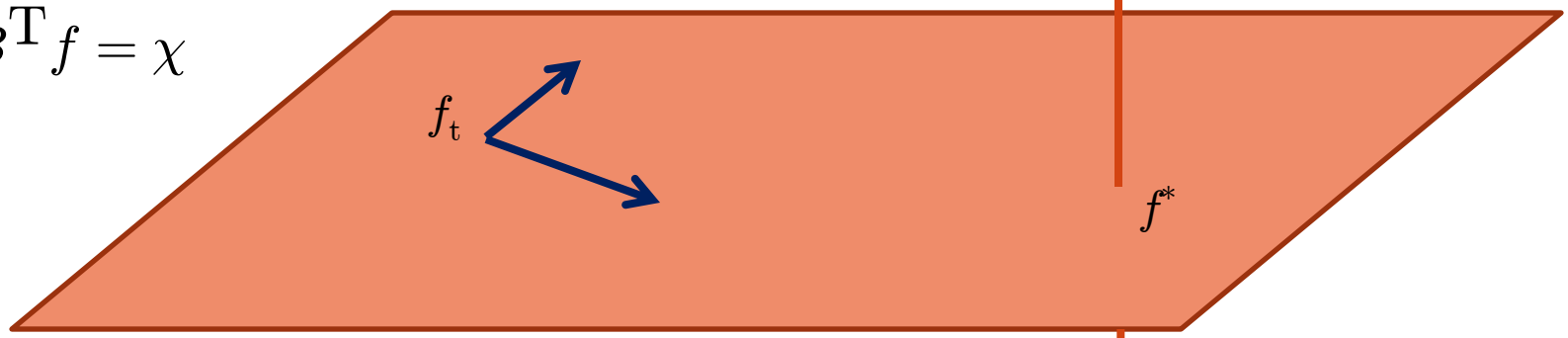
An affine translation of span of cycles (cycle space C_G)

$$C_G = \{c : B^{\mathrm{T}} c = 0\}$$

Basis of fundamental cycles

$$B^{\mathrm{T}} f = \chi$$

$$f = R^{-1} B v$$



Move along one coordinate direction at a time.
Choose basis vector c_e w.p. proportional to

$$\frac{c_e^T R c_e}{r_e} = \mathrm{st}(e) + 1$$

[SV09]

Convergence analysis?

Coordinate Descent: Convergence

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

Iteration count for gradient descent:

$$T = O\left(\frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

Coordinate Descent: Convergence

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

Iteration count for gradient descent:

$$T = O\left(\frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

Worse by a factor of $m - n + 1$ in our case

Convergence

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

$$\lambda_{\min}(A) = 1 \qquad \lambda_{\max}(A) \cdot \mathrm{Tr}(A) = \mathrm{st}(T) + O(n)$$

Tree stretch

Same calculation as for voltage preconditioning.

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

New Condition Numbers

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

$$\lambda_{\min}(A) = 1 \qquad \lambda_{\max}(A) \cdot \mathrm{Tr}(A) = \mathrm{st}(T) + O(n)$$

Tree stretch

Same calculation as for voltage preconditioning.

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right)$$

Worse by a factor of $m - n + 1$ in our case

We start with a trace guarantee,
Not an eigenvalue one

Final Convergence

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

$$\lambda_{\min}(A) = 1 \qquad \lambda_{\max}(A) \cdot \mathrm{Tr}(A) = \mathrm{st}(T) + O(n)$$

Tree stretch

Same calculation as for voltage preconditioning.

Pick spanning tree T to be low stretch.

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right) = \tilde{O}\left(m \log n \log\left(\frac{n}{\epsilon}\right)\right)$$

Final Convergence

$$\min_{x \perp \vec{1}} \quad \frac{1}{2} \cdot x^{\mathrm{T}} A x - b^{\mathrm{T}} x$$

$$\forall e \in E \setminus T: \quad c_e^T R(f_0 + f) = 0$$

$$f = \sum_{e \in E \setminus T} f_e c_e$$

$$\lambda_{\min}(A) = 1 \qquad \lambda_{\max}(A) \cdot \mathrm{Tr}(A) = \mathrm{st}(T) + O(n)$$

Tree stretch

Same calculation as for voltage preconditioning.

Pick spanning tree T to be low stretch.

Iteration count for randomized coordinate descent:

$$T = O\left(\frac{\mathrm{Tr}(A)}{\lambda_{\min}(A)} \log\left(\frac{n}{\epsilon}\right)\right) = \tilde{O}\left(m \log n \log\left(\frac{n}{\epsilon}\right)\right)$$

NEARLY-LINEAR

Algorithmic Approach

Initialize: Get a crude feasible $\vec{f}$

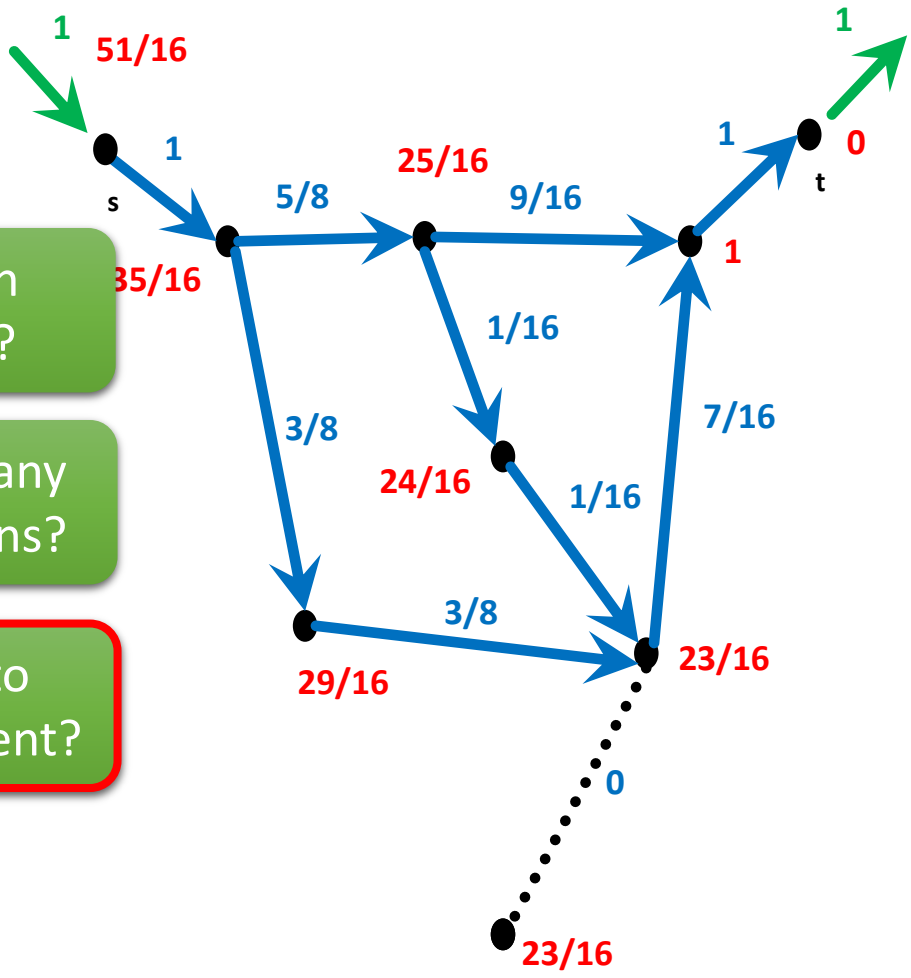
Improve:
- Pick a cycle
- Fix it
- Repeat

Which cycle?

How many iterations?

How to implement?

Output: Round to voltages v



IDEA: A nearly-linear number of cheap (i.e. $\mathrm{O}(\log n)$ iterations!

Implement updates efficiently

- Need to compute the potential drop along a tree cycle C_e

$$\sum_{e' \in C_e} f_e r_e$$

- Need to add a constant α to a tree cycle C_e

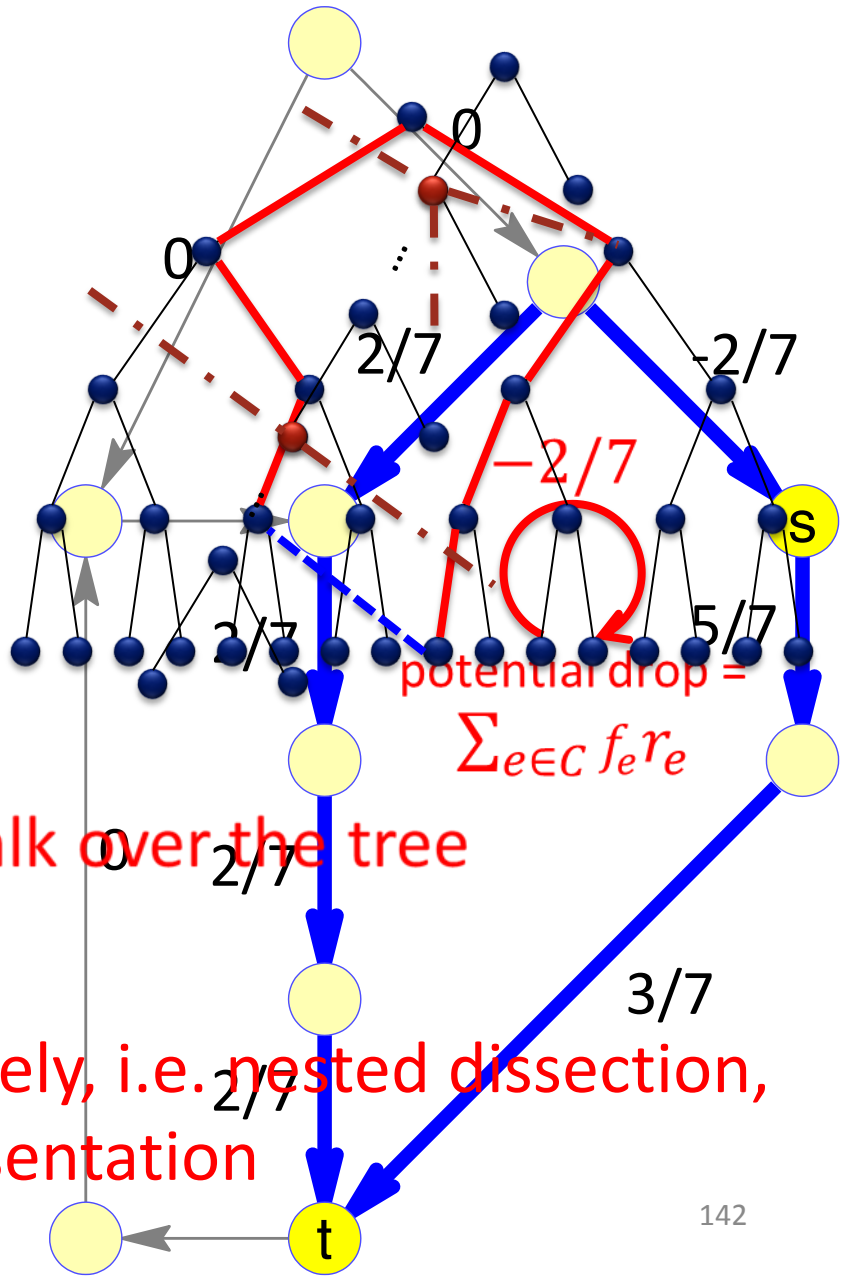
$$\forall e' \in C_e, \qquad f_e \leftarrow f_e + \alpha;$$

- When the tree is balanced

  requires $O(\log n)$ time to walk over the tree

- Otherwise

  decompose the tree recursively, i.e. nested dissection, gives $\mathrm{O}(\log\ n)$ sparse representation



Resulting Algorithm

Which cycle?

Fundamental cycle from low-stretch spanning tree T, sampled with probability proprtional to stretch.

- Pick a cycle
- Fix it
- Repeat

How many iterations?

How to implement?

Resulting Algorithm

Which cycle?

Fundamental cycle from low-stretch spanning tree T, sampled with probability proprtional to stretch.

- Pick a cycle
- Fix it
- Repeat

How many iterations?

$$\tilde{O}\left(m \log n \log\left(\frac{n}{\epsilon}\right)\right)$$

Randomized coordinate descent analysis

How to implement?

Resulting Algorithm

- Pick a cycle
- Fix it
- Repeat

Which cycle?

Fundamental cycle from low-stretch spanning tree T, sampled with probability proprtional to stretch.

How many iterations?

$$\tilde{O}\left(m \log n \log\left(\frac{n}{\epsilon}\right)\right)$$

Randomized coordinate descent analysis

How to implement?

$$O(\log n)$$

Exploiting separators of size 1 in tree

Resulting Algorithm

Which cycle?

Fundamental cycle from low-stretch spanning tree T, sampled with probability proprtional to stretch.

- Pick a cycle
- Fix it
- Repeat

How many iterations?

$$\tilde{O}\left(m \log n \log\left(\frac{n}{\epsilon}\right)\right)$$

Randomized coordinate descent analysis

How to implement?

$$O(\log n)$$

Exploiting separators of size 1 in tree

TOTAL RUNNING TIME: $\tilde{O}\left(m \log^2 n \log\left(\frac{n}{\epsilon}\right)\right)$

Running Time Improvements

$$\tilde{O}\left(m \log^2 n \log\left(\tfrac{n}{\epsilon}\right)\right)$$

Multiple phases with restarts

$$\tilde{O}\left(m \log^2 n \log\left(\tfrac{1}{\epsilon}\right)\right)$$

Lee, Sidford at FOCS'13

Accelerated Coordinate Descent

$$\tilde{O}\left(m \log^{1.5} n \log\left(\tfrac{1}{\epsilon}\right)\right)$$

Running Time Improvements

$$\tilde{O}\left(m \log^2 n \log\left(\tfrac{n}{\epsilon}\right)\right)$$

Multiple phases with restarts

$$\tilde{O}\left(m \log^2 n \log\left(\tfrac{1}{\epsilon}\right)\right)$$

Lee, Sidford at FOCS'13

Accelerated Coordinate Descent

$$\tilde{O}\left(m \log^{1.5} n \log\left(\tfrac{1}{\epsilon}\right)\right)$$

Open Questions

RELATED TO THIS ALGORITHM:

- Can this algorithm be parallelized? **Practically important**.

- Can it be made deterministic? Random order works on any RHS with high probability.

MORE GENERAL:

- Other application of idea about **many cheap iterations**?
  E.g. cycle update view of undirected maximum flow result?

- Other applications of underlying idea: **right representation + right iterative method**
  Almost-linear-time undirected maximum flow falls in this framework
  Can we tackle **other important graph problems**?
  e.g. directed maximum flow, better multicommodity flows, …