Almost-Linear-Time Algorithms for Fundamental Graph Problems

A New Framework and Its Applications

Lorenzo Orecchia

MIT MATHEMATICS

A Tale of Two Disciplines

NEW INSIGHT: Deep connections between core concepts

These two fields have expanded and diverged

Use techniques from one to help the other

They face similar challenges, but with different tools

Fastest Algorithms for
Fundamental Graph Problems

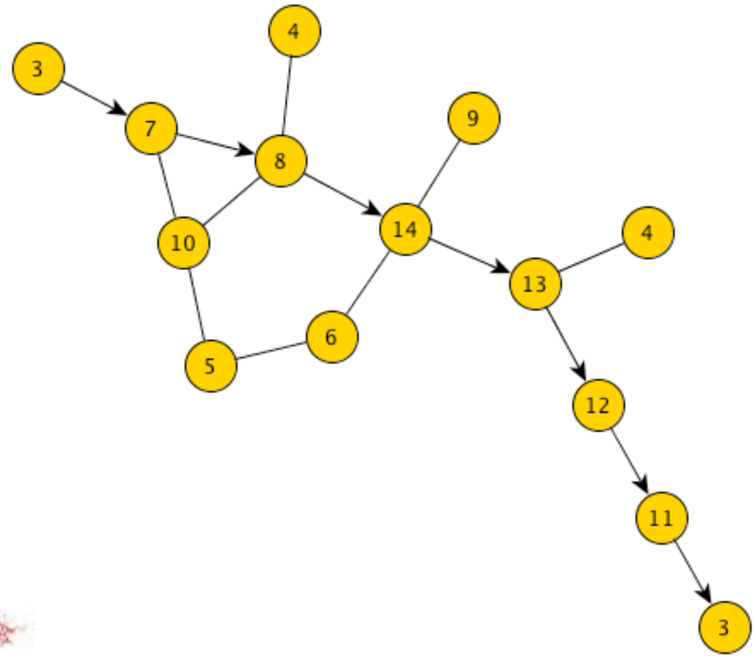
- Asks about paths, flows, cuts, clustering, routing

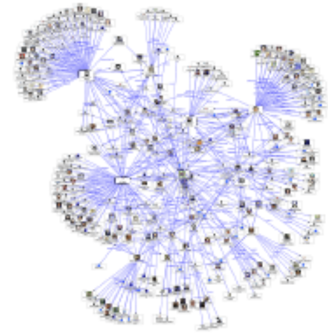
- Asks about matrices, PDEs, computational linear algebra

A New Framework for the
Design of Fast Algorithms

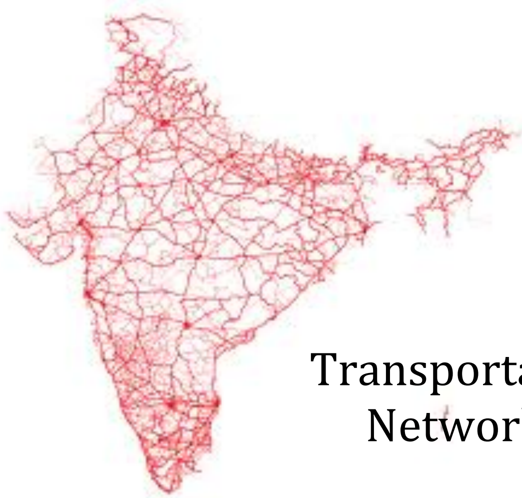
Why Graph Algorithms?

Computer
Networks



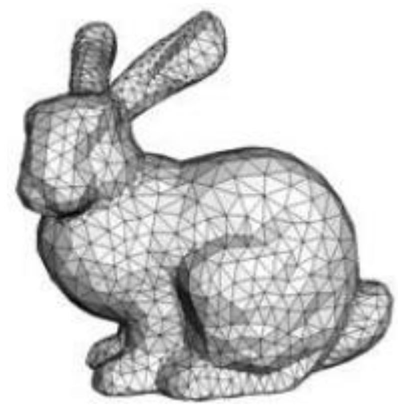


Social
Networks



Transportation
Networks

Representations of
Physical Objects



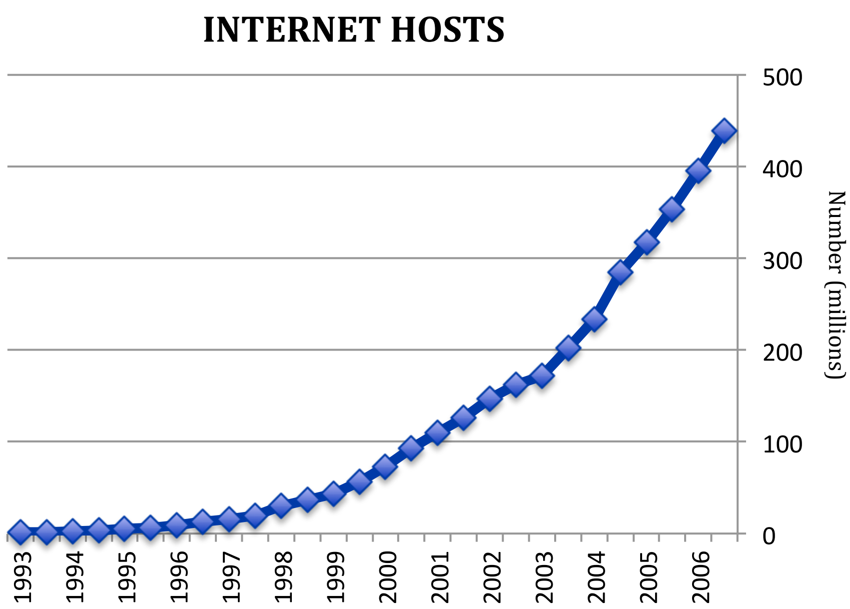
Why Fast Graph Algorithms

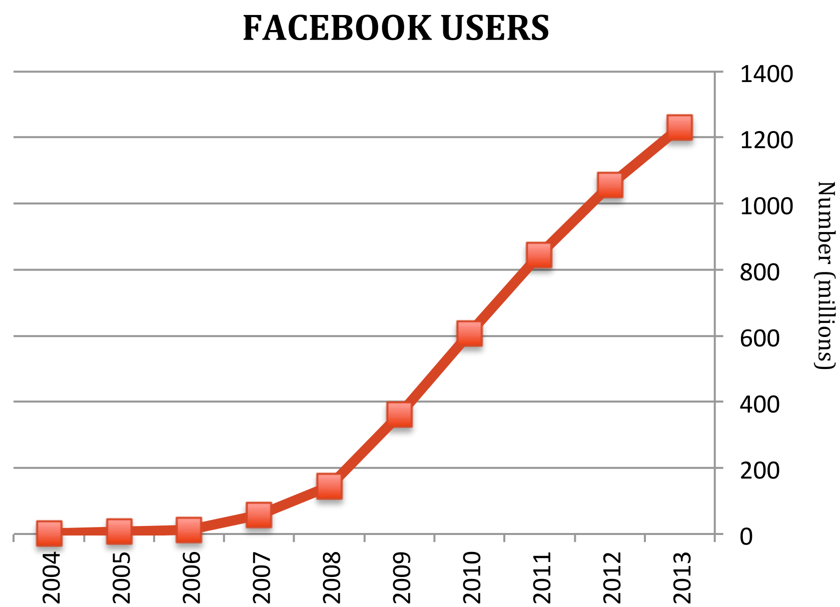
- Classical Algorithms (1970s-1990s):

  Standard notion of efficiency is polynomial running time

- Today:

  Graphs of interest are getting larger and larger …





Even quadratic running time is **unfeasibly large** for these instances

Why Fast Graph Algorithms

- Classical Algorithms (1970s-1990s):

  Standard notion of efficiency is polynomial running time

- Today:

  Graphs of interest are getting larger and larger …

  Even quadratic running time is unfeasibly large for these instances

- New Efficiency Requirement: **as close as possible to linear**

ALMOST-LINEAR RUNNING TIME

Input Size	Running Time
n	$O(n) \cdot$ [Lower-order terms] $O(n^{1+\epsilon})$ for any $\epsilon > 0$

Contrast with **Super-Linear Time** $\Omega(n^{1+\delta})$ for some $\delta > 0$

$$n^2, n^{1.5}, n^{1.001}$$

Almost-Linear-Time Algorithms

GOAL: Build library of primitives running in almost-linear time

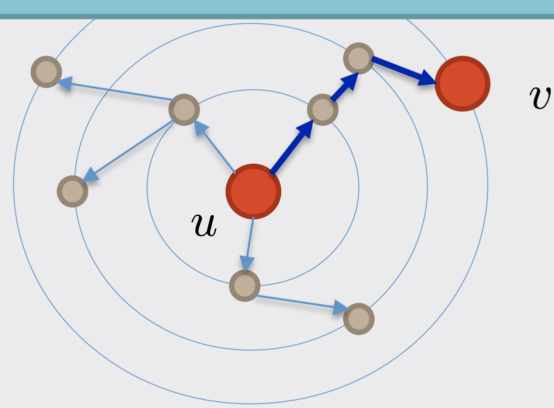
- What can you do to a graph in almost-linear time?

Almost-Linear Time

Reachability
Shortest Path
Connectivity
Minimum Cost Spanning Tree

SHORTEST PATH PROBLEM:
What is

Probe Graph Structure in Simple Way



Almost-Linear-Time Algorithms

GOAL: Build library of primitives running in almost-linear time

- What can you do to a graph in almost-linear time?

Simple Probe of Graph Structure

Almost-Linear Time

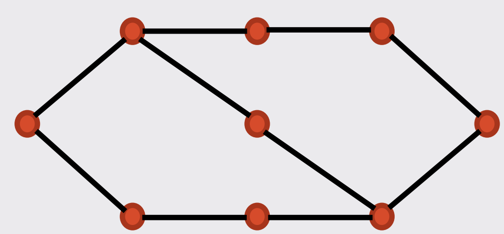
Reachability
Shortest Path
Connectivity
Minimum Cost Spanning Tree
...

Laplacian Systems of Linear Equations [Spielman, Teng'04]

Deep Probe of Graph Structure

Solve systems of linear equations with an implicit graph structure

$$Lx = b \quad \Longleftrightarrow$$



Fundamental problem in numerical analysis with ubiquitous applications

Almost-Linear-Time Algorithms: My Contributions

GOAL: Build library of primitives running in almost-linear time

Laplacian Systems of Linear Equations [Spielman, Teng'04]

BREAKTHROUGH: First almost-linear-time algorithm for complex graph problem

IDEA: Combine Computational Linear Algebra and Combinatorial Optimization

DISADVANTAGES: Involved theoretical algorithm, 3 papers = 100+ pages

New Solver for Laplacian Systems [Kelner, **Orecchia**, Sidford, Zhu '12]

BREAKTHROUGH: Faster, simple almost-linear-time algorithm

IDEA: Combine **Continuous Optimization** and Combinatorial Optimization

ADVANTAGES: 5 lines of pseudo-code, proof fits on 1 blackboard

Almost-Linear-Time Algorithms: My Contributions

GOAL: Build library of primitives running in almost-linear time

New Solver for Laplacian Systems [Kelner, **Orecchia**, Sidford, Zhu '12]

Faster, simple almost-linear-time algorithm

A New Framework for Designing Fast Algorithms

Combining Continuous Optimization and Combinatorial Optimization

Applying the Framework to Undirected Flow Problems

s-t Maximum Flow [Kelner, **Orecchia**, Sidford, Lee '13][Sherman'13]

First **almost-linear-time** algorithm for this foundational problem

Previous best running time: $O(n^{4/3} \text{ polylog}(n))$ [Christiano et al. '11]

Almost-Linear-Time Algorithms: My Contributions

GOAL: Build library of primitives running in almost-linear time

New Solver for Laplacian Systems [Kelner, **Orecchia**, Sidford, Zhu '12]

Faster, simple almost-linear-time algorithm

A New Framework for Designing Fast Algorithms

Combining Continuous Optimization and Combinatorial Optimization

Applying the Framework to Undirected Flow Problems

- s-t Maximum Flow [Kelner, **Orecchia**, Sidford, Lee '13][Sherman'13]
- Concurrent Multi-commodity Flow [Kelner, **Orecchia**, Sidford, Lee '13]
- Oblivious Routing [Kelner, **Orecchia**, Sidford, Lee '13]

… and Undirected Cut Problems

- Minimum s-t cut [Kelner, **Orecchia**, Sidford, Lee '13][Sherman'13]
- Approximate Sparsest Cut [Kelner, **Orecchia**, Sidford, Lee '13] [Sherman'13]
- Approximate Minimum Conductance Cut [**Orecchia**, Sachdeva, Vishnoi '12]

Talk Outline

New Solver for Laplacian Systems of Linear Equations

Faster, simple almost-linear-time algorithm

A New Framework for Designing Fast Algorithms

Combining Continuous Optimization and Combinatorial Optimization

Applying the Framework: Undirected s-t Maximum Flow

First almost-linear-time algorithm for a foundational graph problem

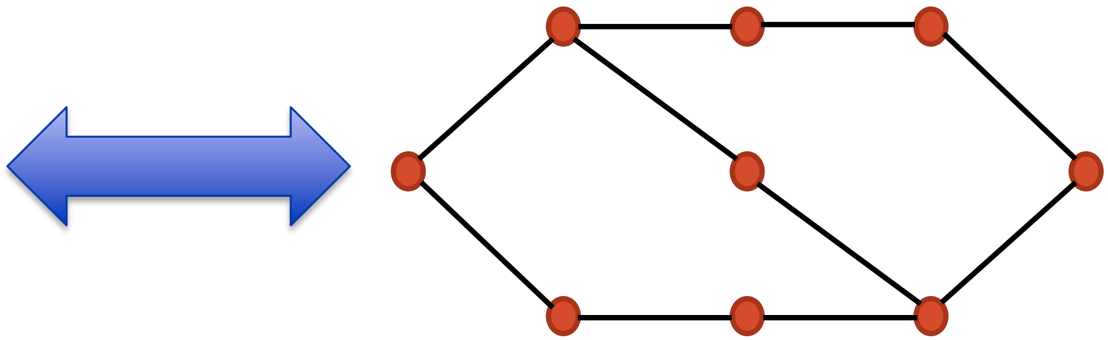
Future Directions

Solving Laplacian Systems In Almost-Linear Time:

A Simple Algorithm

Laplacian Systems of Linear Equations

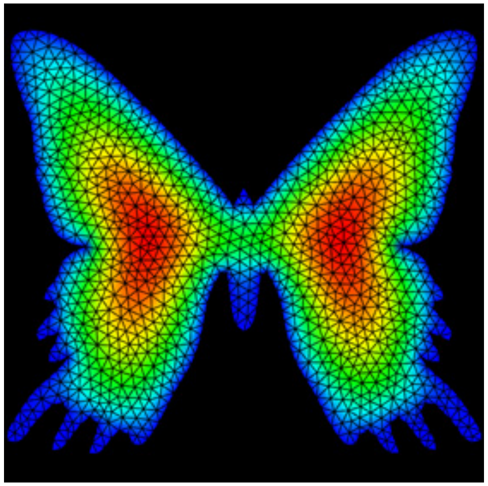
$$Ax = b$$



Fundamental Problem in Numerical Analysis

<u>Some Applications</u>

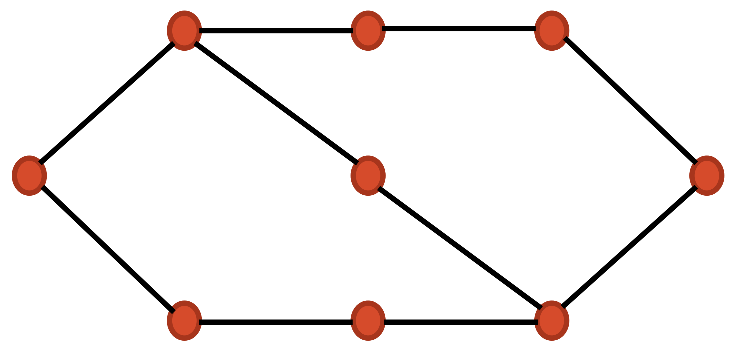
- **Finite-element method**
- Image Smoothing
- Network Analysis



Laplacian Systems of Linear Equations

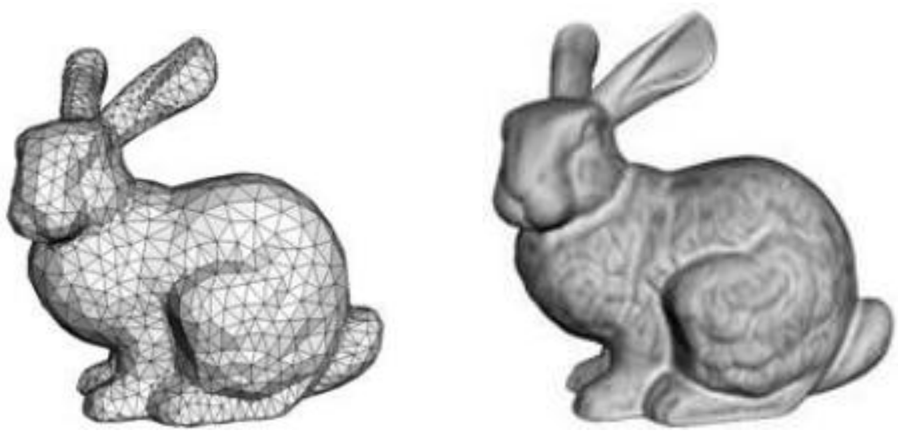
Fundamental Problem in Numerical Analysis and Simulation of Physical Systems

$$Ax = b$$



Some Applications

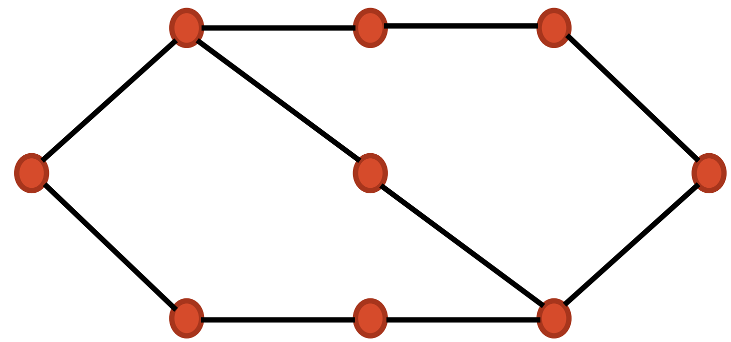
- Finite-element method
- **Image Smoothing**
- Network Analysis



Laplacian Systems of Linear Equations

Fundamental Problem in Numerical Analysis and Simulation of Physical Systems

$$Ax = b$$

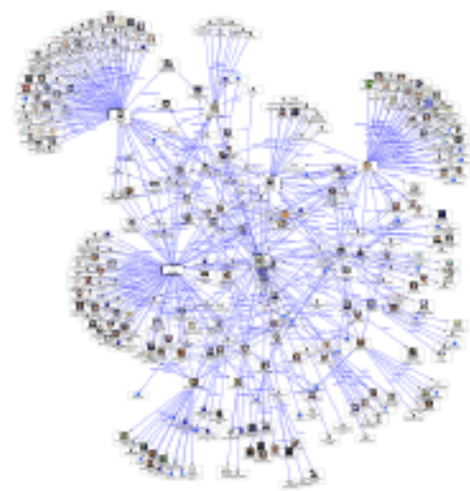


<u>Some Applications</u>

- Finite-element method

- Image Smoothing

- **Network Analysis**

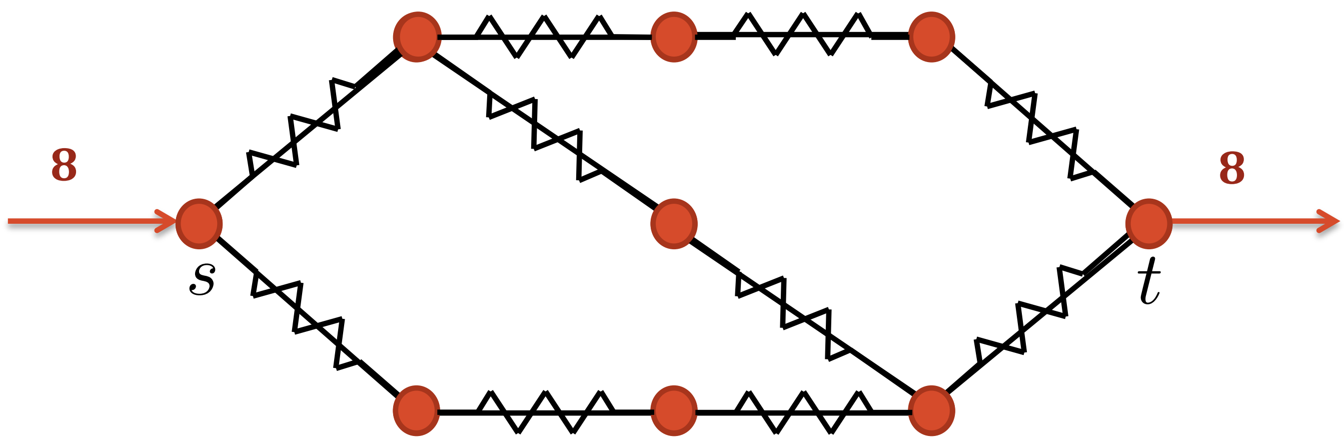


Laplacian Systems and Electrical Flow

$$Ax = b$$

Matrix A defines a graph

Vector b defines:
flow input/output



Graph → Electrical Circuit

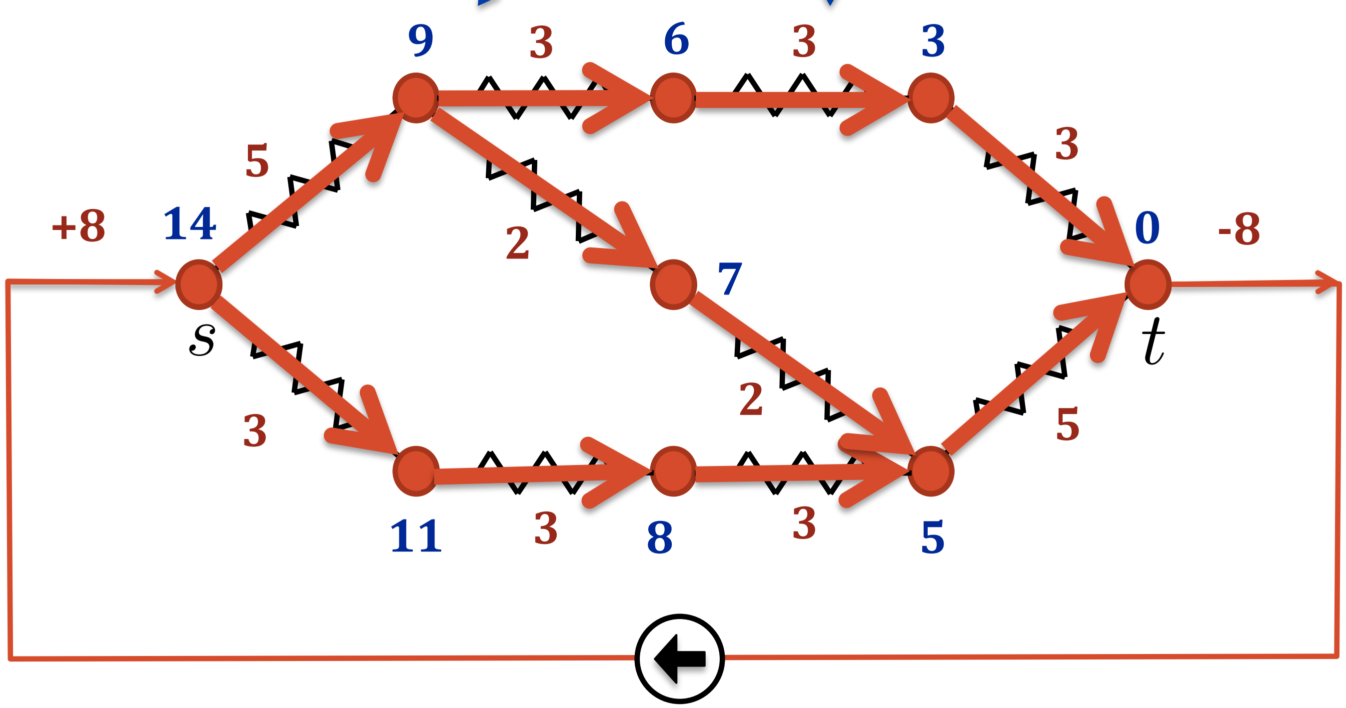
Edges → Unit resistors

Laplacian Systems and Electrical Flow

$$Ax = b$$

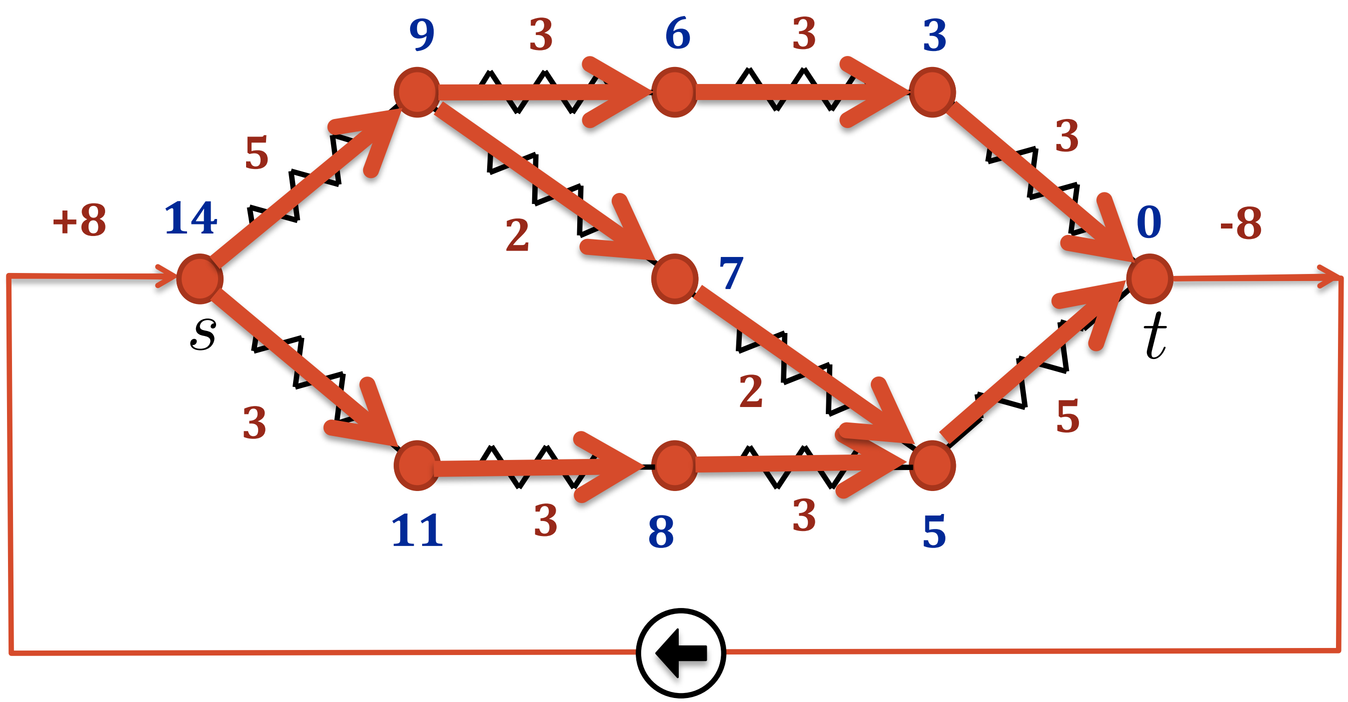
Solution x is **voltage** induced by current

Vector b defines **electrical current** input/output



Current Source

Laplacian Systems and Electrical Flow

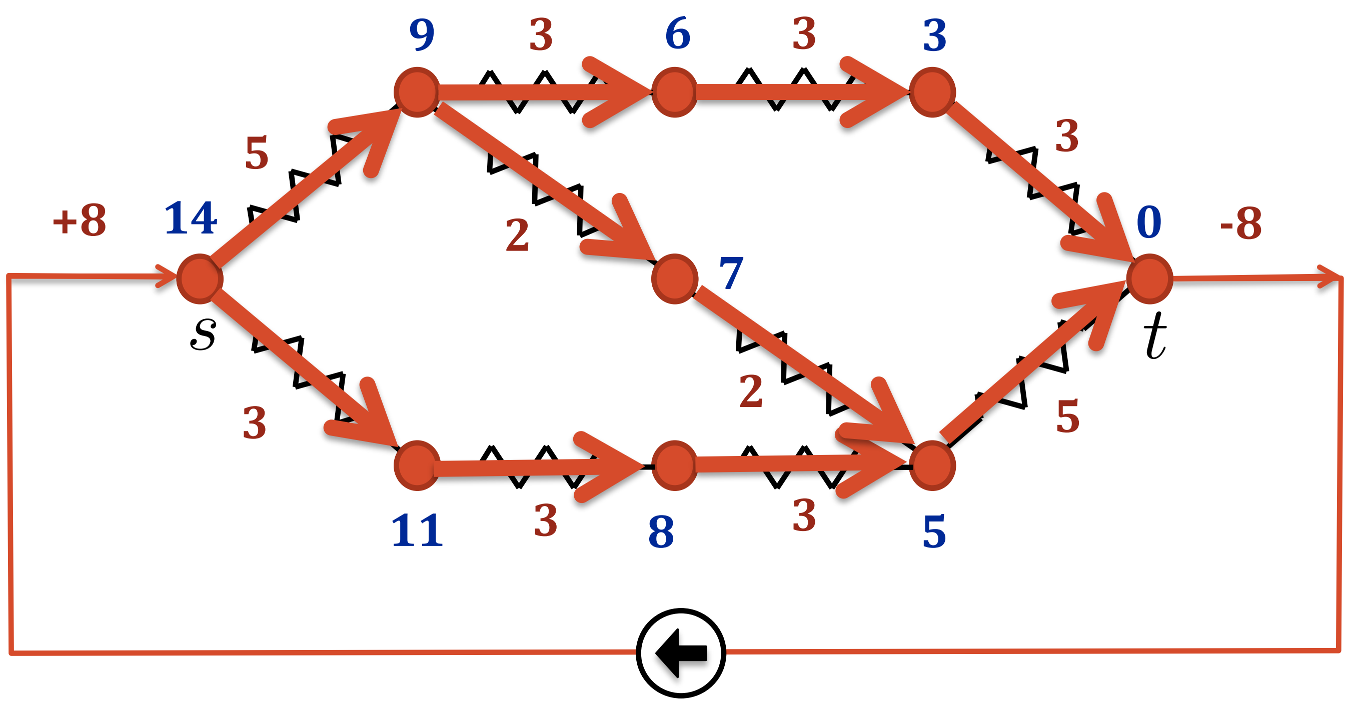


Computational Challenge: Compute how <u>electrical flow spreads</u> in the circuit in almost-linear-time in the number of edges m

Optimization Characterization: Electrical flow minimizes energy

$$\min_{f \text{ routes } (s,t)} \sum_{e \in E} r_e f_e^2$$

Laplacian Systems and Electrical Flow



Equivalent Characterization (Ohm's Law):
There exist voltages v such that for every edge $e = (a,b)$,

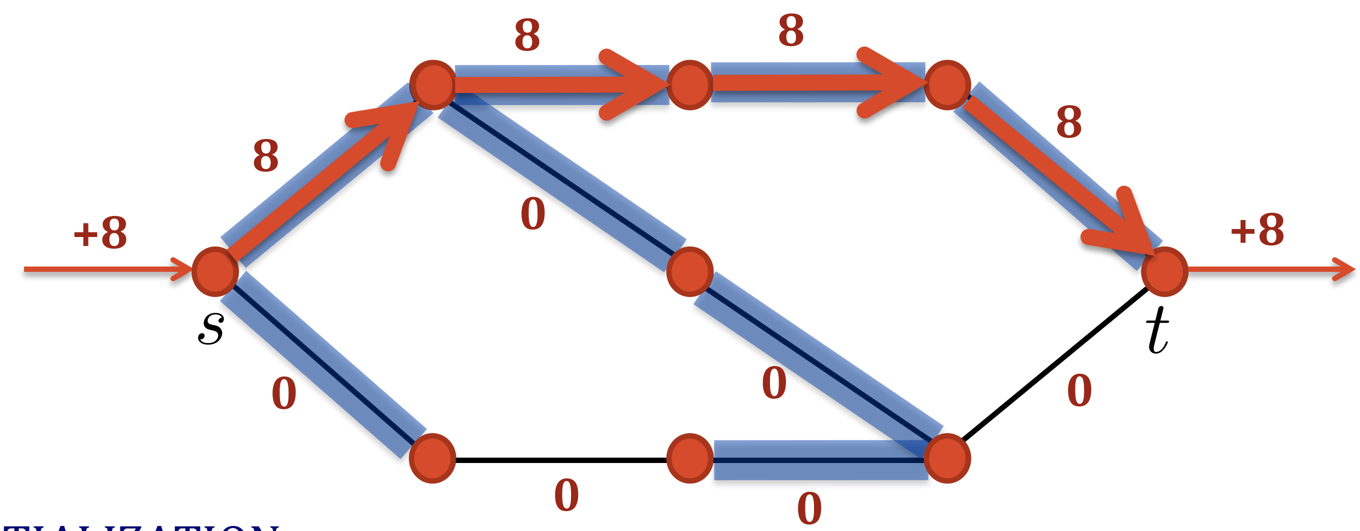
$$f_e = \frac{(v_b - v_a)}{r_e}$$

Electrical Flow ⟶ f_e; Voltage Gap ⟵ $(v_b - v_a)$; Edge Resistance ⟵ r_e

Previous Work

- Vast amounts of work on solving various subclasses of graphs
  - Multigrid on grids and meshes
- General direct solvers
  - Gaussian elimination, Strassen's algorithm
- General iterative solvers
  - Conjugate gradient, Chebyshev's method
- For Laplacians, long line of work leading to almost-linear-time algorithm
  - **Very complicated:** Algorithm and analysis of Spielman and Teng is divided into 3 papers totaling >130 pages
- All previous almost-linear-time graph algorithms have same structure
  - Can be seen as combinatorial analogue of Multigrid

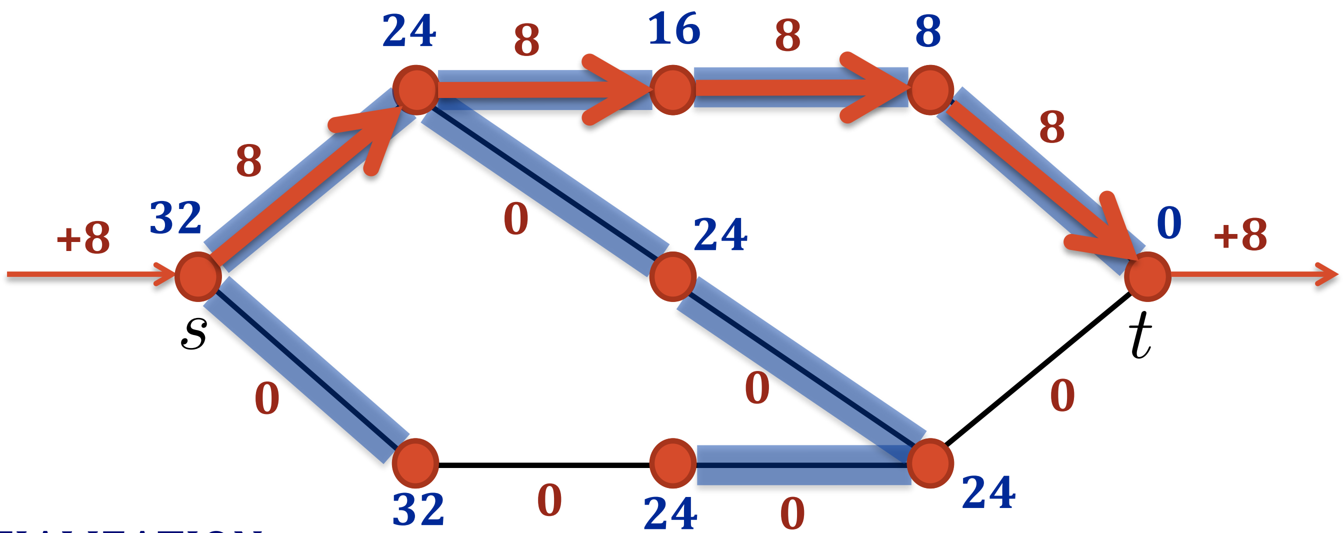
Our Fastest, Simplest Laplacian Solver



INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow f_0.

Our Fastest, Simplest Laplacian Solver



INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow f_o.

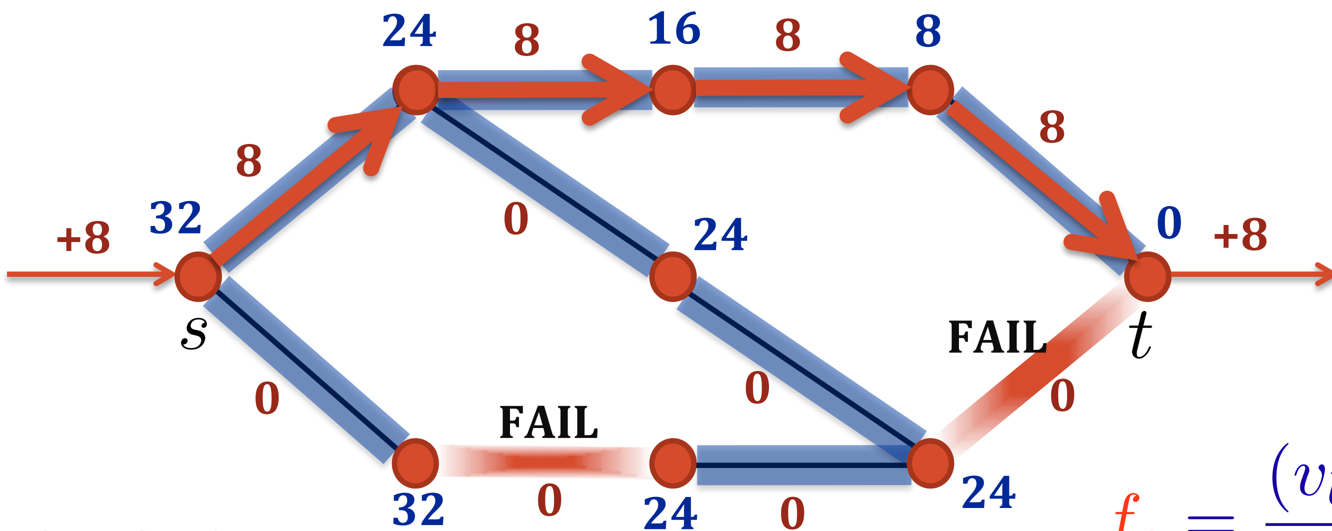
NOTE:

Flow f_o is electrical flow if and only if **Ohm's Law** holds for all edges e=(b,a)

$$f_e = \frac{(v_b - v_a)}{r_e}$$

- Apply **Ohm's Law** to the spanning-tree edges to deduce voltages.

Our Fastest, Simplest Laplacian Solver



$$f_e = \frac{(v_b - v_a)}{r_e}$$

INITIALIZATION:

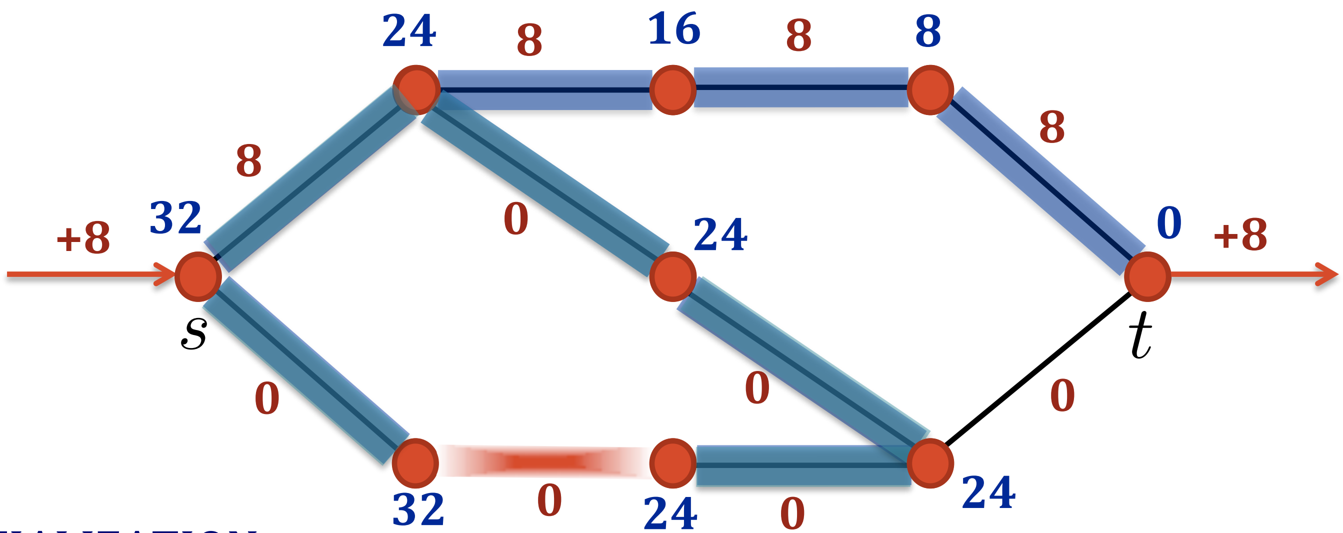
- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow f_o.

NOTE:

Flow f_o is electrical flow if and only if **Ohm's Law** holds for all edges $e=(b,a)$

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.

Our Fastest, Simplest Laplacian Solver



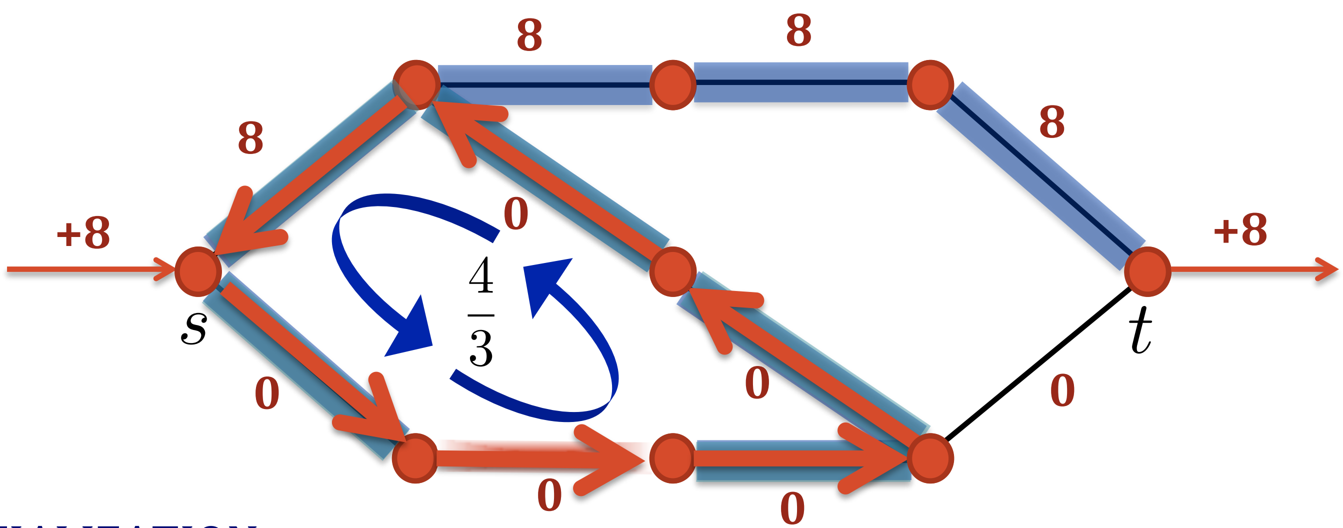
INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow $\mathbf{f_0}$.

MAIN LOOP (CYCLE FIXING):

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.
- Consider **cycle** corresponding to **failing off-tree edge** e

Our Fastest, Simplest Laplacian Solver



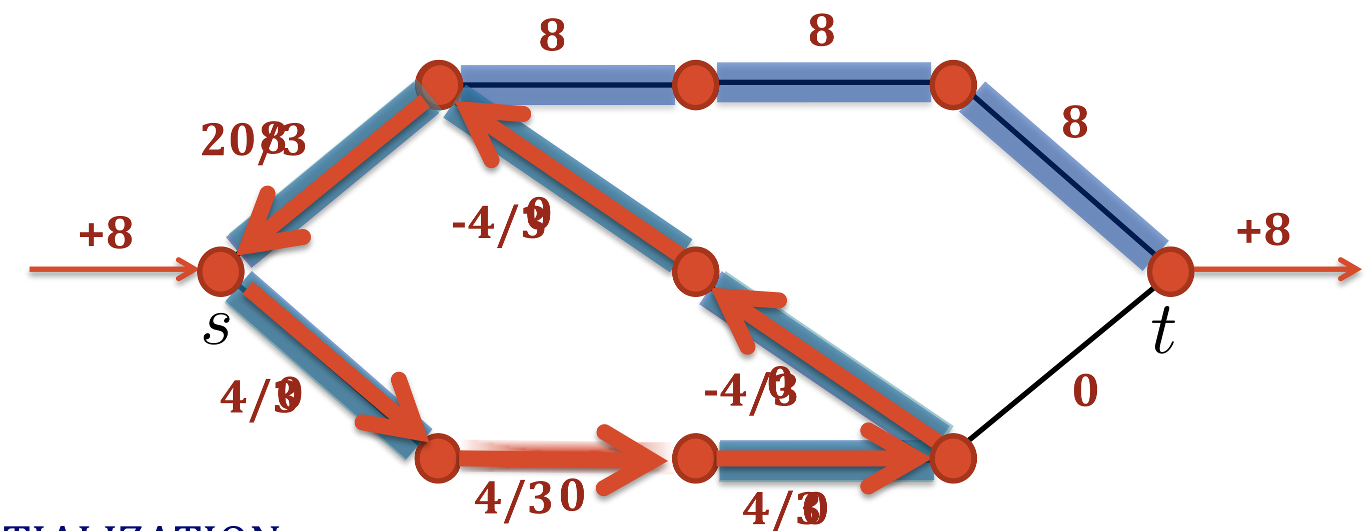
INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow $\mathbf{f_0}$.

MAIN LOOP (CYCLE FIXING):

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.
- Consider **cycle** corresponding to **failing off-tree edge** e
- Send **flow around cycle** until Ohm's Law is satisfied on e

Our Fastest, Simplest Laplacian Solver



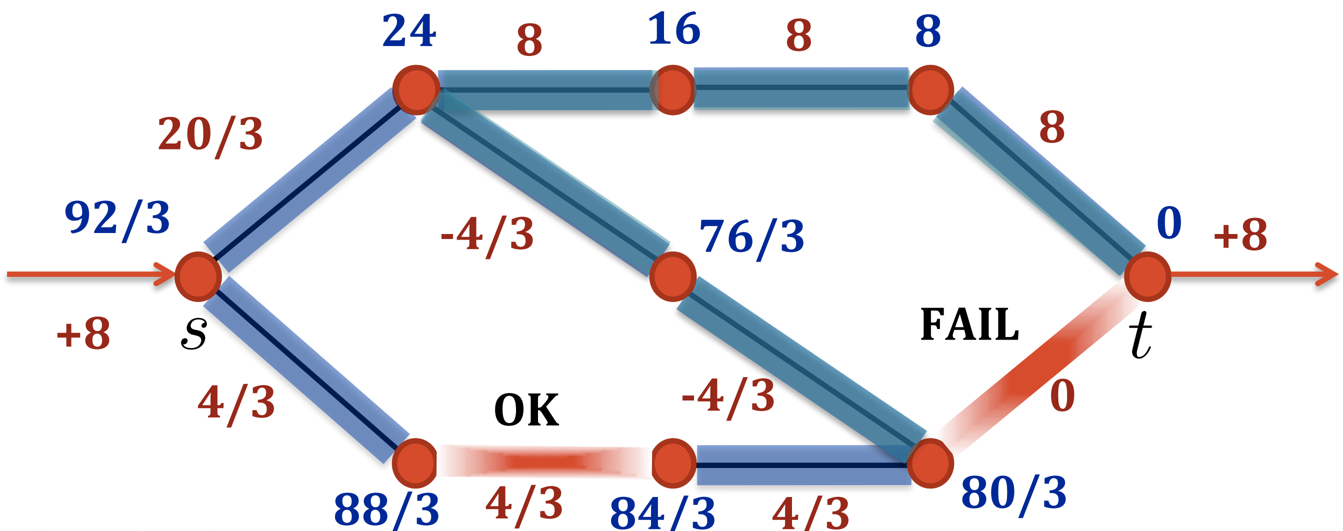
INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow $\mathbf{f_0}$.

MAIN LOOP (CYCLE FIXING):

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.
- Consider **cycle** corresponding to **failing off-tree edge** e
- Send **flow around cycle** until Ohm's Law is satisfied on e
- Repeat

Our Fastest, Simplest Laplacian Solver



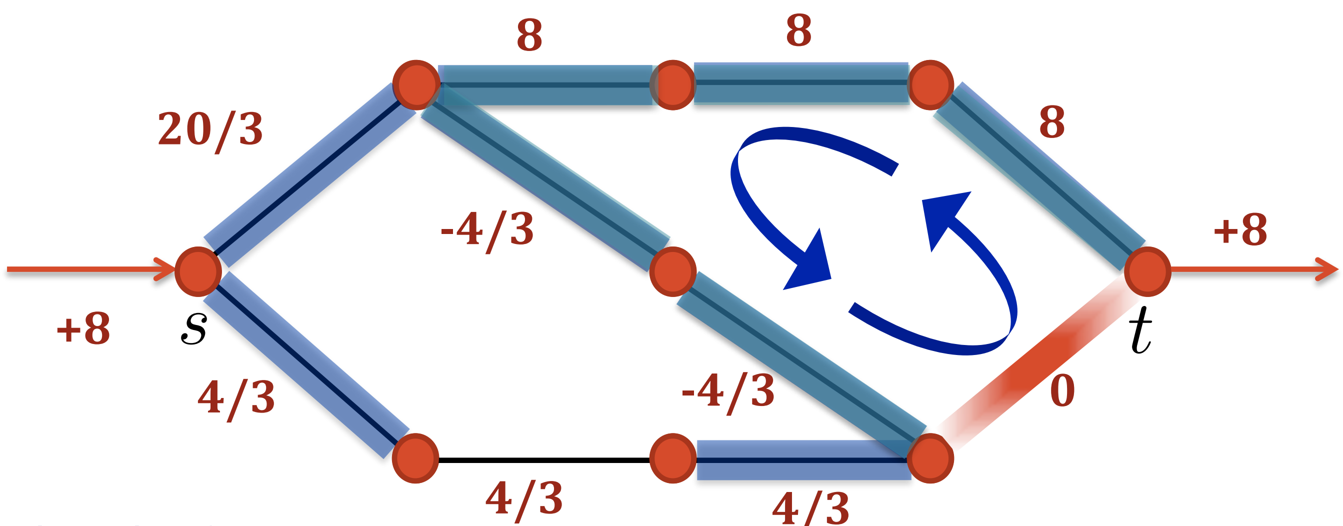
INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow $\mathbf{f_0}$.

MAIN LOOP (CYCLE FIXING):

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.
- Consider **cycle** corresponding to **failing off-tree edge** e
- Send **flow around cycle** until Ohm's Law is satisfied on e
- Repeat

Our Fastest, Simplest Laplacian Solver



INITIALIZATION:

- Choose a spanning tree **T** of G.
- Route flow along **T** to obtain initial flow $\mathbf{f_0}$.

MAIN LOOP (CYCLE FIXING):

- Apply **Ohm's Law** to the spanning-tree edges to **deduce voltages.**
- Check if voltages and flows obey Ohm's Law on **off-tree edges**.
- Consider **cycle** corresponding to **failing off-tree edge** e
- Send **flow around cycle** until Ohm's Law is satisfied on e
- Repeat

Algorithm Analysis

- Pick a cycle
- Fix it
- Repeat

Does it converge? → **YES**. Converges to electrical flow.

How quickly ? → Depends on:

1. Choice of **spanning tree**
2. Order of **cycle updates**

   Randomized Order

Algorithm Analysis

- Pick a cycle
- Fix it
- Repeat

Does it converge? $\longrightarrow$ **YES**. Converges to electrical flow.

How quickly ? $\longrightarrow$ Depends on:

1. Choice of **spanning tree**
2. Order of **cycle updates**

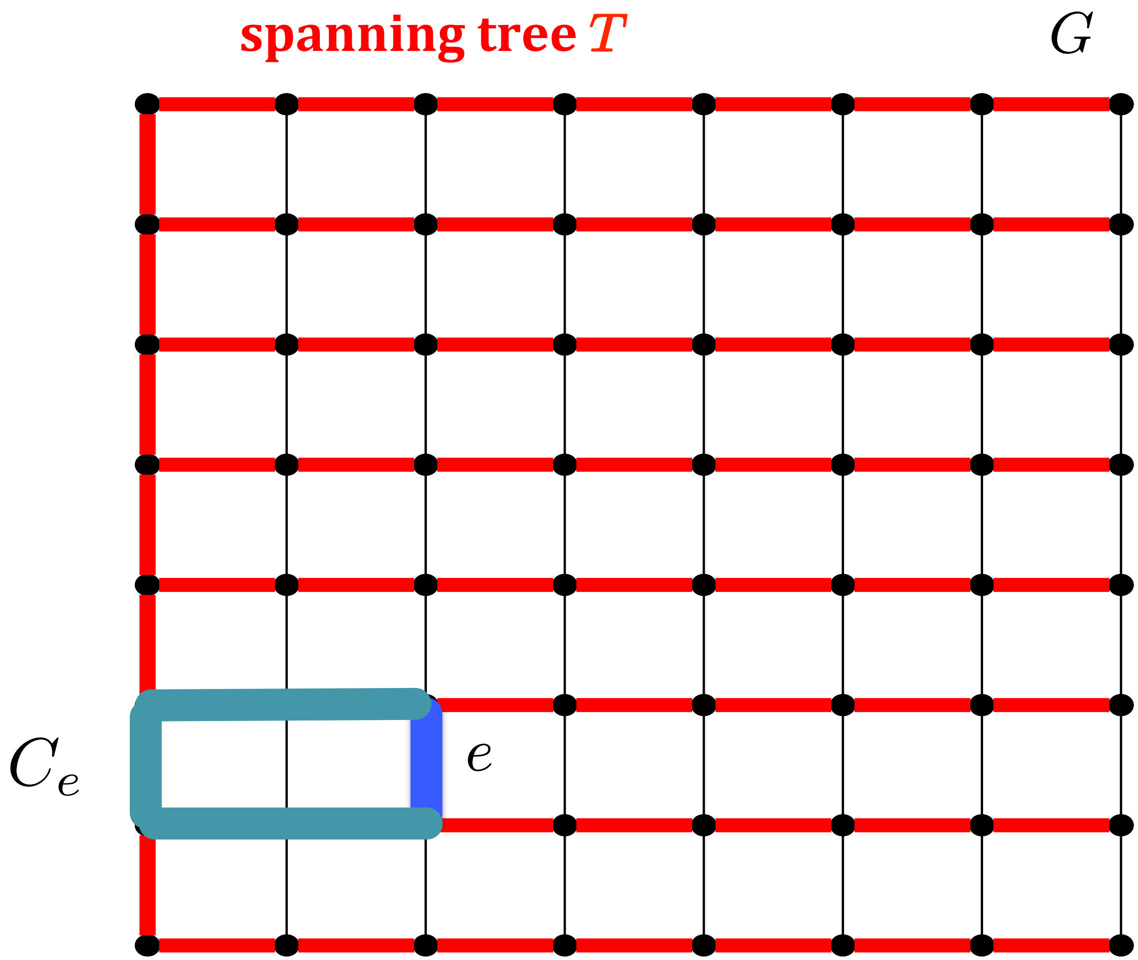
   Randomized Order

CHOICE OF SPANNING TREE:

- Cycle-fixing updates can interfere with one another, lead to slow convergence
- Choose spanning tree such that cycles interfere minimally:

  Spanning tree with minimal average cycle-length
- Number of iterations is $O(m) \cdot$ [average cycle-length]

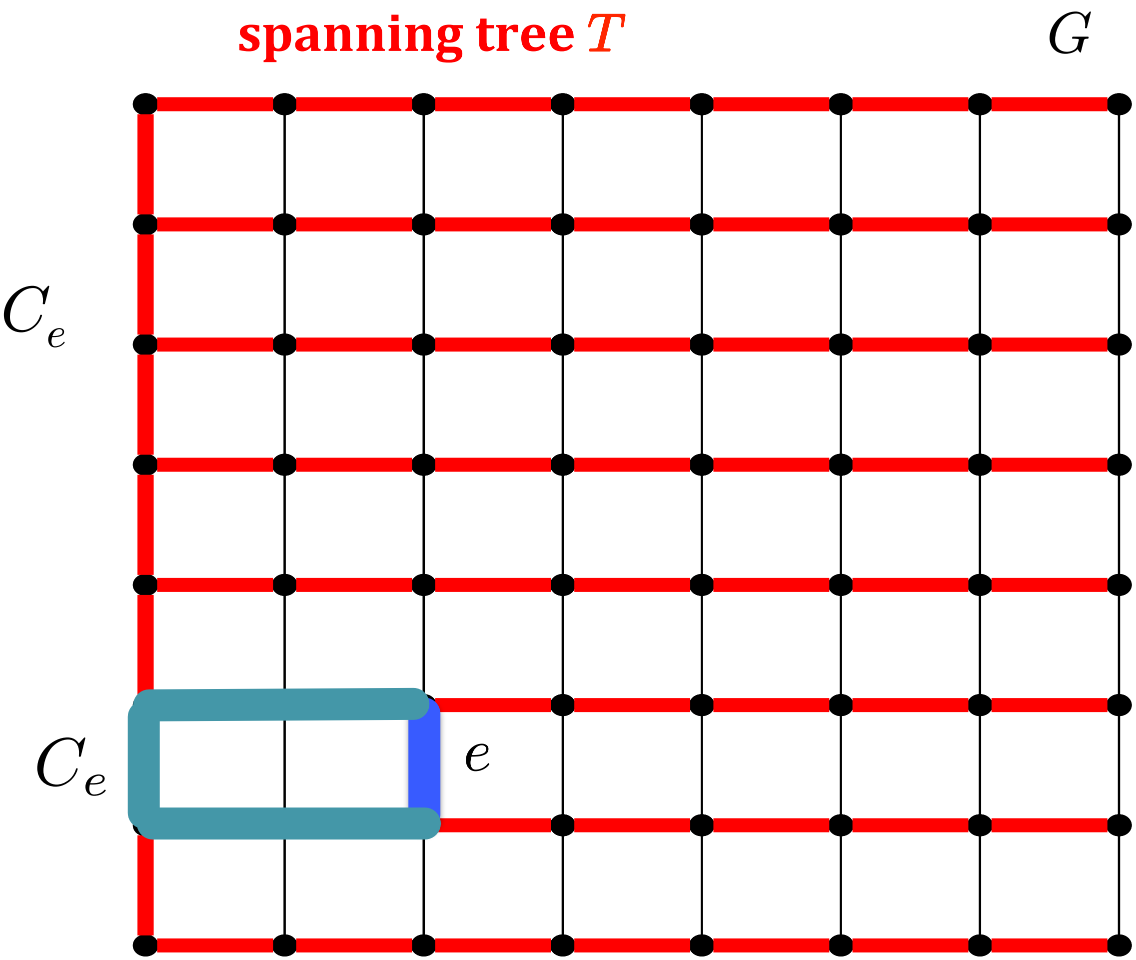
Low-Stretch Spanning Trees



Low-Stretch Spanning Trees

Stretch of e
=
Length of cycle C_e

$\mathrm{st}(e) = 5$

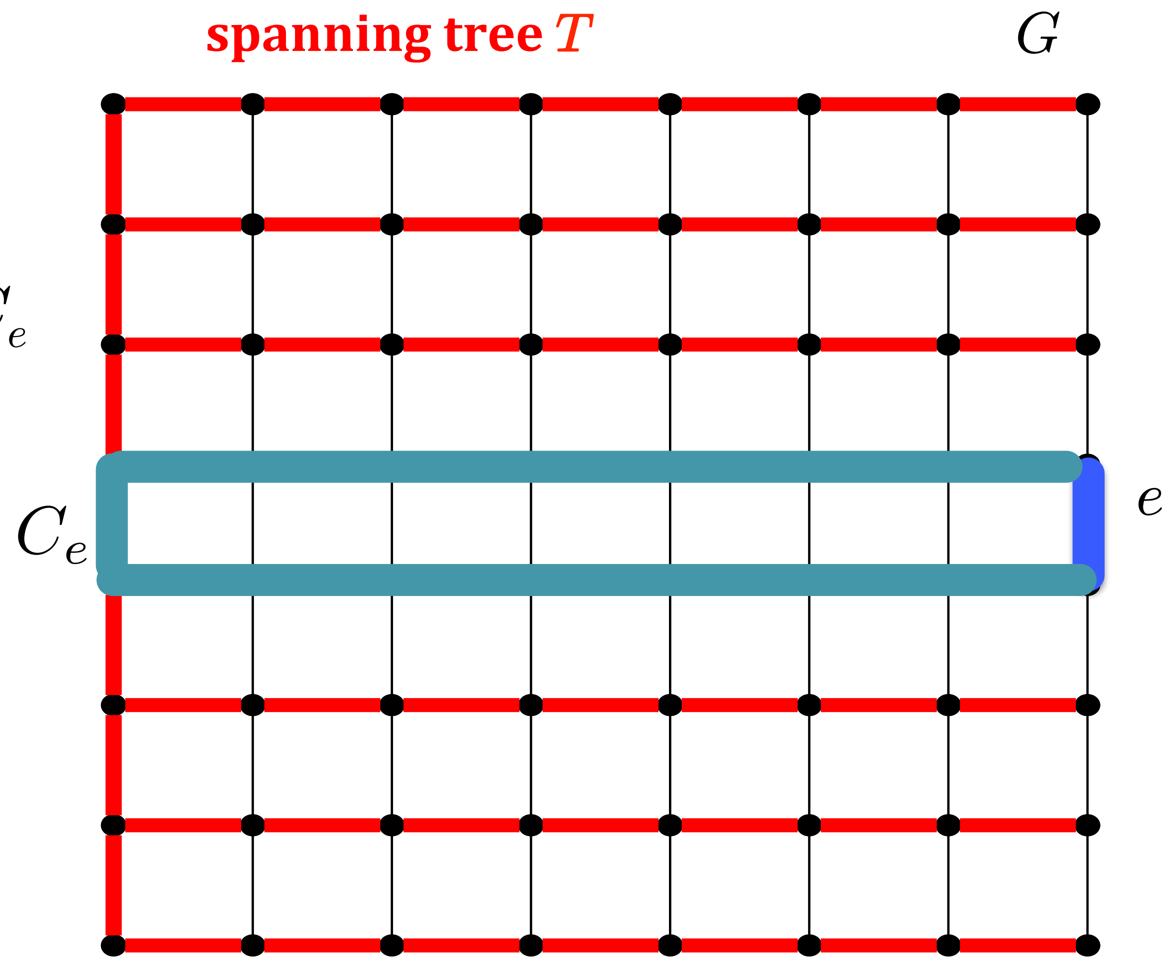


Low-Stretch Spanning Trees

Stretch of e

=

Length of cycle C_e



$\mathrm{st}(e) = 2\sqrt{n} + 1$

Low-Stretch Spanning Trees

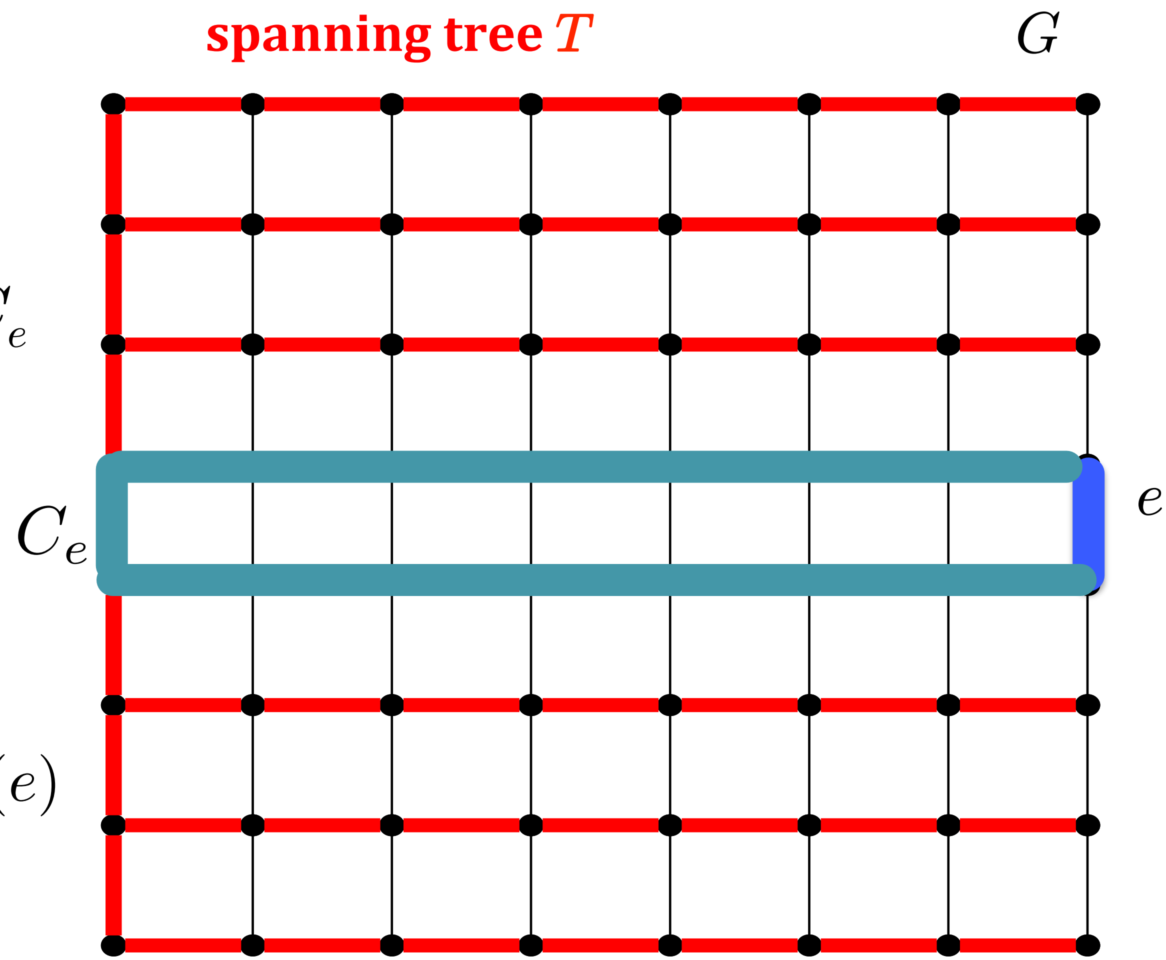
Stretch of e

=

Length of cycle C_e

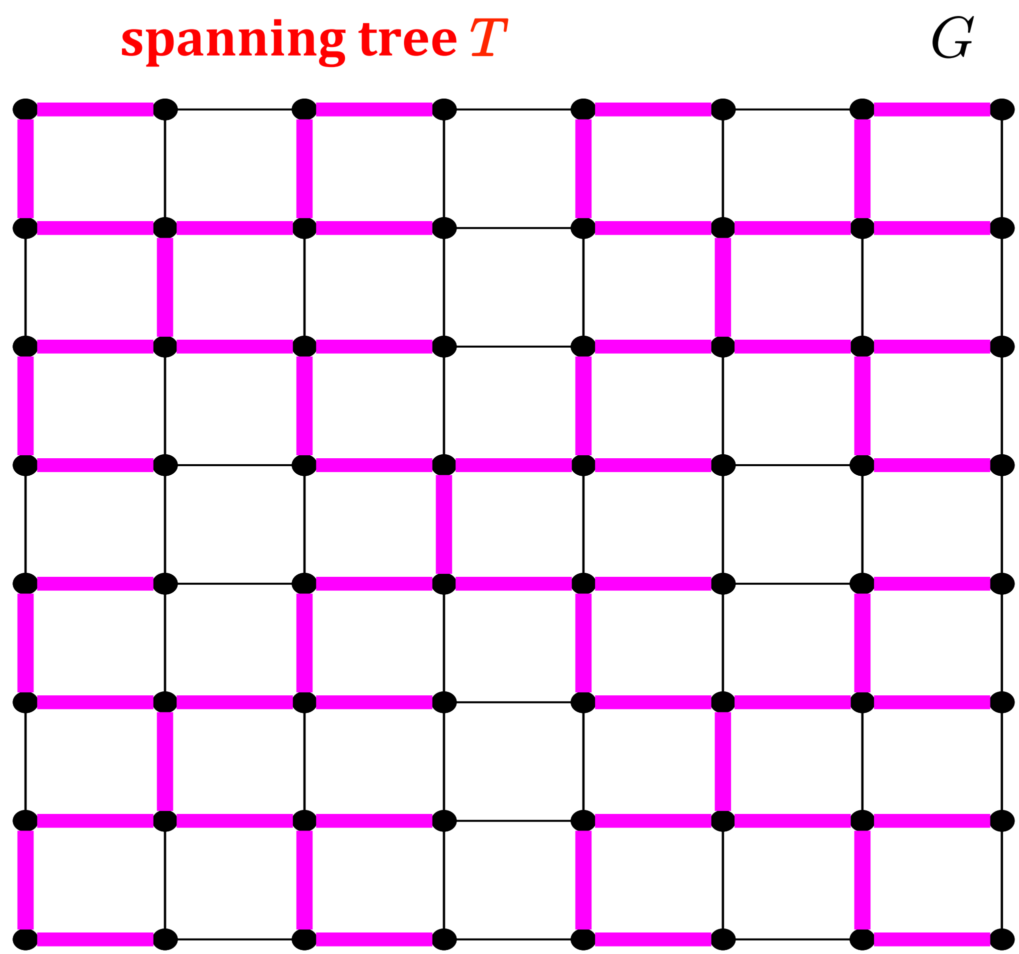
Average Stretch:

$$\mathrm{st}(T) = \frac{1}{m} \sum_{e \in E} \mathrm{st}(e)$$



$$\mathrm{st}(T) = \Omega(\sqrt{n})$$

Low-Stretch Spanning Trees



Stretch of e
=
Length of cycle C_e

Average Stretch:

$$\mathrm{st}(T) = \frac{1}{m} \sum_{e \in E} \mathrm{st}(e)$$

Fact [Abraham, Neiman '12]: It is possible to compute a spanning tree with **average stretch** O(log n loglog n) in almost-linear time.

Algorithm Analysis

- Pick a cycle
- Fix it
- Repeat

Does it converge? → **YES**. Converges to electrical flow.

How quickly ? → Depends on:

1. Choice of **spanning tree**
   Use low-stretch spanning tree

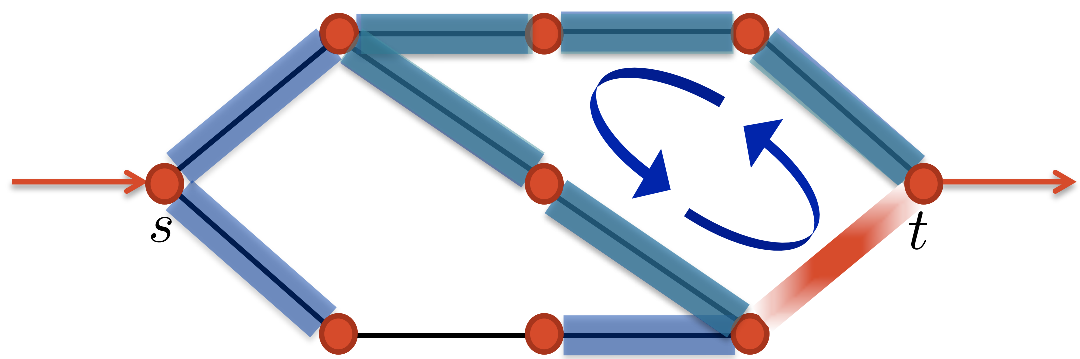
2. Order of **cycle updates**
   Randomized

Number of iterations is $O(m)\cdot$ [average cycle-length] = $O(m \log n \log \log n)$

Each cycle update can be implemented in $O(\log n)$ time using simple data structure

TOTAL RUNNING TIME: $O(m \log^2 n \log \log n)$

Summary of Laplacian Solver



- Simple algorithm based on cycle updates

- Practically appealing: Generated interests from practitioners and is being implemented by groups at UCSB and Sandia Labs

- Numerical stability is very easy to prove

- Formalizes Kaczmarz heuristic used in Computerized Tomography

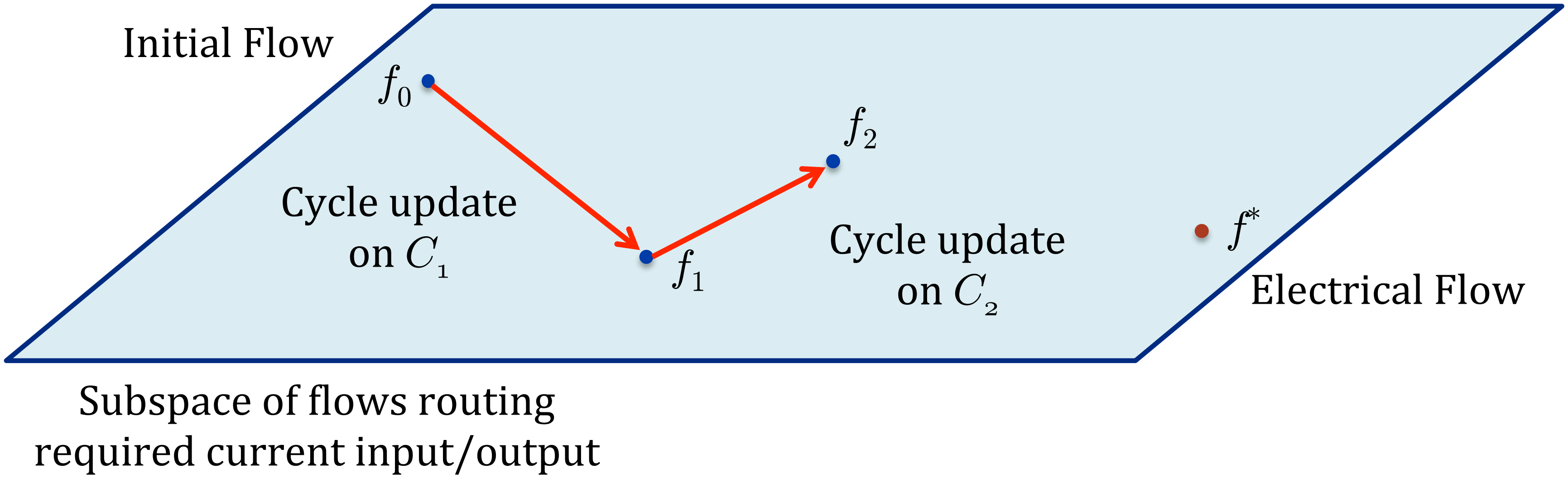
- Replaces more complicated setup based on Spielman-Teng

A Novel Framework for the Design of Almost-Linear-Time Graph Algorithms:

Generalizing Our Approach to Electrical Flow

Working in the Space of Cycles

START: Geometric interpretation of electrical flow algorithm



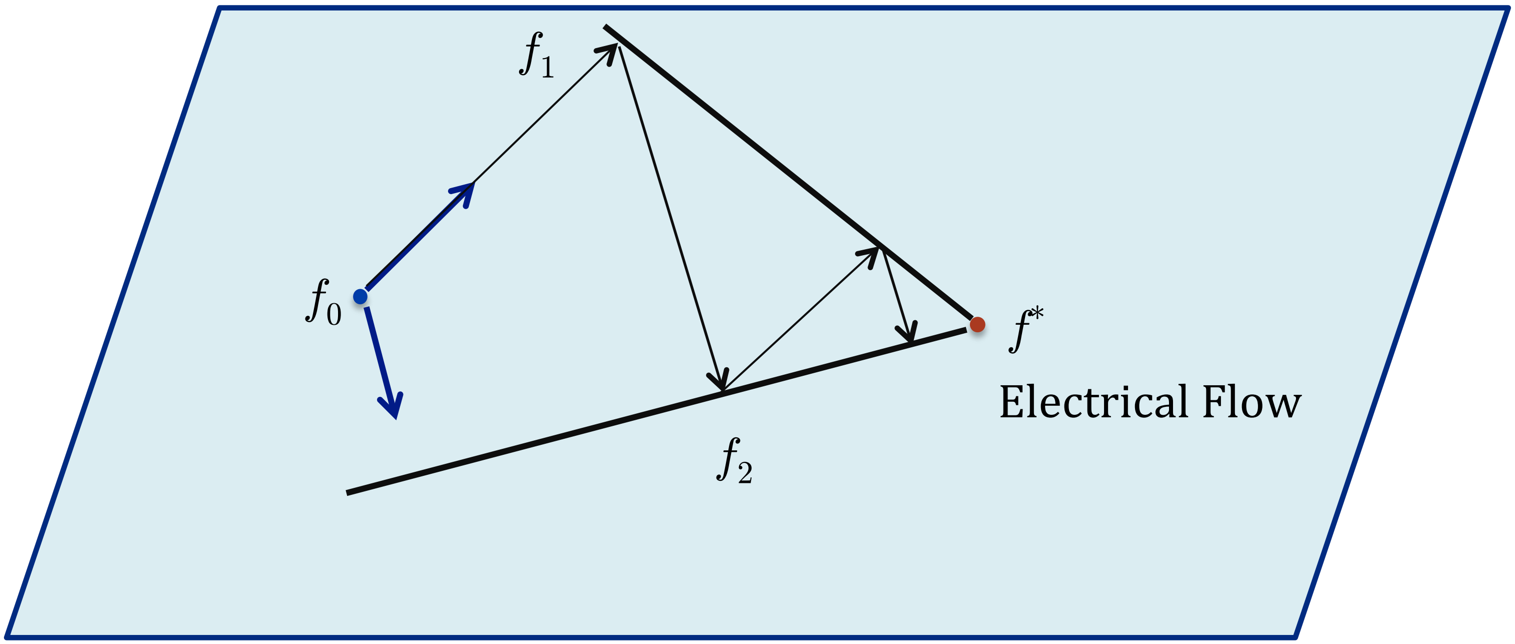
NB: Our iterative solutions never leave this subspace thanks to cycle updates

$$f^* \approxeq f_0 + \sum_i \alpha_i C_i$$

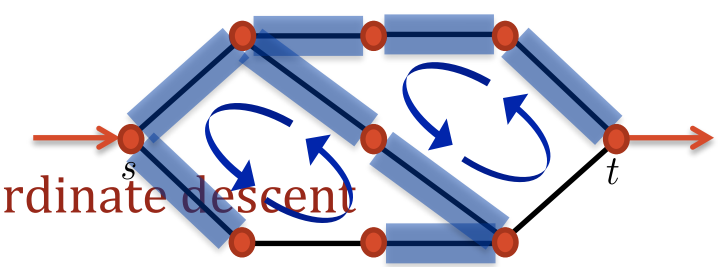
Linear combination of cycles

Coordinate Descent in the Space of Cycles

GOAL: $$f^* \approxeq f_0 + \sum_i \alpha_i C_i$$

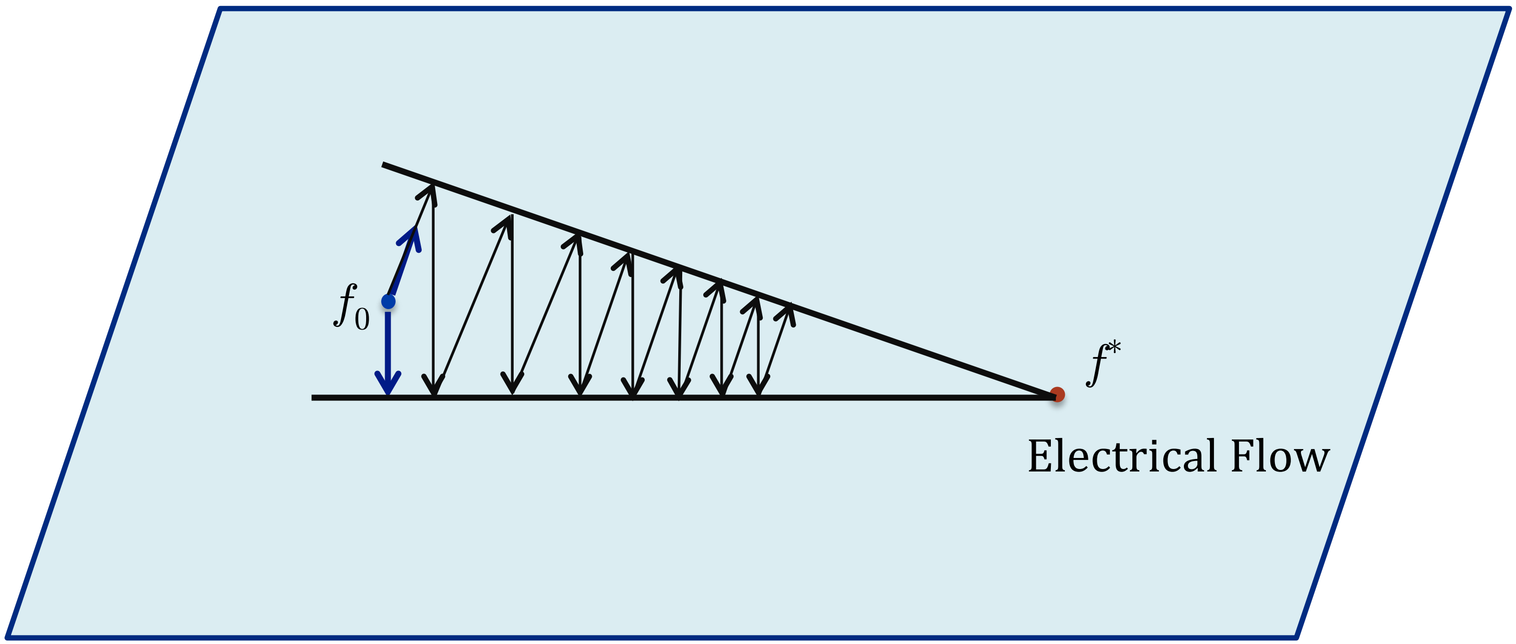


- Pick a basis of the space of cycles
- Fix a coordinate at the time, i.e., coordinate descent



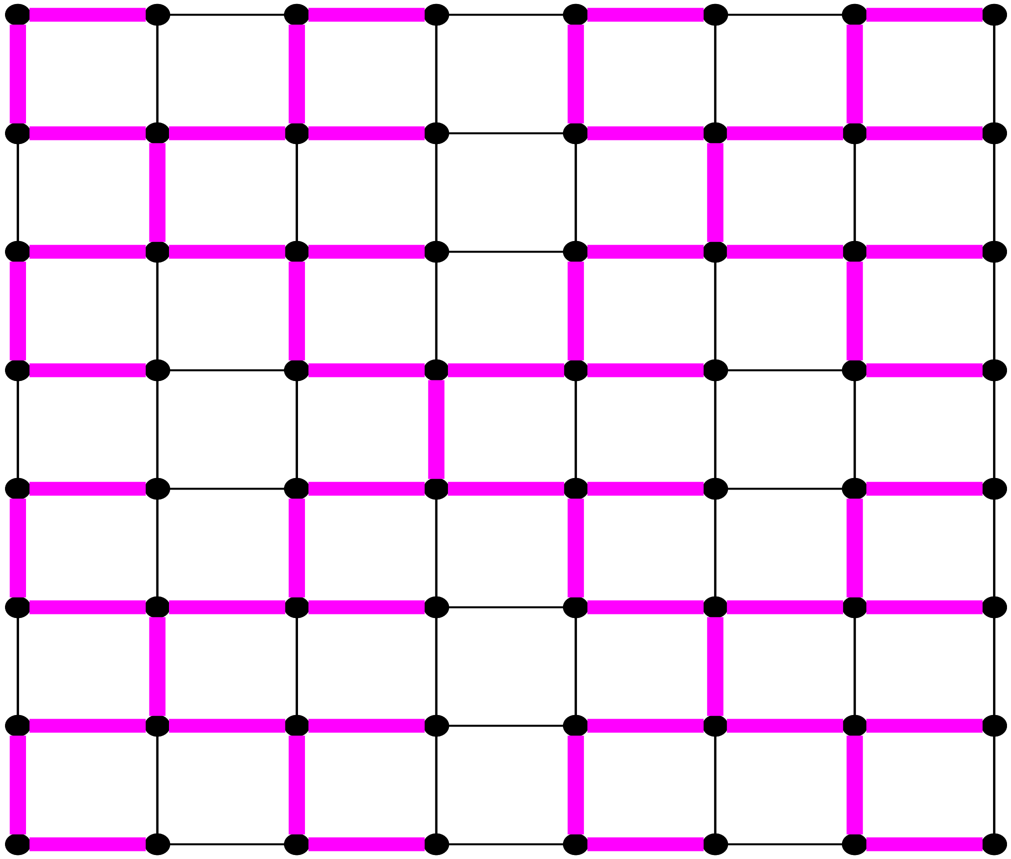
Coordinate Descent in the Space of Cycles

GOAL: $$f^* \approxeq f_0 + \sum_i \alpha_i C_i$$



EXAMPLE:
High interference between basis vectors yields slow convergence

Coordinate Descent in the Space of Cycles



RECALL:
Low-stretch spanning tree yields low-interference basis

Electrical Flow: Algorithmic Components

ITERATIVE METHOD

Continuous Optimization:

Randomized Coordinate Descent

Joint Design of Components

PROBLEM REPRESENTATION

Combinatorial Optimization:

- Space of cycles
- Basis given by **Low-Stretch Spanning Tree**

A Framework for Algorithmic Design

ITERATIVE
METHOD

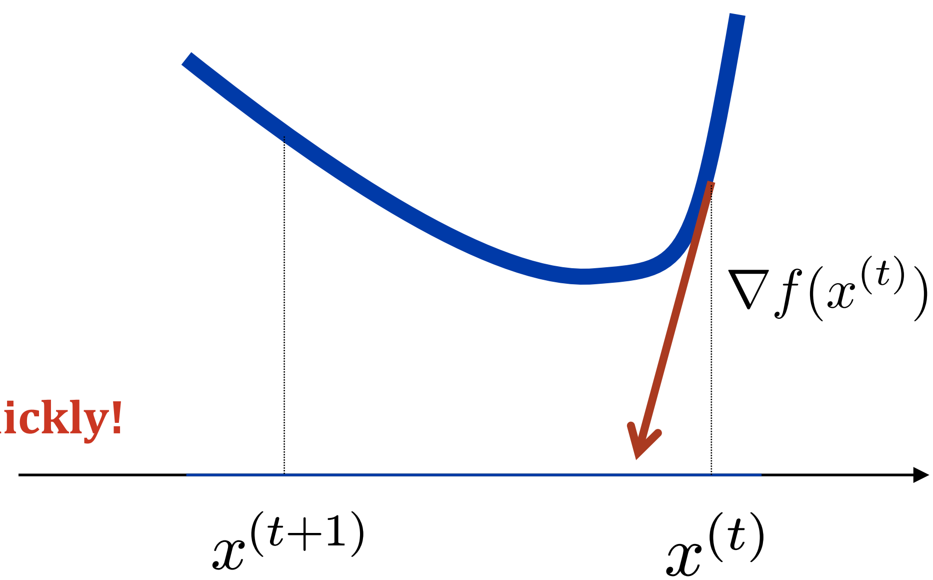
Leverage Continuous Optimization Ideas:

- Gradient Descent
- Coordinate Descent
- Nesterov's Algorithm

Fast convergence of these methods depends on **smoothness** of objective function

$$\min_{x \in X} f(x)$$

Gradient changes too quickly!



A Framework for Algorithmic Design

ITERATIVE
METHOD

Leverage Continuous Optimization Ideas

Fast convergence requires **smooth** problem

PROBLEM
REPRESENTATION

Not all representations are created equal:

Use combinatorial techniques

to produce a smooth representation

Efficiency and simplicity rely on **combining these two components** in the right way

Applying Our Design Framework:

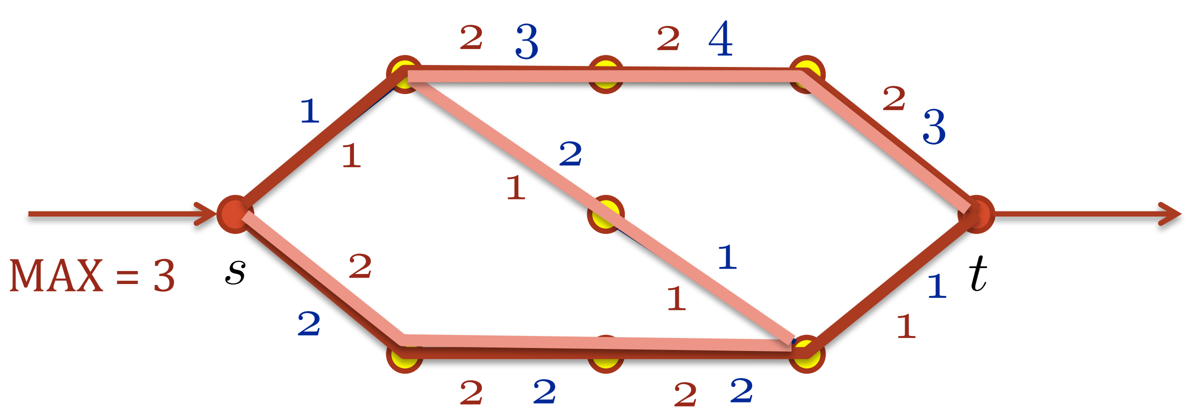
Undirected s-t Maximum Flow

Example: Undirected s-t Maximum Flow

INPUT:

- Undirected Graph $G=(\mathrm{V},\mathrm{E})$, n vertices in V, m edges in E
- Edges have positive capacities c_e
- Special vertices: source s and sink t

GOAL: Route maximum flow from s to t while respecting capacities



Previous Work: Augmenting Paths

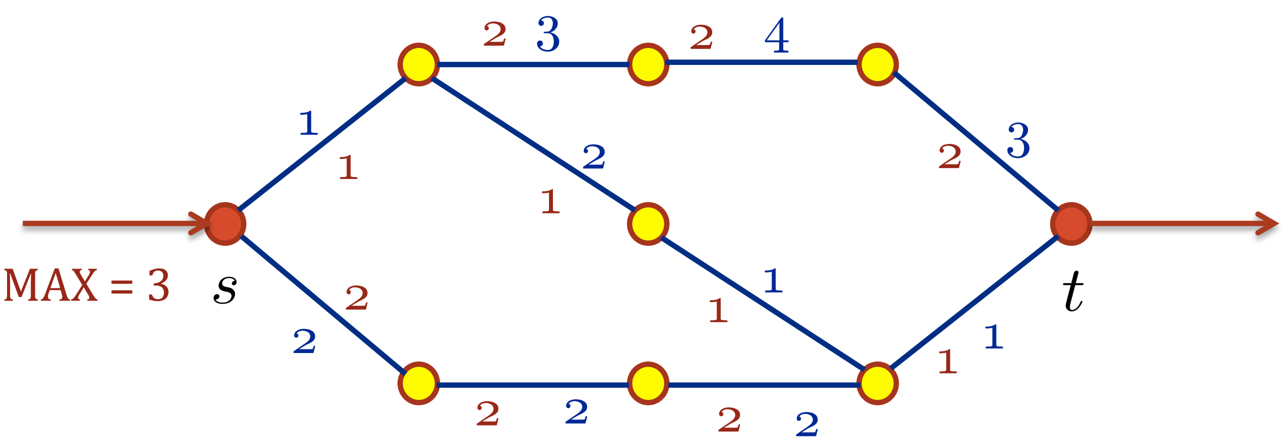
IDEA: Route one path at the time until one edge is congested.
Modify graph to allow pushing flow back.
Repeat.

Different policies for choosing augmentation lead to different variants.

DISCRETE ALGORITHM: Intermediate flows routed are always **integral**.
Convergence analysis is combinatorial.

Running Times:	[Edmonds, Karp '72]	$O(m^2 n)$
	[Dinic '70]	$O(mn^2)$
	…	
	[Goldberg-Rao '98]	$O(m\sqrt{n})$

An Optimization View of Maximum Flow



Two Equivalent Formulations

Maximize s-t flow
while respecting capacities:

$\longleftrightarrow$

Minimize maximum congestion
while routing unit flow from s to t

$$\forall e \in E, \; \frac{f_e}{c_e} \leq 1$$

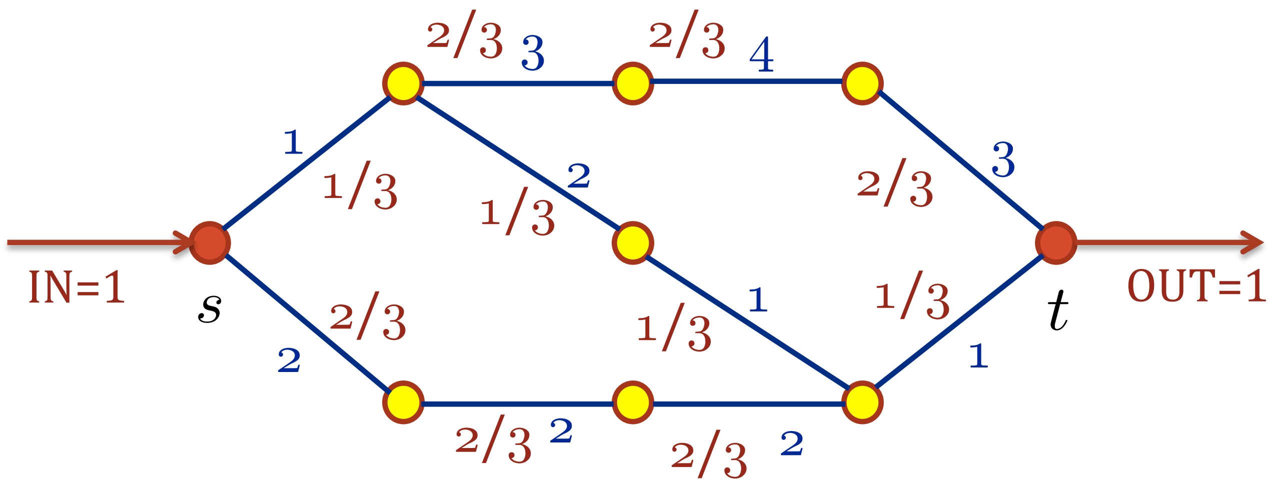
$\updownarrow$

Max edge congestion ≤ 1

$$\min_f \max_e \frac{f_e}{c_e}$$

$$\text{s.t. f routes } s - t$$

An Optimization View of Maximum Flow



Two Equivalent Formulations

Maximize s-t flow
while respecting capacities:

$\longleftrightarrow$

Minimize maximum congestion
while routing unit flow from s to t

$$\forall e \in E, \ \frac{f_e}{c_e} \leq 1$$

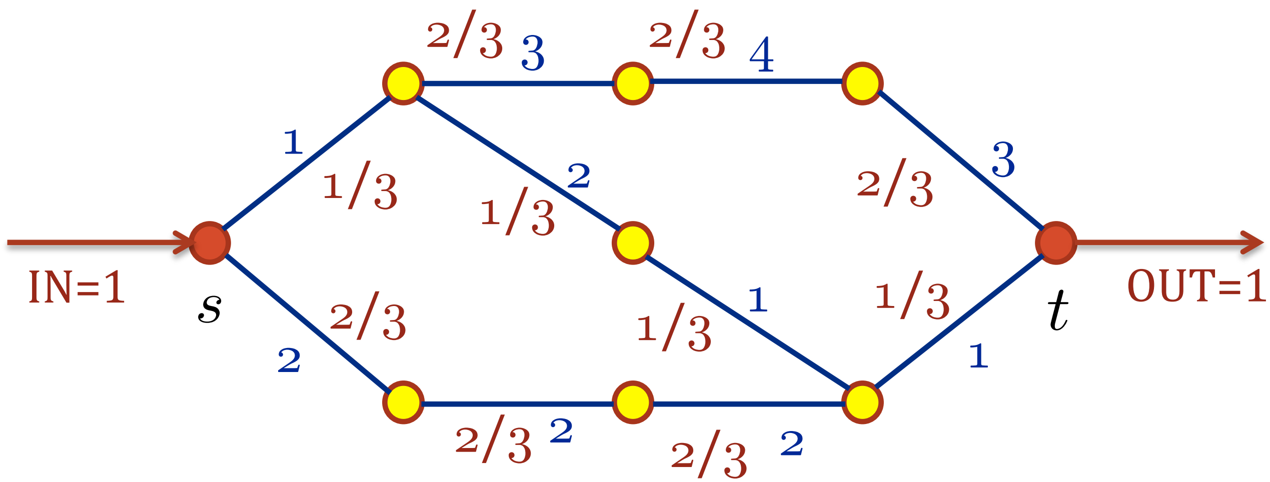
$\updownarrow$

Max edge congestion ≤ 1

$$\min_f \max_e \frac{f_e}{c_e}$$

$$\text{s.t. } \text{f routes } s - t$$

An Optimization View of Maximum Flow



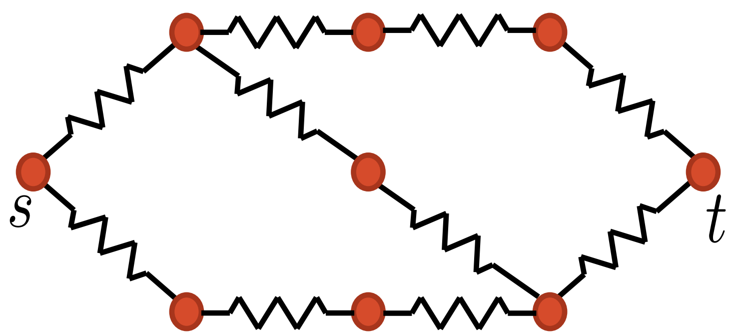
Minimize maximum congestion
while routing unit flow from s to t

$$\min_f \|C^{-1} f\|_\infty$$

$$\text{s.t. f routes } s - t$$

Connection with Electrical Flow

Electrical Flow

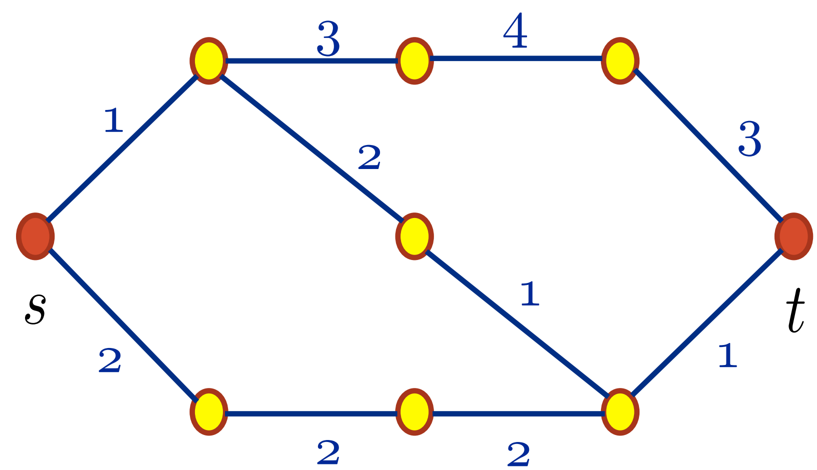


$$\min_f \sum_{e \in E} r_e f_e^2$$

$$\text{s.t. f routes } s - t$$

Energy Minimization

s-t Maximum Flow



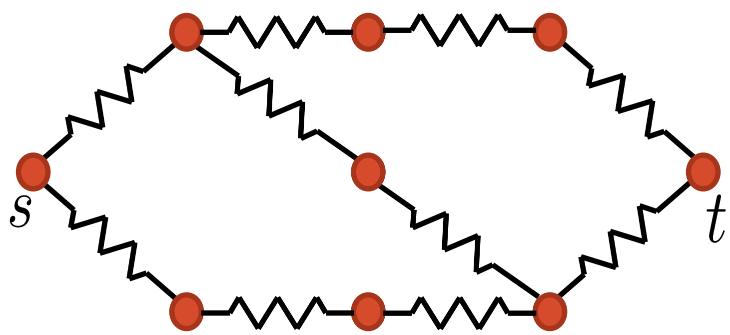
$$\min_f \|C^{-1} f\|_\infty$$

$$\text{s.t. f routes } s - t$$

Congestion Minimization

Connection with Electrical Flow

Electrical Flow

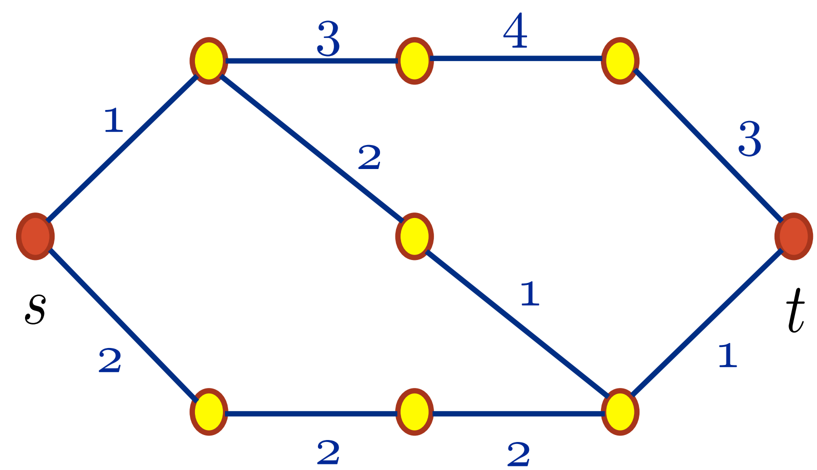


Set resistances as: $r_e = \dfrac{1}{c_e}$

$$\min_f \|C^{-1/2} f\|_2$$

$$\text{s.t. f routes } s-t$$

s-t Maximum Flow



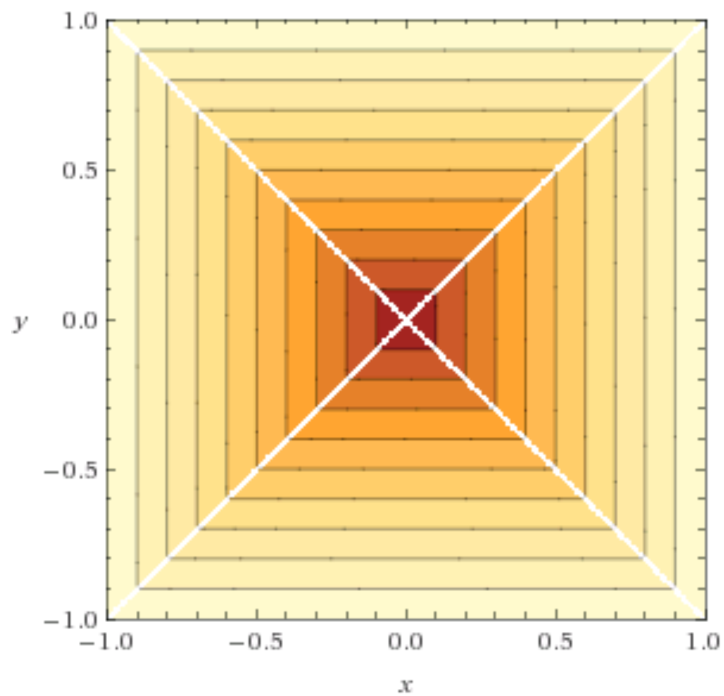
$$\min_f \|C^{-1} f\|_\infty$$

$$\text{s.t. f routes } s-t$$

Applying the framework: Can we change basis to make problem smoother?

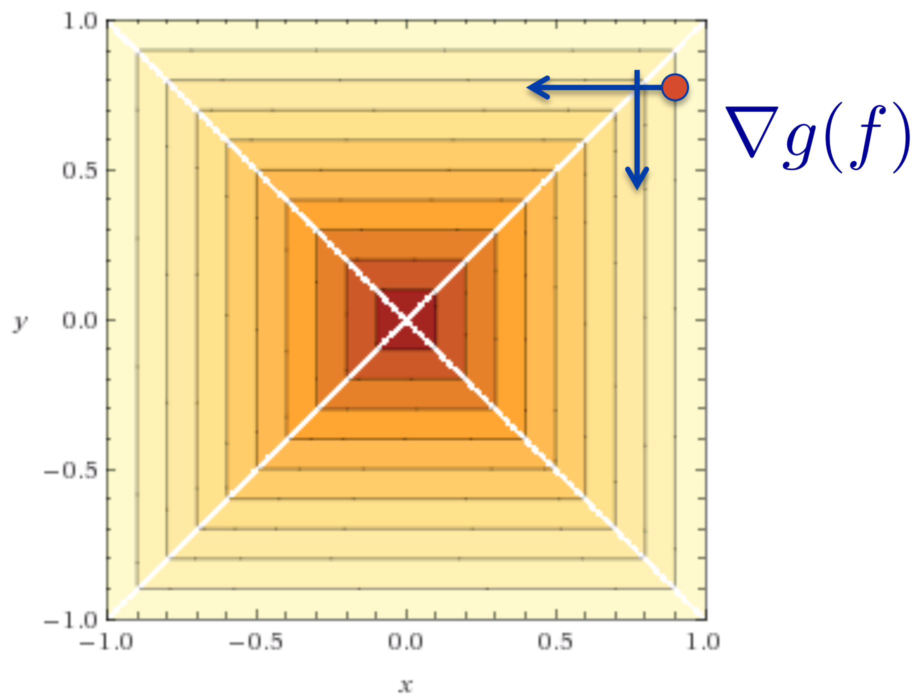
An Extra Difficulty

Objective function: $$g(f) = \|C^{-1}f\|_\infty$$



An Extra Difficulty

Objective function: $$g(f) = \|C^{-1}f\|_\infty$$



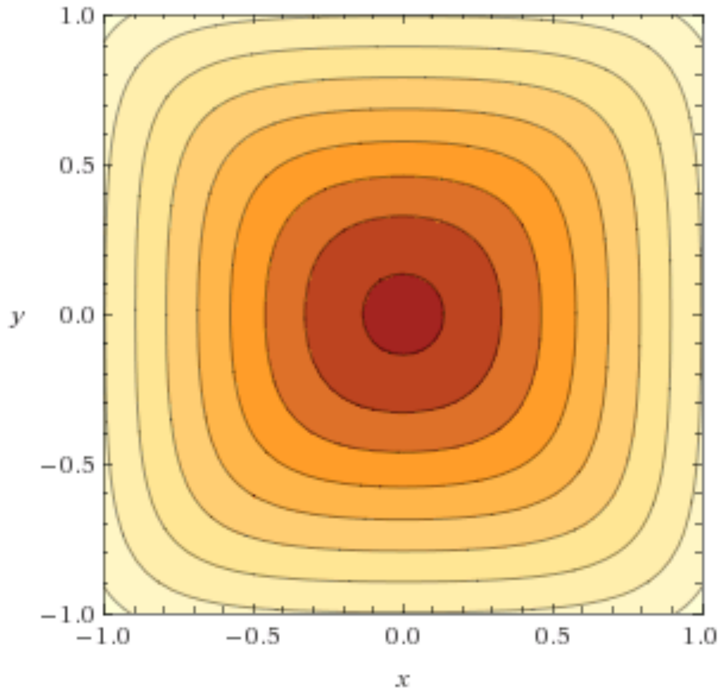
PROBLEM: OBJECTIVE IS EXTREMELY NON-SMOOTH

No change of basis can help

An Extra Step: Regularization

Objective function: $g(f) = \|C^{-1}f\|_\infty$

$\mathrm{softmax}(C^{-1}f)$



PROBLEM: No change of basis can smoothen objective

SOLUTION: Change objective

Find function that is close to objective but somewhat smooth

Applying the Framework: Comparison

Electrical Flow

s-t Maximum Flow

$$\|C^{-1/2} f\|_2$$

OBJECTIVE

$$\|C^{-1} f\|_\infty$$

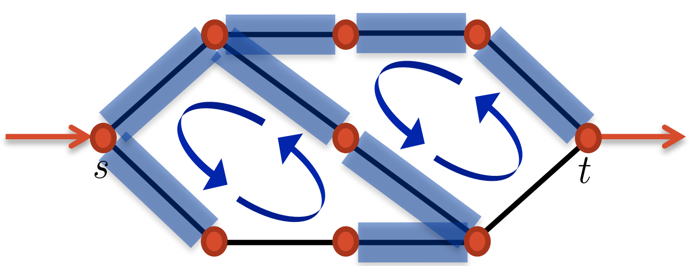
No regularization needed

PROBLEM REPRESENTATION

Regularize to softmax

Use basis given by
Low-stretch spanning tree

Which basis to use?
Little interference in $\|\cdot\|_\infty$



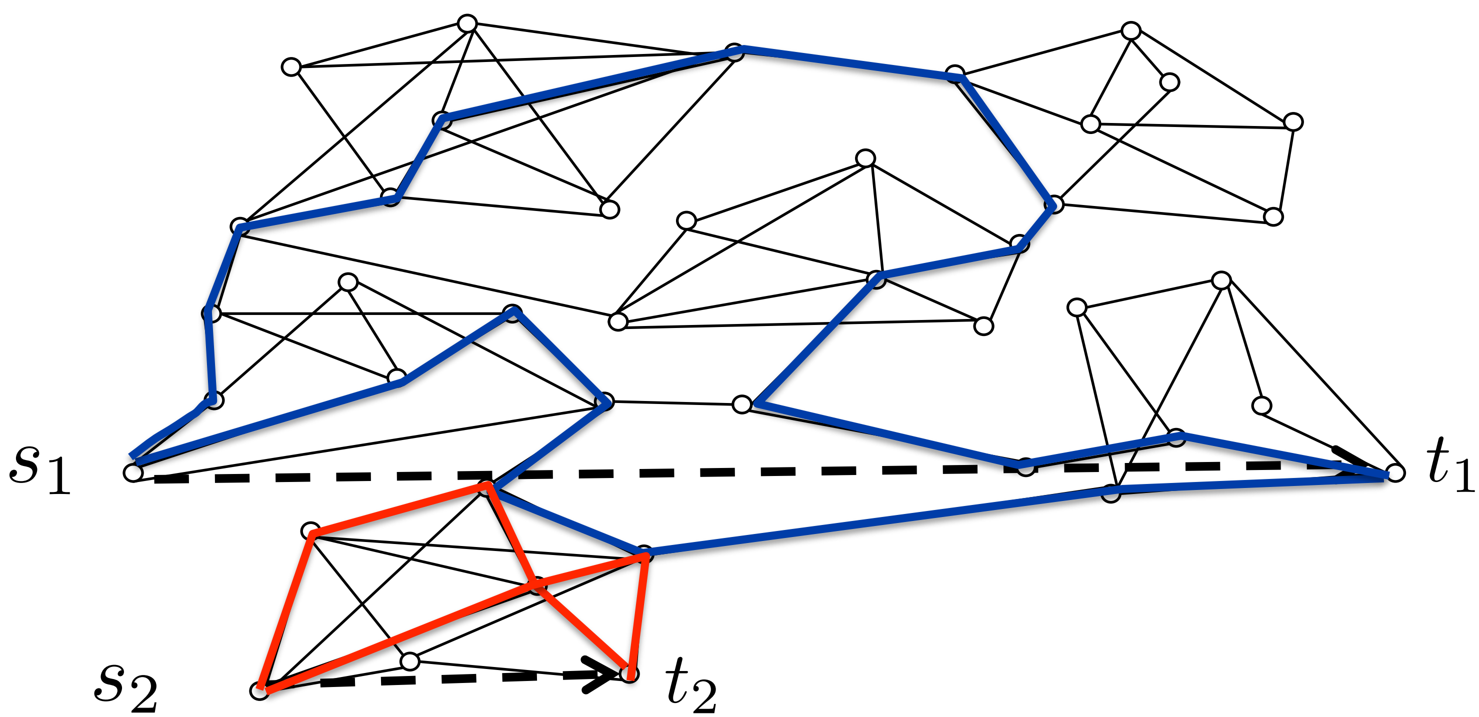
Surprising equivalence:
Basis is
OBLIVIOUS ROUTING SCHEME

ITERATIVE METHOD

Oblivious Routing

GOAL: **Route** traffic between **many pairs** of users on the Internet
Minimize **maximum congestion** of a link

Routing = Probability Distribution over Paths = **Flow**



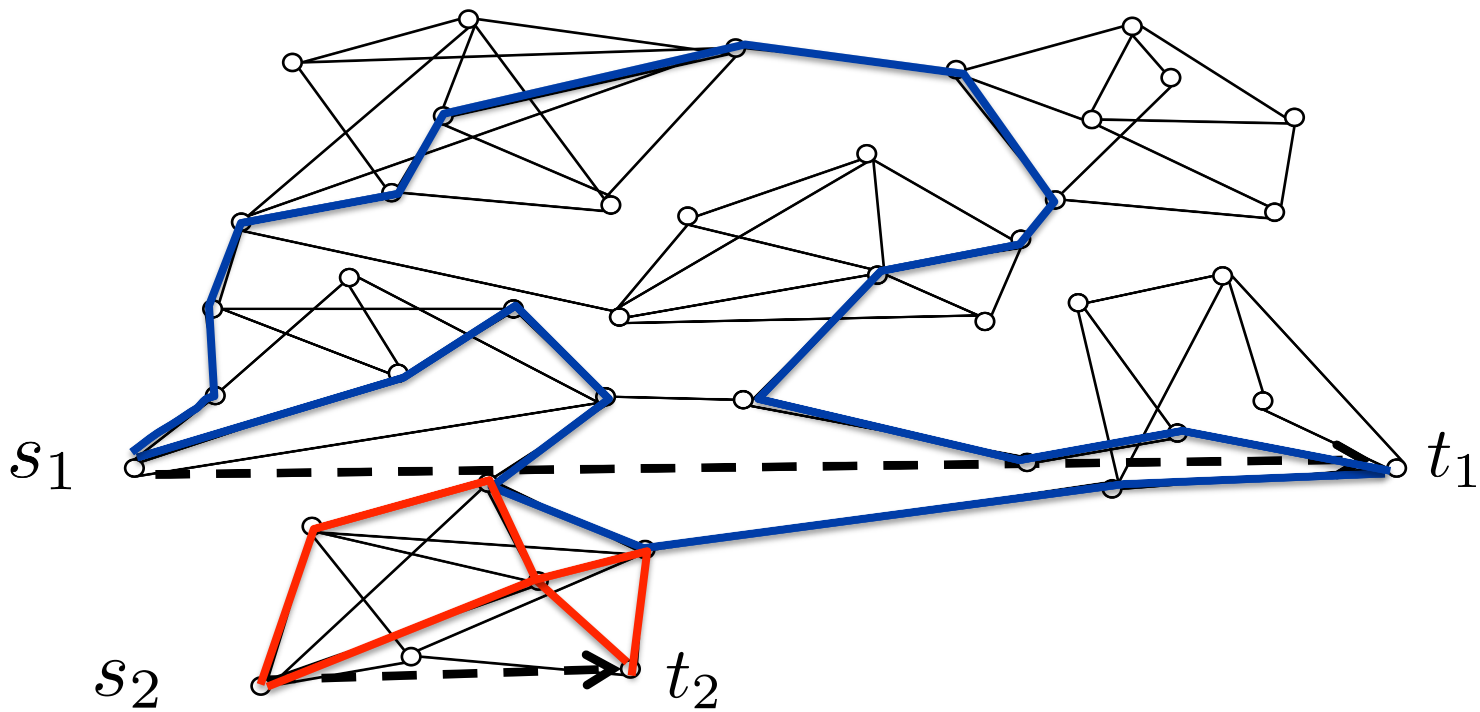
Oblivious Routing

GOAL: **Route** traffic between **many pairs** of users on the Internet
Minimize **maximum congestion** of a link

DIFFICULTY: Requests arrive online in arbitrary order
How to **avoid global flow computation** at every new arrival

SOLUTION:

Oblivious Routing: Every request is **routed obliviously** of the other requests



Oblivious Routing

GOAL: **Route** traffic between **many pairs** of users on the Internet
Minimize **maximum congestion** of a link

DIFFICULTY: Requests arrive online in arbitrary order
How to **avoid global flow computation** at every new arrival

SOLUTION:

Oblivious Routing: Every request is **routed obliviously** of the other requests

PRE-PREPROCESSING: Routes are pre-computed

MEASURE OF PERFORMANCE:

Worst-case ratio between congestion
of oblivious-routing and optimal a posteriori routing

COMPETITIVE RATIO

Oblivious Routing: A New Scheme

GOAL: **Route** traffic between **many pairs** of users on the Internet
Minimize **maximum congestion** of a link

DIFFICULTY: Requests arrive online in arbitrary order
How to **avoid global flow computation** at every new arrival

SOLUTION:

Oblivious Routing: Every request is **routed obliviously** of the other requests

PRE-PREPROCESSING: Routes are pre-computed

ALMOST-LINEAR RUNNING TIME

MEASURE OF PERFORMANCE:

Worst-case ratio between congestion
of oblivious-routing and optimal a posteriori routing

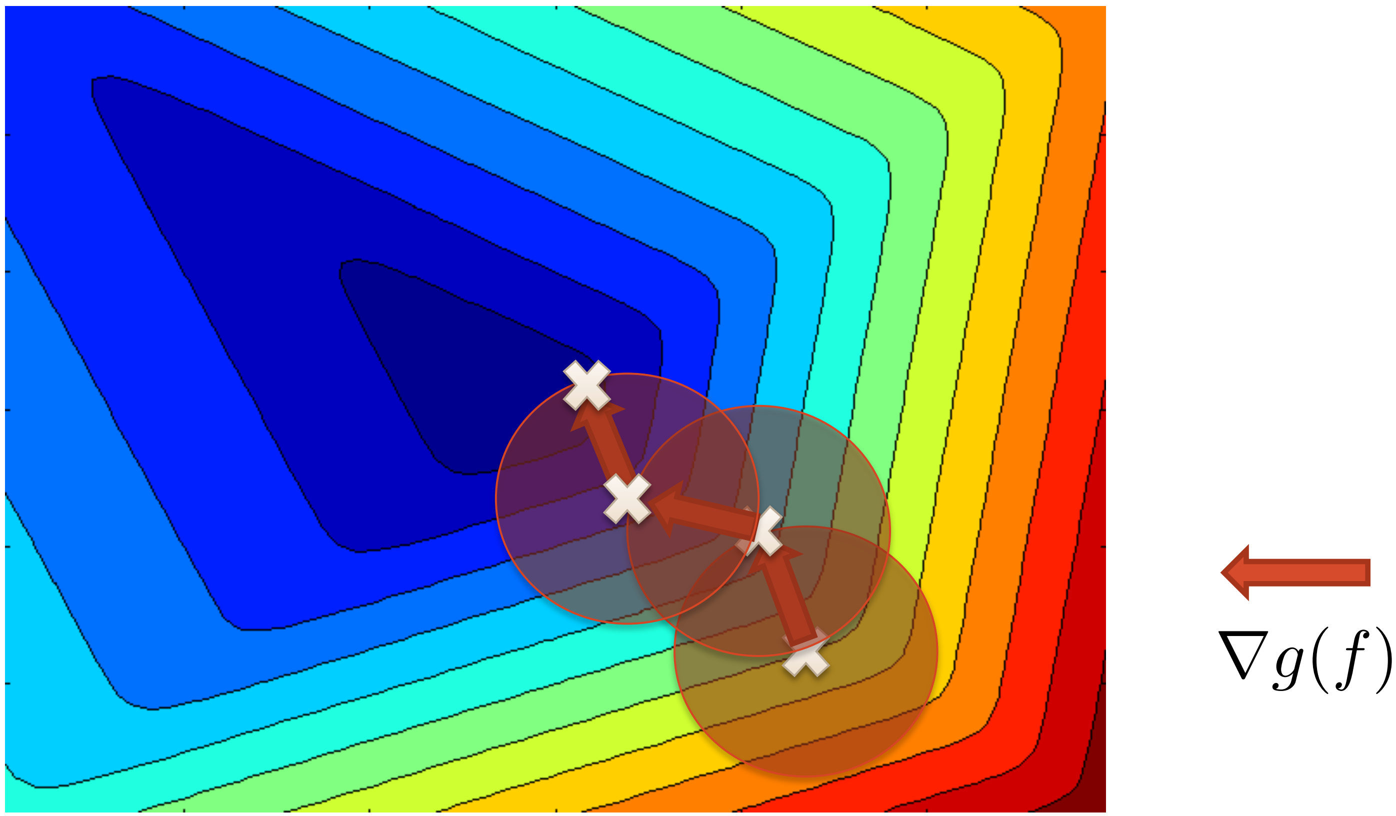
COMPETITIVE RATIO

SUBLINEAR COMPETITIVE RATIO

Applying the Framework: Comparison

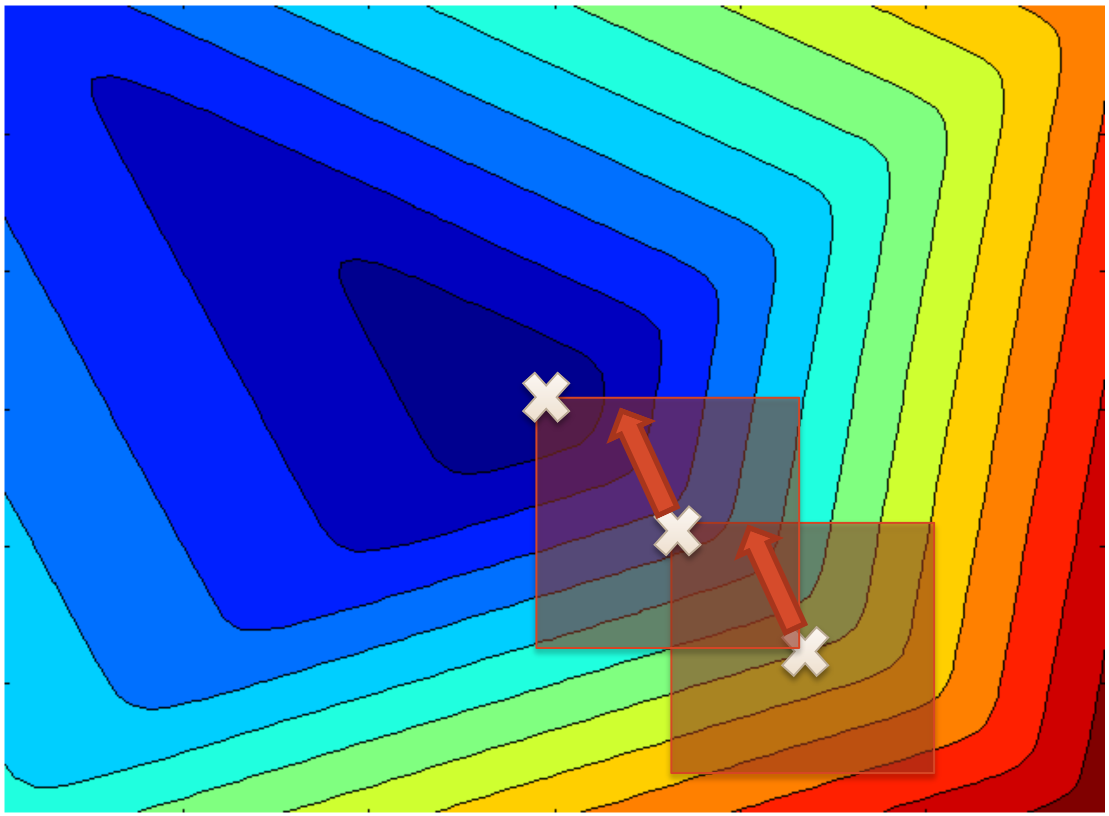
Electrical Flow		s-t Maximum Flow
$\|C^{-1/2} f\|_2$	OBJECTIVE	$\|C^{-1} f\|_\infty$
No regularization needed Use basis given by low-stretch spanning tree	PROBLEM REPRESENTATION	Regularize to softmax Use basis given by oblivious routing scheme
Coordinate Descent	ITERATIVE METHOD	Non-Euclidean Gradient Descent

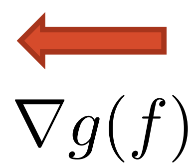
Euclidean Gradient Descent



Contour map of $g(f) = \|C^{-1}f\|_\infty$ over feasible subspace

Non-Euclidean Gradient Descent





Contour map of $g(f) = \|C^{-1} f\|_\infty$ over feasible subspace

Applying the Framework: Comparison

Electrical Flow		s-t Maximum Flow
$\|C^{-1/2}f\|_2$	OBJECTIVE	$\|C^{-1}f\|_\infty$
No regularization needed Use basis given by low-stretch spanning tree	PROBLEM REPRESENTATION	Regularize to softmax Use basis given by oblivious routing scheme
Coordinate Descent	ITERATIVE METHOD	Non-Euclidean Gradient Descent

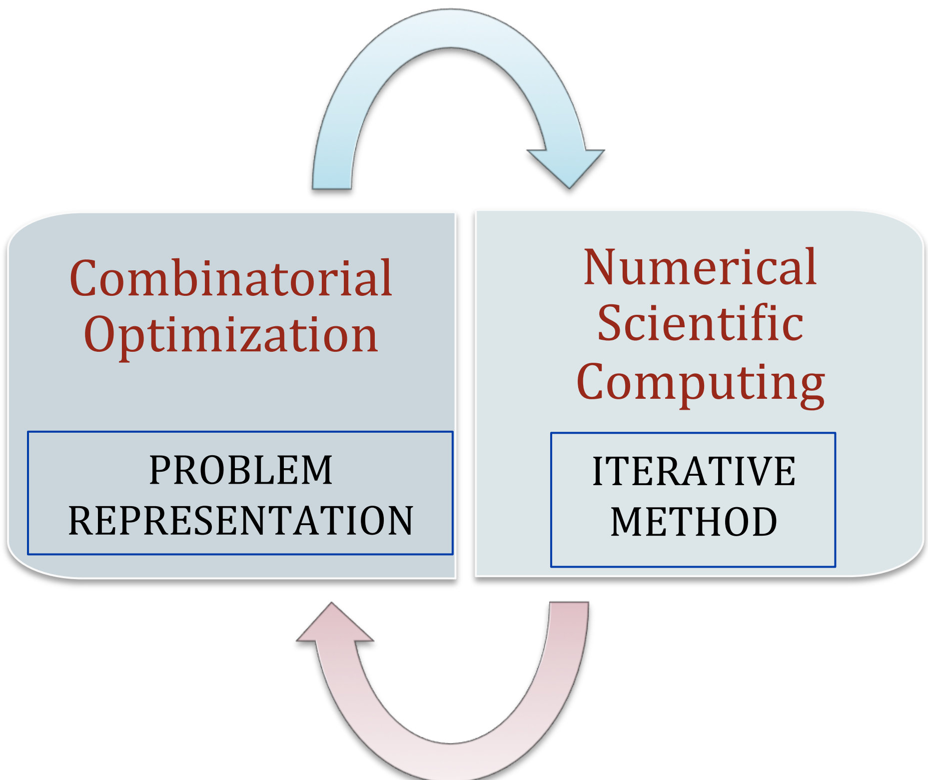
ALMOST-LINEAR-TIME FOR BOTH PROBLEMS

Where Do We Go From Here?

Future Directions

A New Algorithmic Approach

- A novel design framework for fast graph algorithms
- Incorporates and leverages idea from multiple fields
- Based on radically different approach
- Has yielded conceptually simple, powerful algorithms
- Combinatorial insight plays a crucial role
  - Low-stretch spanning trees
  - Oblivious routings
- Numerous potential applications in Algorithms and other fields



What Are the Limits of Almost-Linear Time?

Almost-Linear Time

Reachability
Shortest Path
Connectivity
Minimum Cost Spanning Tree
…

Super-linear Time

?

Directed Flow Problems
Directed Cut Problems

All-pair Shortest Path
Network Design
…

Laplacian Systems of Linear Equations

Undirected Flow Problems:

- s-t Maximum Flow
- Concurrent Multi-commodity Flow
- Oblivious Routing

Undirected Cut Problems:

- Minimum s-t cut Approximate Sparsest Cut
- Approximate Minimum Conductance Cut

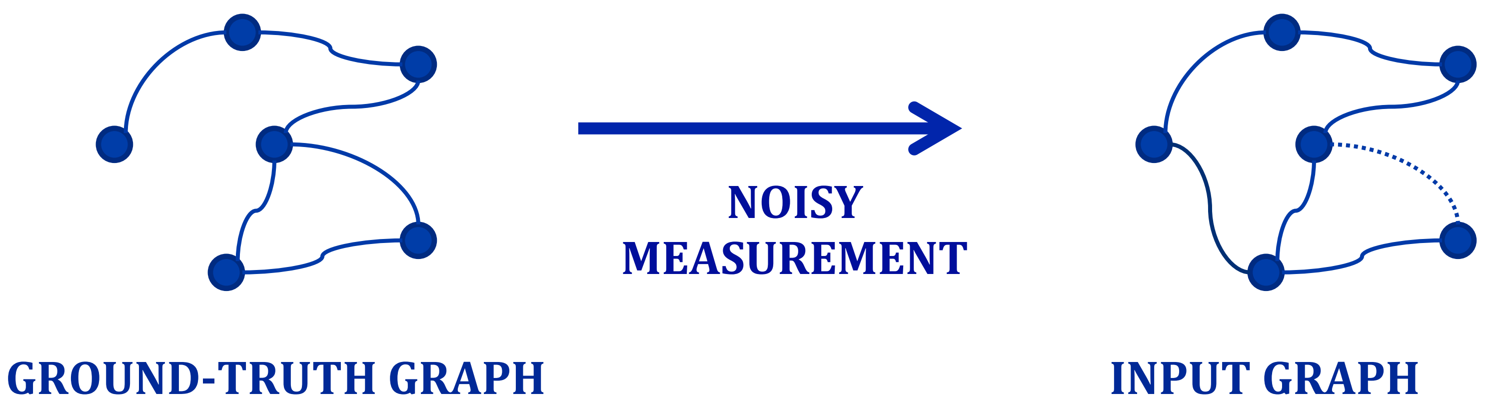
Recent Partial Progress:

- Improved running time for directed flow problems [Madry'13]
- Conditional lowerbounds for All-Pair Shortest Path [Williams'13]

Properties of Resulting Algorithms

OBSERVE: Our algorithms solve regularized versions of the problem

SOLUTIONS ARE STABLE UNDER NOISE



Our iterative solutions are stable under noise

Practical Advantage: Real-world Instances are often noisy samples

REGULARIZATION PREVENTS OVERFITTING TO NOISE

CONNECTIONS TO: Convex Optimization, Machine Learning, Statistics, Complexity Theory

Connecting Theory and Practice

EMPIRICAL OBSERVATION:

Many of the algorithms obtained in this framework
resemble heuristics used successfully in practice

Examples:

- METIS for Graph Partitioning
- PageRank Random Walks for Clustering
- Kaczmarz Iteration for Solving Linear Systems

Future Work: <u>Interpret and improve existing heuristics</u>

Example: Clustering heuristics in computational biology

A Modern Theory of Algorithms

BROAD VISION:

Convergence of Combinatorial and Continuous Optimization
yields new approach to the design of algorithms

PERSPECTIVE: We have only made first steps in leveraging this insight

5-10 year plan: much richer toolkit of almost-linear-time algorithms

RENEWED FOCUS ON PRACTICAL APPLICATIONS:

- Scalability
- Conceptual simplicity, practical appeal
- Address fundamental problems with wide applicability

LONG-TERM GOAL:

Redefine the relationship between Theory of Algorithms and other areas:

Scientific Computing, Machine Learning, Experimental Algorithms, and more