Electrical Flows, Optimization, and New Approaches to the Maximum Flow Problem

Aleksander Mądry

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Spectral graph theory: Understanding graphs via eigenvalues and eigenvectors of associated matrices

Central object: Laplacian matrix

"Linear-algebraic" graph theory: Understanding graphs via examining associated linear-algebraic objects

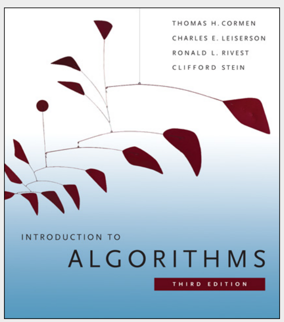
Central object: Electrical flows

Our goal: Incorporate this approach into algorithmic graph theory toolkit

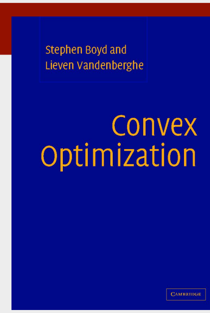
Our focus: Maximum Flow problem

(+ random spanning tree generation)

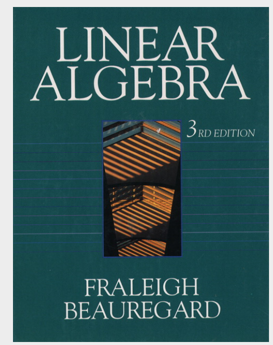
Underlying theme: Merging combinatorial and continuous methods



Combinatorial methods
(trees, paths, partitions, matchings, routings,…)



Convex opt. primitives
(gradient-descent, interior-point methods,…)

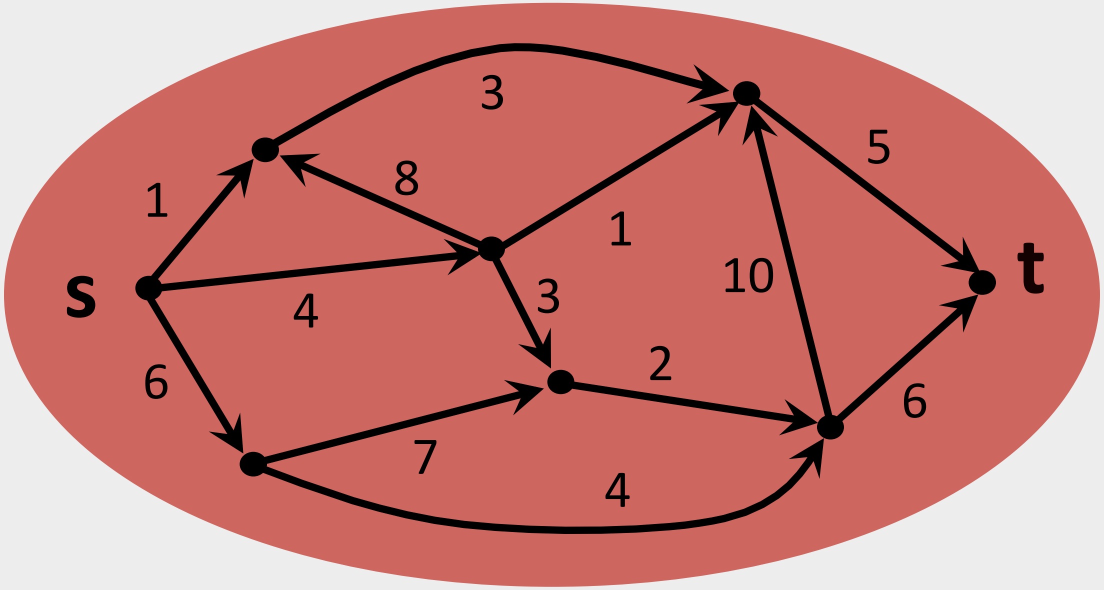


Linear-algebraic tools
(eigenvalues, electrical flows, linear systems,…)

This is a part of a broader agenda

Maximum flow problem

Input: Directed graph **G**, integer **capacities u_e**, **source s** and **sink t**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

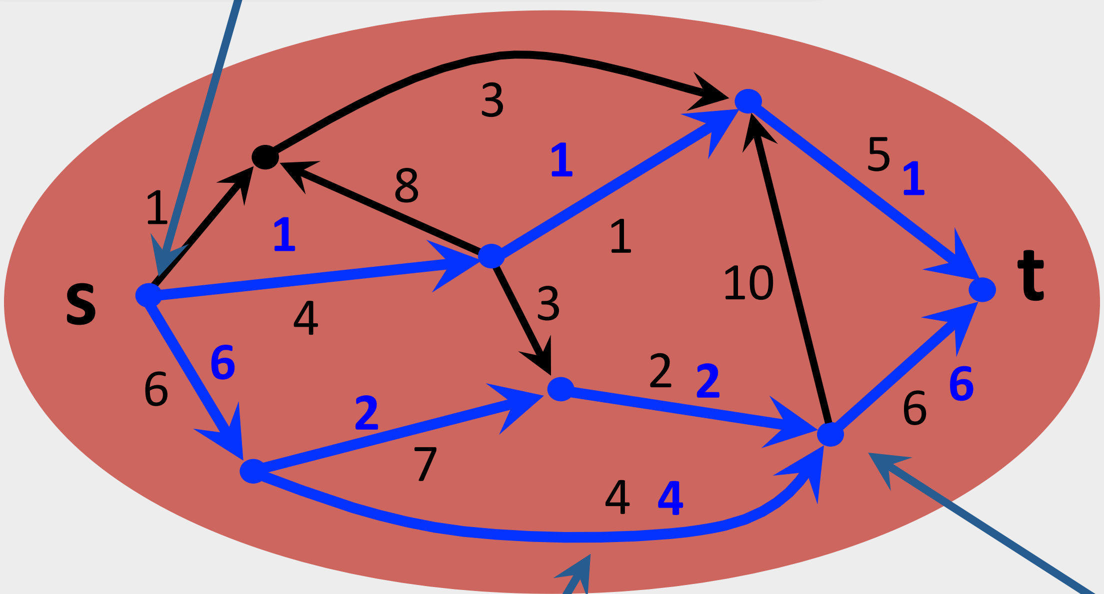
Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Maximum flow problem

Input: Directed graph **G**, integer **capacities u_e**, **source s** and **sink t**

value = net flow out of **s**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

Max flow value
F*=10

no overflow on arcs:
0 ≤ f(e) ≤ u(e)

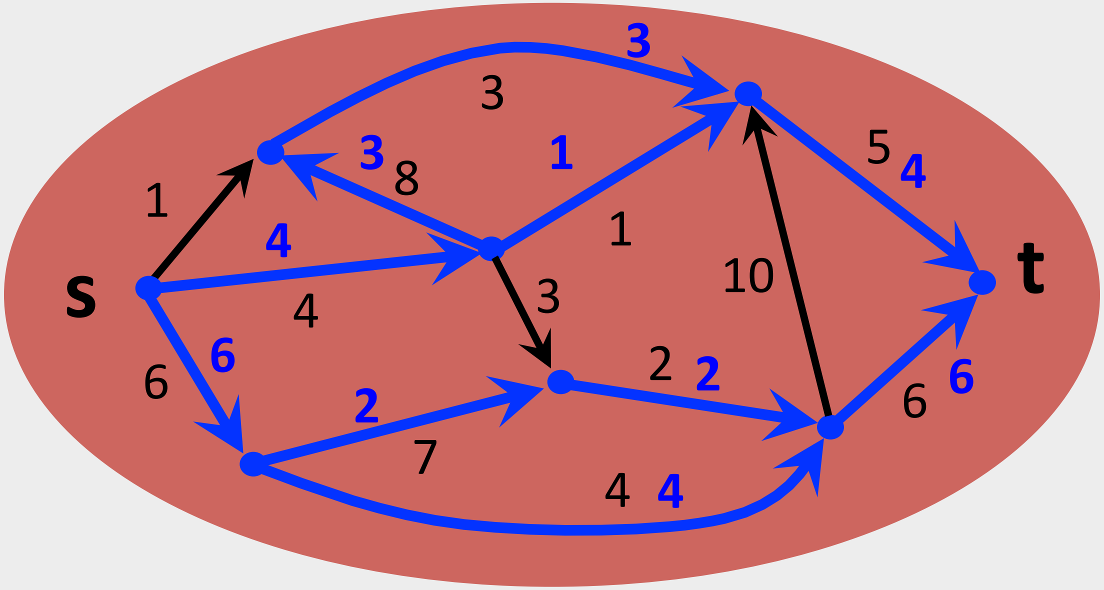
no leaks at all **v≠s,t**

Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Maximum flow problem

Input: Directed graph **G**,
integer **capacities u_e**,
source s and **sink t**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

Max flow value
F*=10

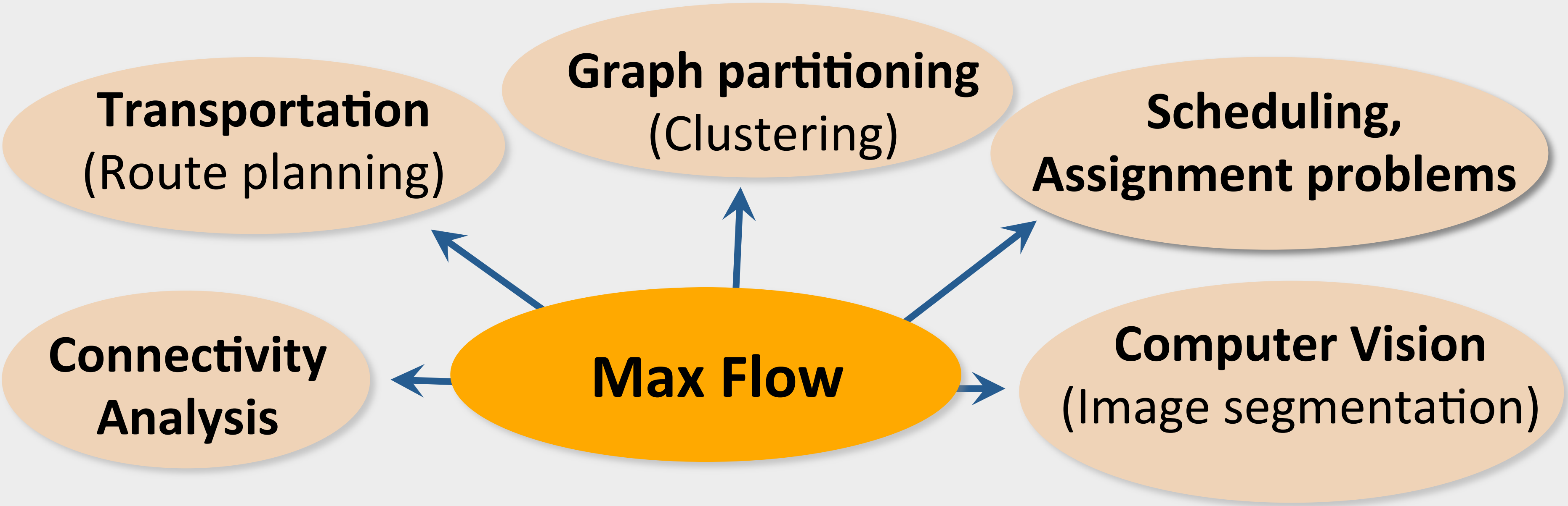
Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Why is this a good problem to study?

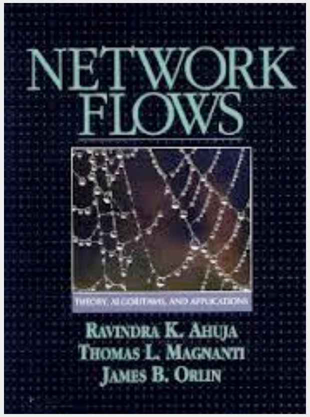
Max flow is a fundamental optimization problem

- **Extensively studied since 1930s** (classic 'textbook problem')
- **Surprisingly diverse set of applications**
- **Very influential in development of (graph) algorithms**



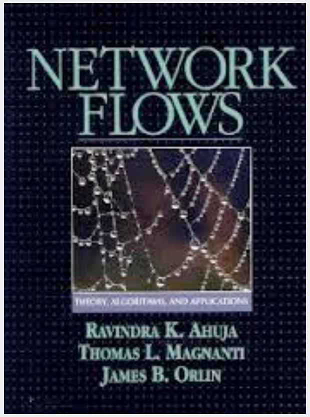
What is known about Max Flow?

A **LOT** of previous work



What is known about Max Flow?

A (very) rough history outline



[Dantzig '51]	$O(mn^2 U)$
[Ford Fulkerson '56]	$O(mn U)$
[Dinitz '70]	$O(mn^2)$
[Dinitz '70] [Edmonds Karp '72]	$O(m^2 n)$
[Dinitz '73] [Edmonds Karp '72]	$O(m^2 \log U)$
[Dinitz '73] [Gabow '85]	$O(mn \log U)$
[Goldberg Rao '98]	$\tilde{O}(m \min(m^{1/2},n^{2/3}) \log U)$
[Lee Sidford '14]	$\tilde{O}(mn^{1/2} \log U)$

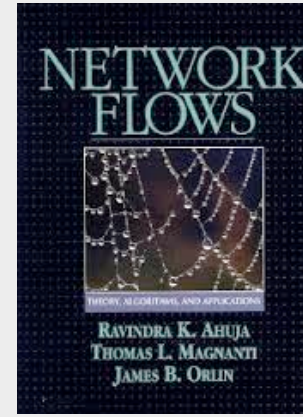
Our focus: Sparse graph (**$m=O(n)$**) and unit-capacity (**$U=1$**) regime

→ It is a good benchmark for combinatorial graph algorithms
→ Already captures interesting problems, e.g., **bipartite matching**

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **$\tilde{O}()$** hides polylogs)

What is known about Max Flow?

A (very) rough history outline



[Dantzig '51]	**$O(n^3)$**
[Ford Fulkerson '56]	**$O(n^2)$**
[Dinitz '70]	**$O(n^3)$**
[Dinitz '70] [Edmonds Karp '72]	**$O(n^3)$**
[Dinitz '73] [Edmonds Karp '72]	**$\tilde{O}(n^2)$**
[Dinitz '73] [Gabow '85]	**$\tilde{O}(n^2)$**
[Goldberg Rao '98]	**$\tilde{O}(n^{3/2})$**
[Lee Sidford '14]	**$\tilde{O}(n^{3/2})$**

Our focus: Sparse graph (**m=O(n)**) and unit-capacity (**U=1**) regime

→ It is a good benchmark for combinatorial graph algorithms
→ Already captures interesting problems, e.g., **bipartite matching**

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **Õ()** hides polylogs)

What is known about Max Flow?

Emerging barrier: $O(n^{3/2})$

[Even Tarjan '75, Karzanov '73]: Achieved this bound for **U=1** long time ago

Last 40 years: Matching this bound in increasingly more general settings, but **no improvement**

This indicates a fundamental limitation of our techniques

Our goal: Show a new approach finally breaking this barrier

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **Õ()** hides polylogs)

Breaking the $O(n^{3/2})$ barrier

Undirected graphs and approx. answers ($O(n^{3/2})$ barrier still holds here)

[M '10]: **Crude approx. of** max flow **value** in **close to linear** time

[CKMST '11]: **(1-ε)-approx.** to max flow in $\tilde{O}(n^{4/3}\varepsilon^{-3})$ time

[LSR '13, S '13, KLOS '14]: **(1-ε)-approx.** in **close to linear** time

Lecture II

But: What about the **directed** and **exact** setting?

[M '13]: Exact $\tilde{O}(n^{10/7})=\tilde{O}(n^{1.43})$-time alg.

Lecture III

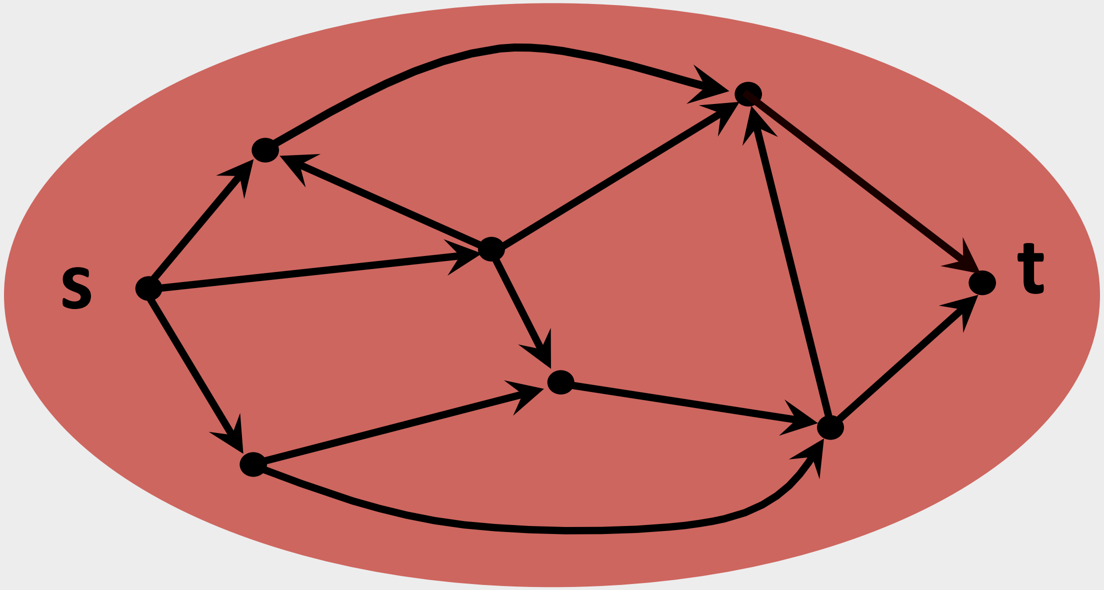
(**n** = # of vertices, $\tilde{O}()$ hides polylog factors)

Previous approach

Augmenting paths framework

[Ford Fulkerson '56]

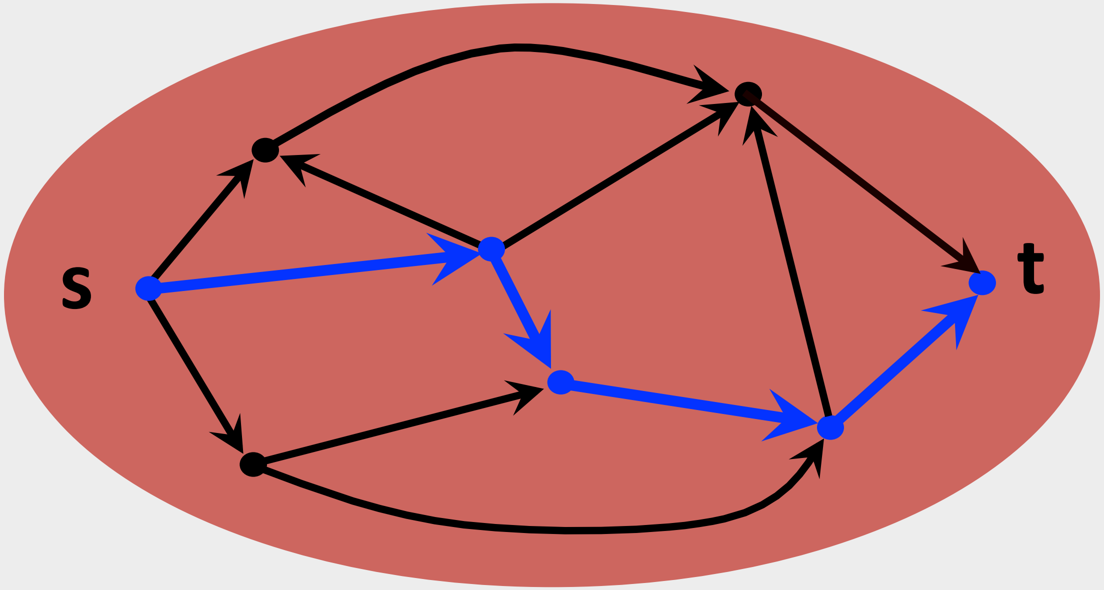
Basic idea: Repeatedly find **s-t paths** in the **residual graph**



Augmenting paths framework

[Ford Fulkerson '56]

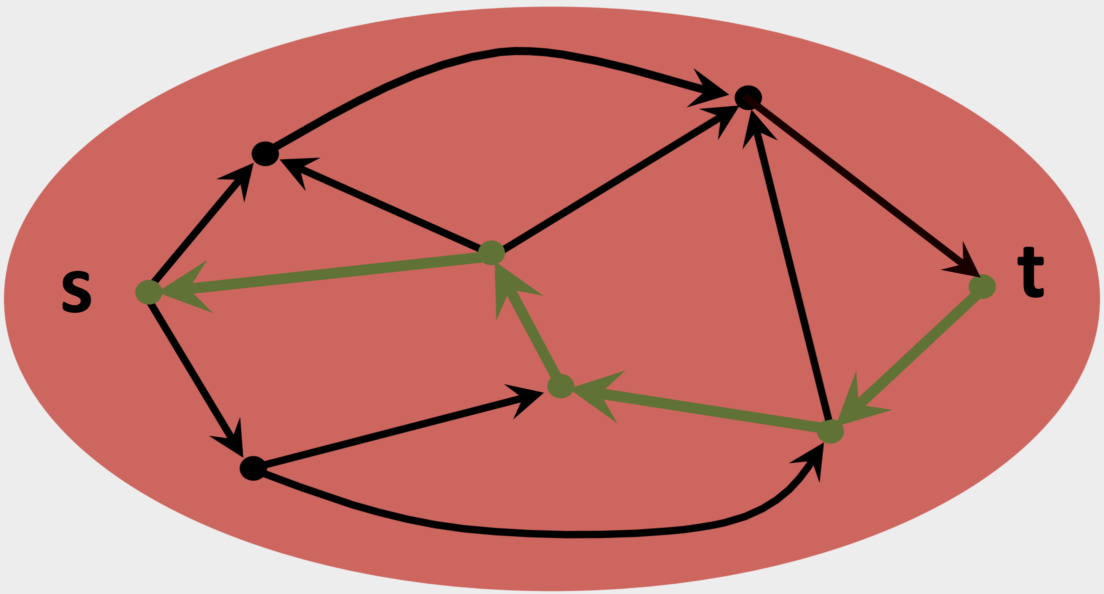
Basic idea: Repeatedly find **s-t paths** in the **residual graph**



Augmenting paths framework

[Ford Fulkerson '56]

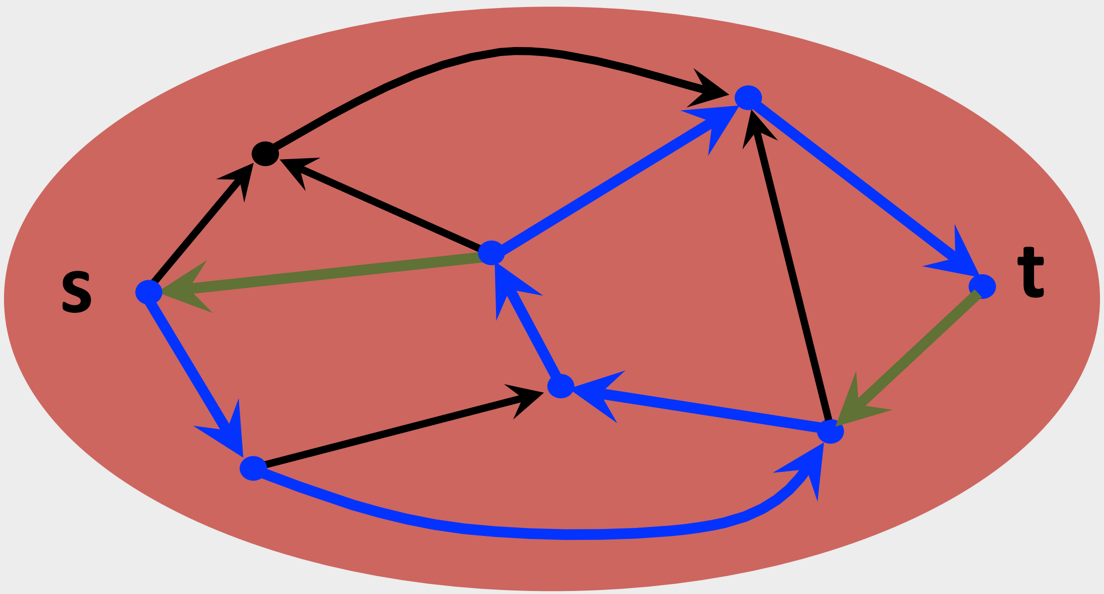
Basic idea: Repeatedly find **s-t paths** in the **residual graph**



Augmenting paths framework

[Ford Fulkerson '56]

Basic idea: Repeatedly find **s-t paths** in the **residual graph**

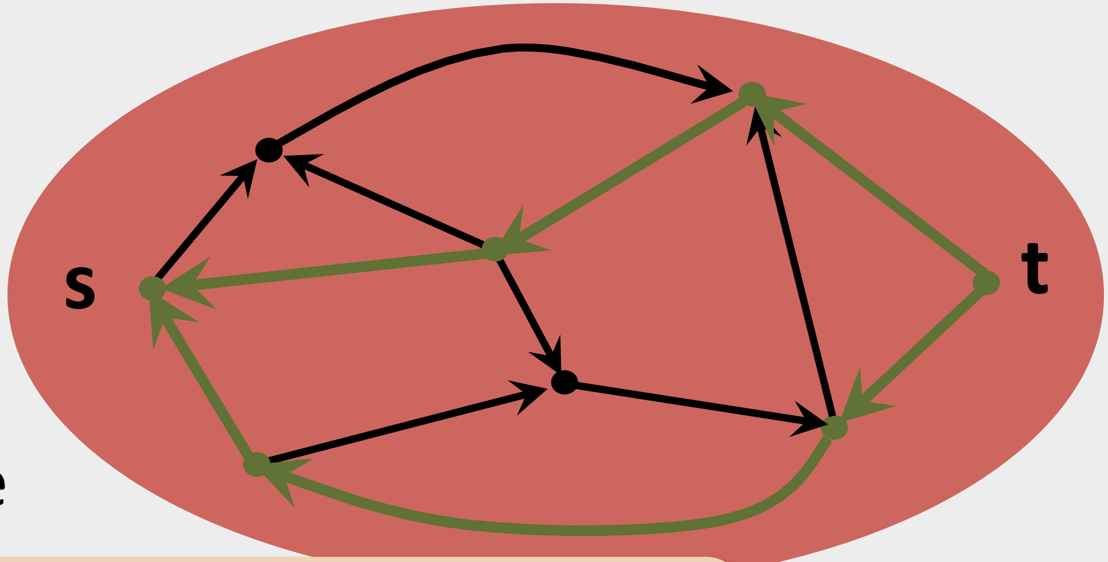


Augmenting paths framework

[Ford Fulkerson '56]

Basic idea: Repeatedly find **s-t paths** in the **residual graph**

Advantage: Simple, purely combinatorial and greedy (flow is built path-by-path)



Problem:
Very difficult to analyze

Naïve impl

(**≤ n** augme   path)

Unclear how to get
a further speed-up via this route

Sophisticated implementation and arguments:

$\mathbf{O(n^{3/2})}$ time **[Karzanov '73] [Even Tarjan '75]**

Beyond augmenting paths

New approach:

Bring linear-algebraic techniques into play

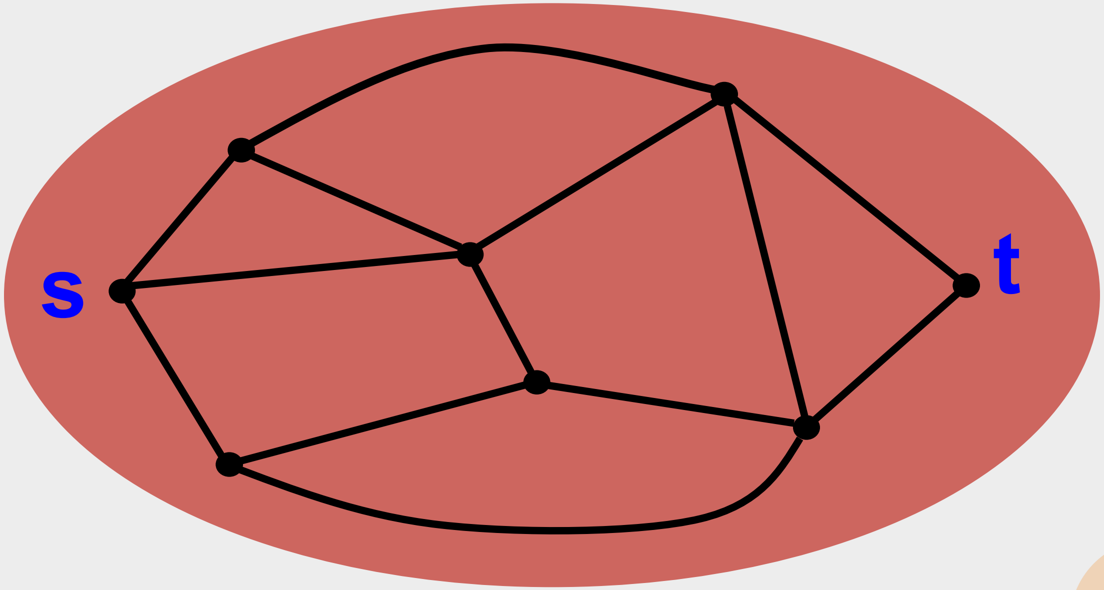
Idea: Probe the **global flow structure** of the graph by **solving linear systems**

How to relate **flow structure** to **linear algebra**?
(And why should it even help?)

Key object: Electrical flows

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



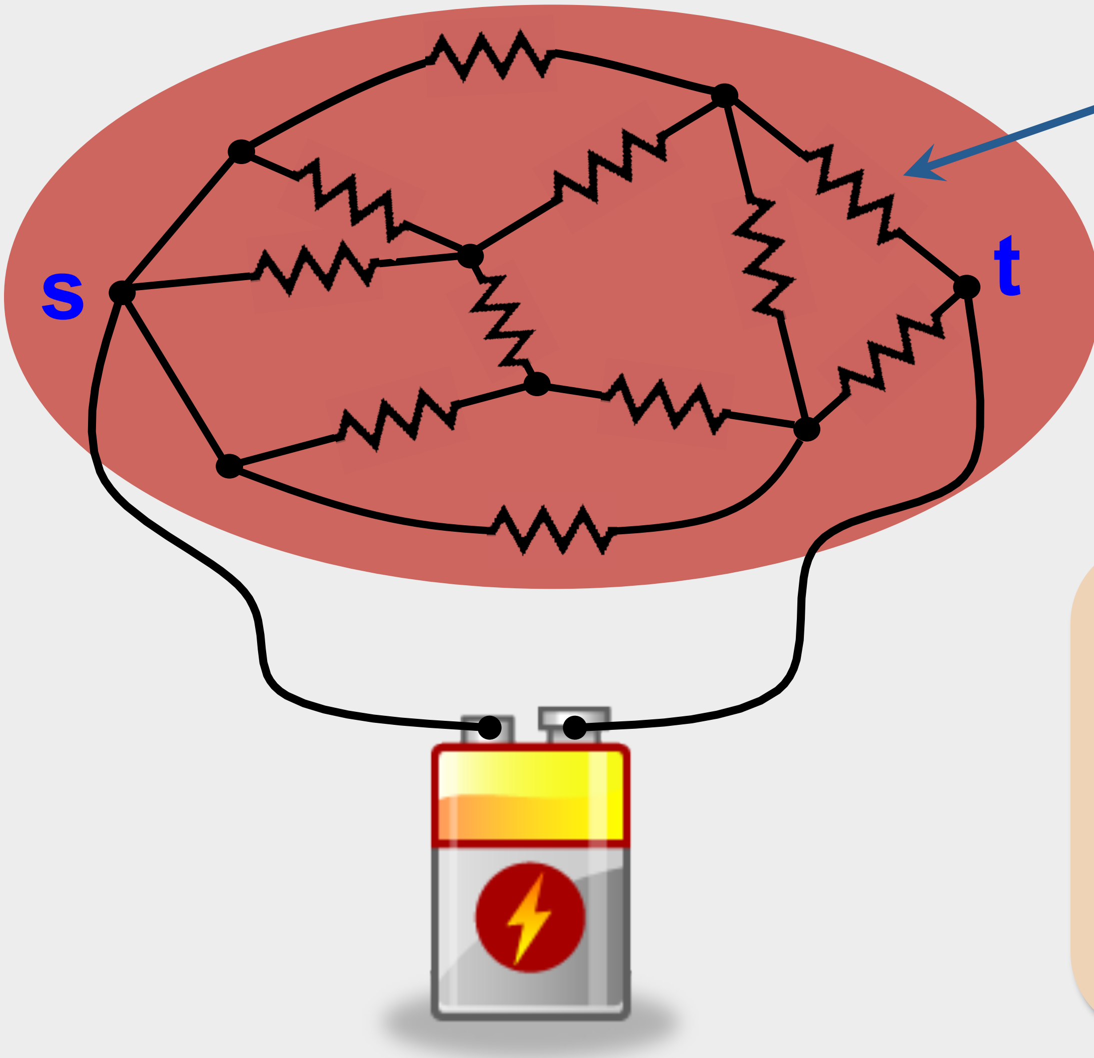
Recipe for elec. flow:

1) Treat edges as **resistors**

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances** r_e, **source s** and **sink t**

resistance r_e



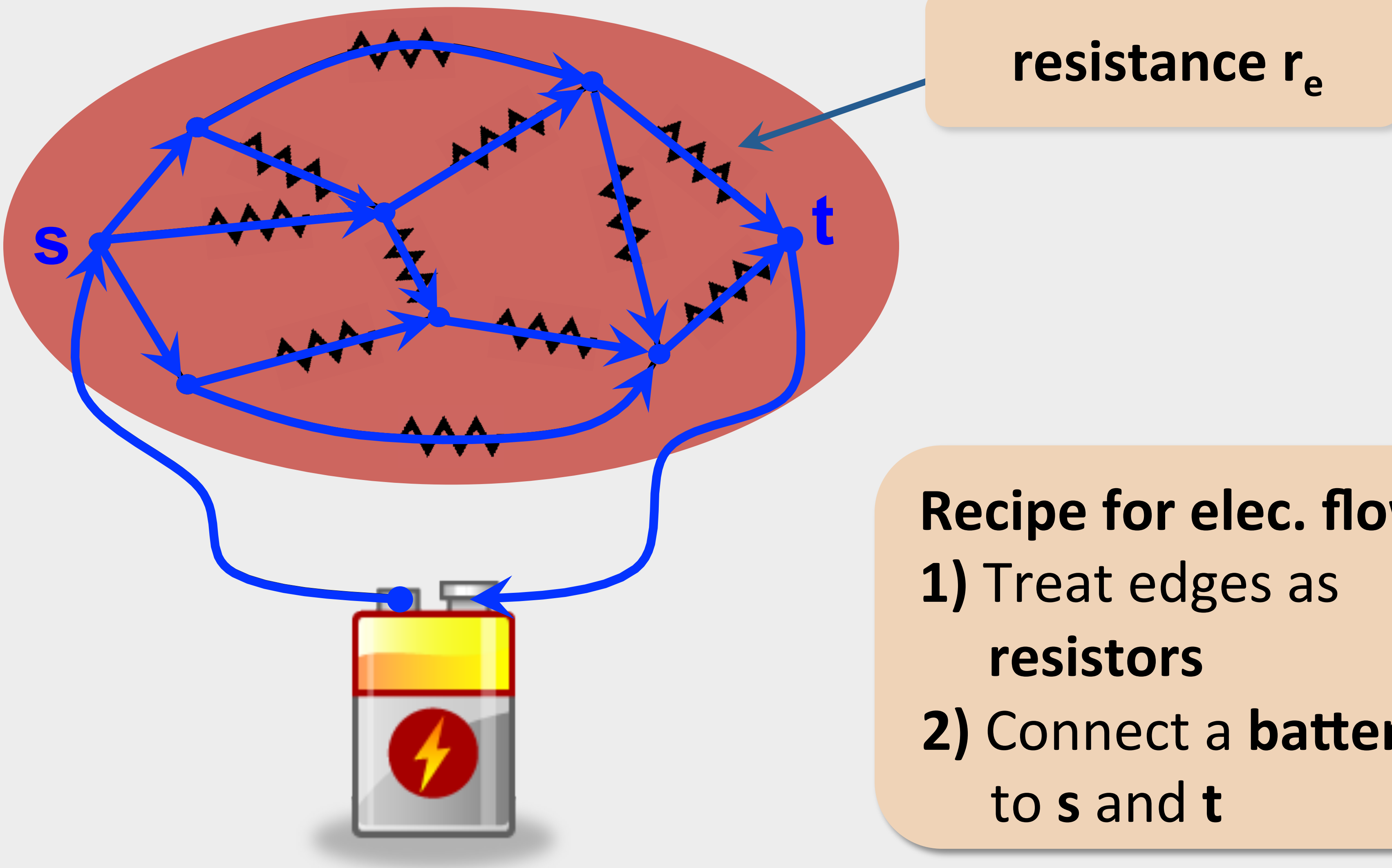
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Electrical flows (Take I)

Input: Undirected graph **G**,
resistances r_e,
source s and **sink t**



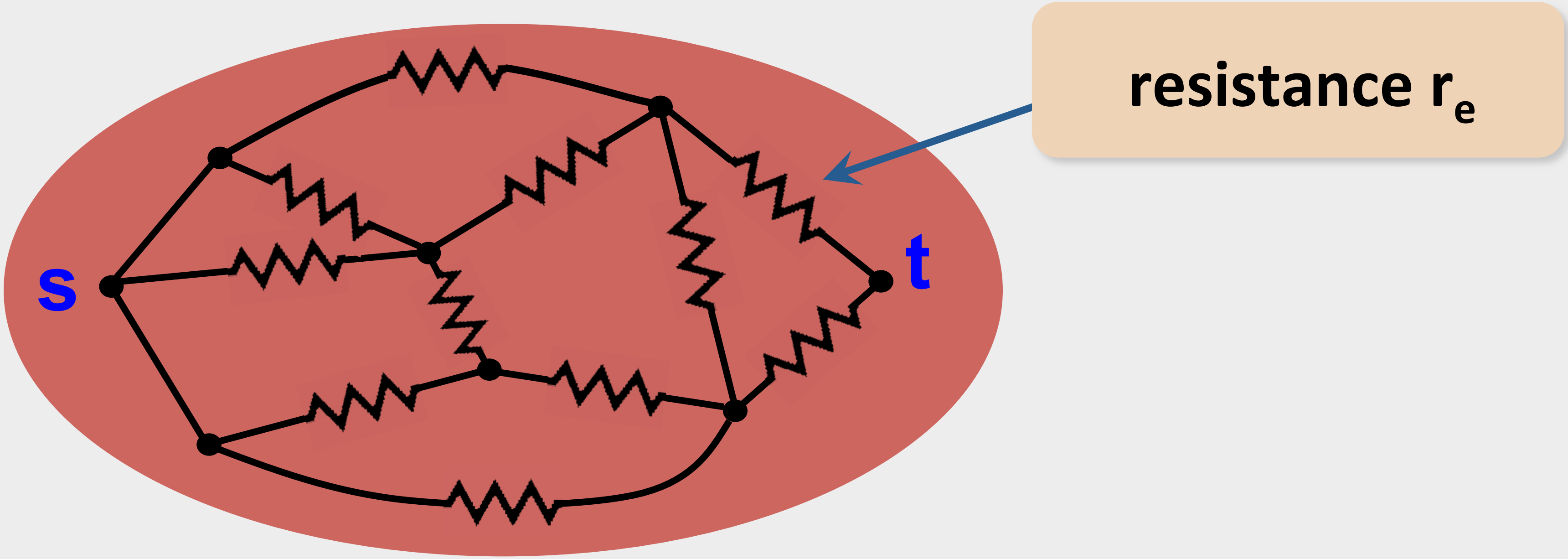
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Electrical flows (Take II)

Input: **Undirected** graph **G**, **resistances** r_e, **source s** and **sink t**

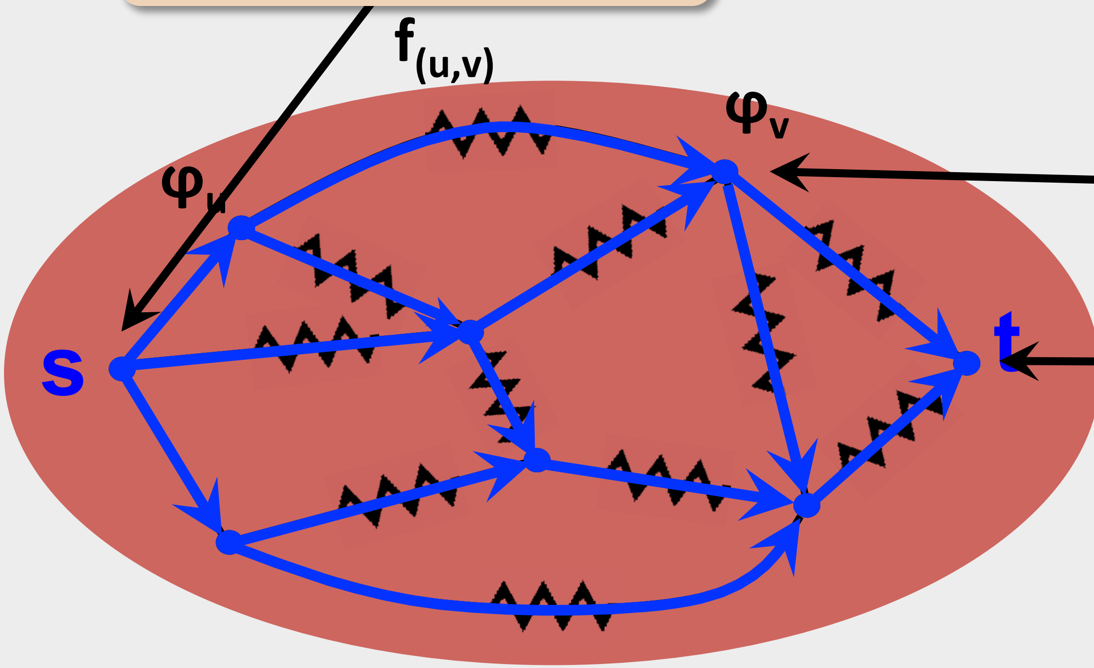


(Another) recipe for electrical flow (of value F):

Ele… ake II)

Input: Undirected graph **G**,
resistances $\mathbf{r_e}$,
source s and **sink t**

deficit of **F** at **s**



no leaks at all **v≠s,t**

excess of **F** at **t**

(Another) recipe for electrical flow (of value F):

Find **vertex potentials** $\boldsymbol{\varphi_v}$ such that setting, for all **(u,v)**

$$f_{(u,v)} \leftarrow (\varphi_v - \varphi_u)/r_{(u,v)} \qquad \textbf{(Ohm's law)}$$

gives a **valid s-t flow** of **value F**

Electrical flows (Take III)

Input: **Undirected** graph **G**, **resistances** $\mathbf{r_e}$, **source s** and **sink t**

Principle of least energy

Electrical flow of value F:
The unique minimizer of the **energy**

$$\mathbf{E}(f) = \Sigma_e\, r_e\, f(e)^2$$

among all **s-t** flows **f** of value **F**

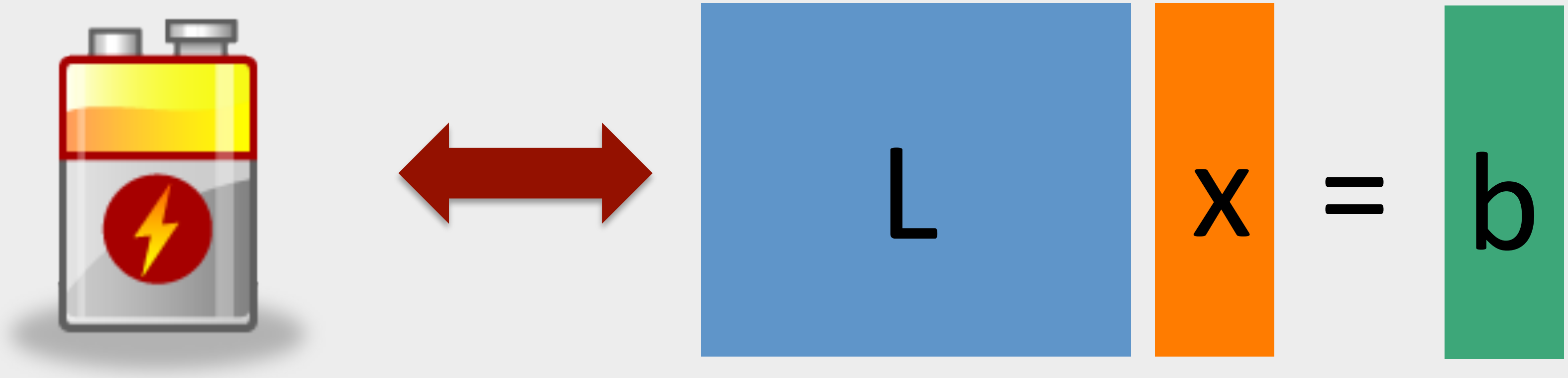
Electrical flows = ℓ_2-minimization

How to compute an electrical flow?

Solve a linear system!

How to compute an electrical flow?

Solve a **Laplacian** system!



Result: Electrical flow is a **nearly-linear time** primitive
[ST ’04, KMP ’10, KMP ’11, KOSZ ’13, LS ’13, CKPPR ‘14]

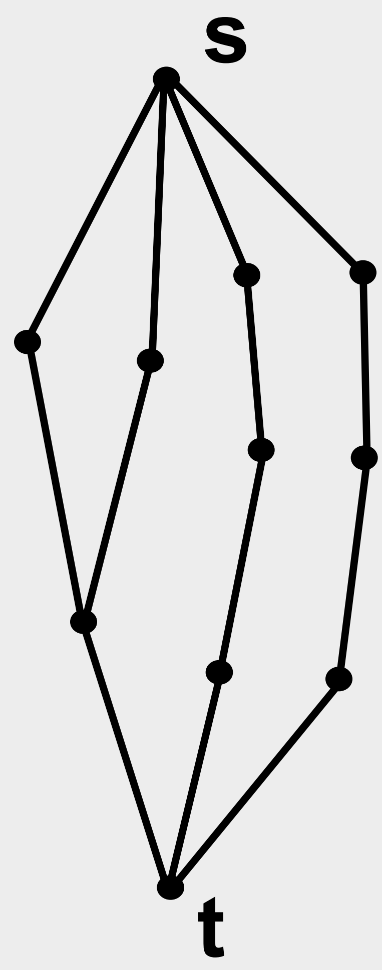
How to employ it?

From electrical flows to undirected max flow

Approx. undirected max flow [Christiano Kelner M. Spielman Teng ’11] via electrical flows

Assume: $\mathbf{F}^*$ known (via binary search)

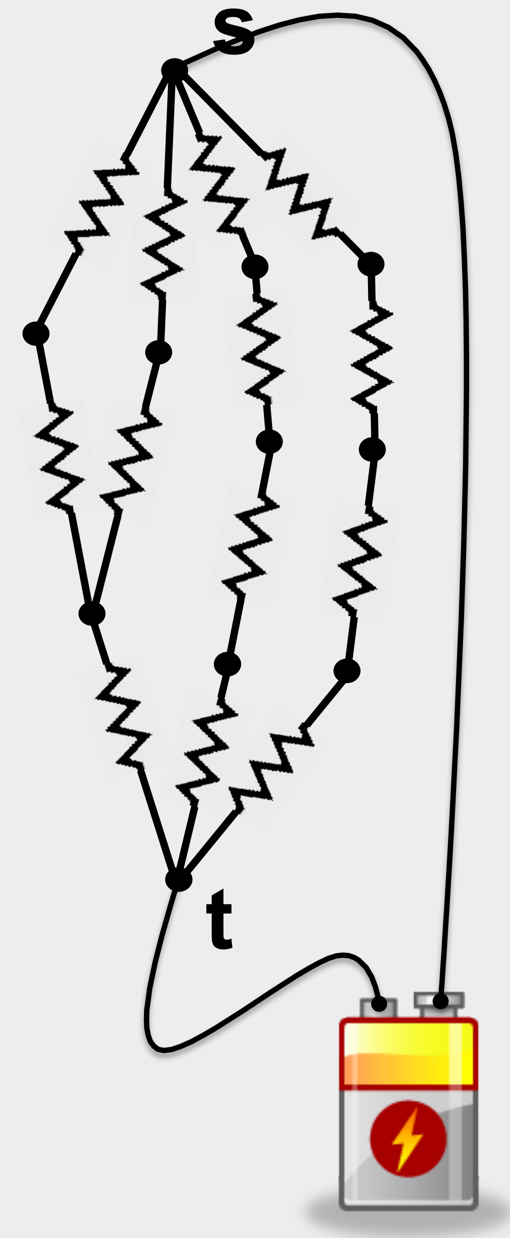
→ Treat edges as resistors of resistance **1**



Approx. undirected max flow [Christiano Kelner M. Spielman Teng '11] via electrical flows

Assume: $\mathbf{F}^*$ known (via binary search)

→ Treat edges as resistors of resistance **1**
→ Compute electrical flow of value $\mathbf{F}^*$



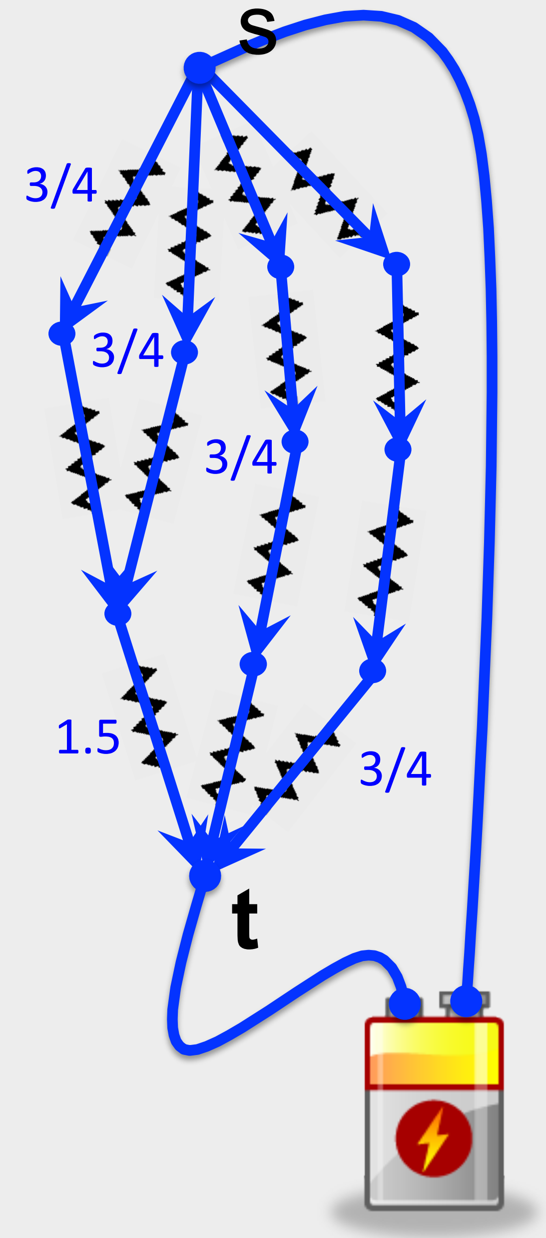
Approx. undirected max flow [Christiano Kelner M. Spielman Teng '11] via electrical flows

Assume: F^* known (via binary search)

→ Treat edges as resistors of resistance **1**

→ Compute electrical flow of value $\mathbf{F^*}$

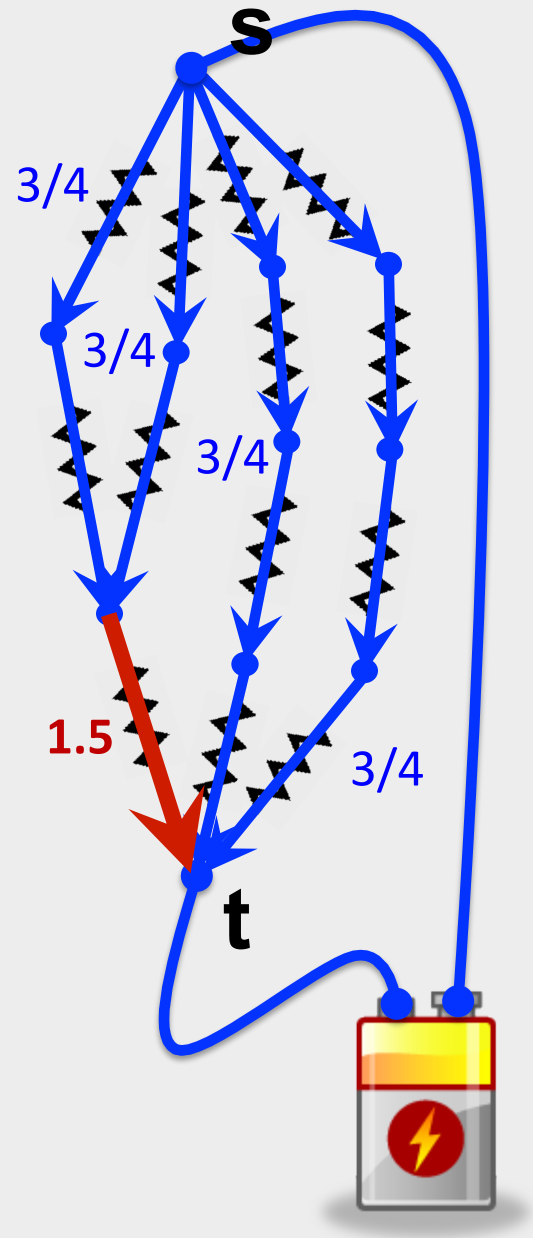
(This flow has **no leaks**, but **can overflow** some edges)



Approx. undirected max flow [Christiano Kelner M. Spielman Teng '11] via electrical flows

Assume: $\mathbf{F}^*$ known (via binary search)

→ Treat edges as resistors of resistance **1**
→ Compute electrical flow of value $\mathbf{F}^*$
(This flow has **no leaks**, but **can overflow** some edges)
→ To fix that: **Increase resistances** on the overflowing edges
Repeat (**hope:** it doesn't happen too often)



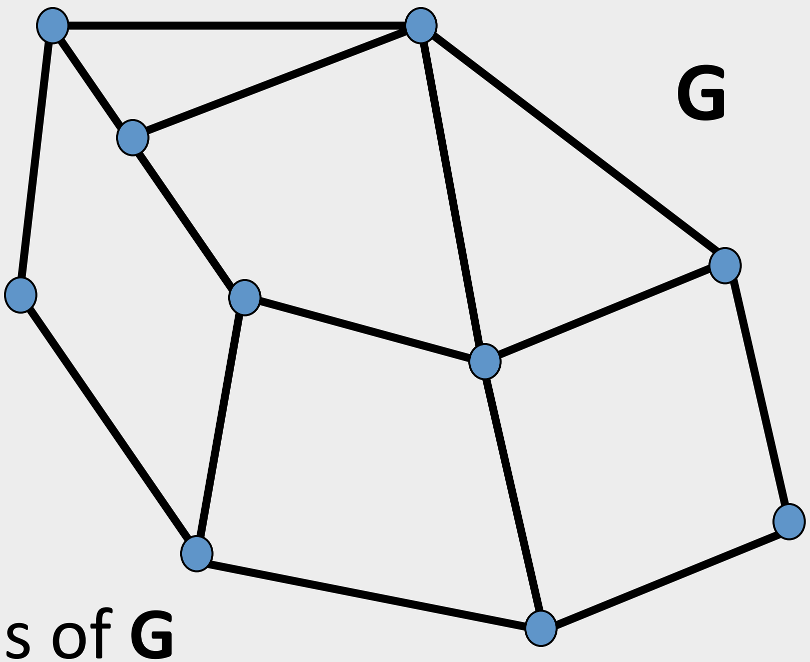
Surprisingly: This approach can be made work!

Tomorrow: Will discuss how to fill in the blanks

Generating Random Spanning Trees

Random Spanning Trees

Goal: Output an uniformly random spanning tree

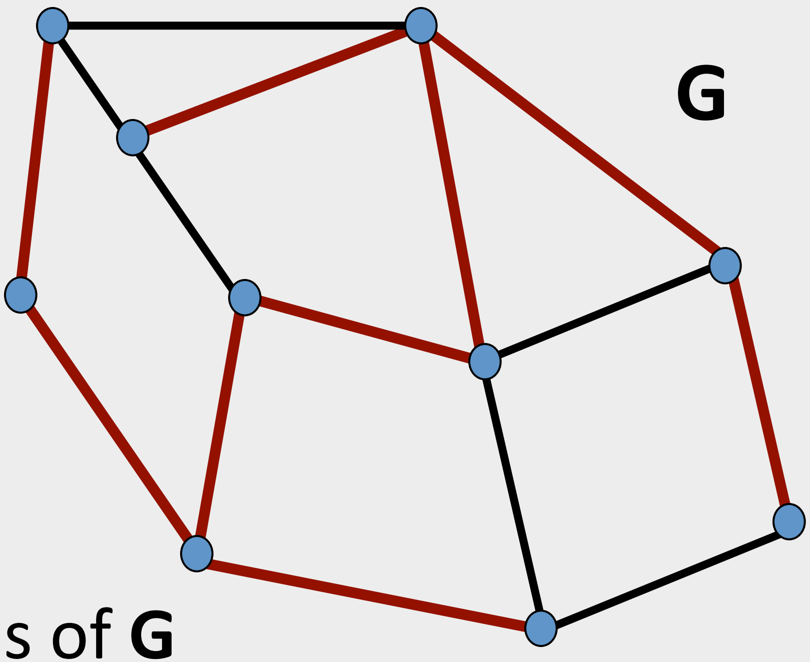


More precisely:

$\mathcal{T}$**(G) =** set of all spanning trees of **G**

Random Spanning Trees

Goal: Output an uniformly random spanning tree

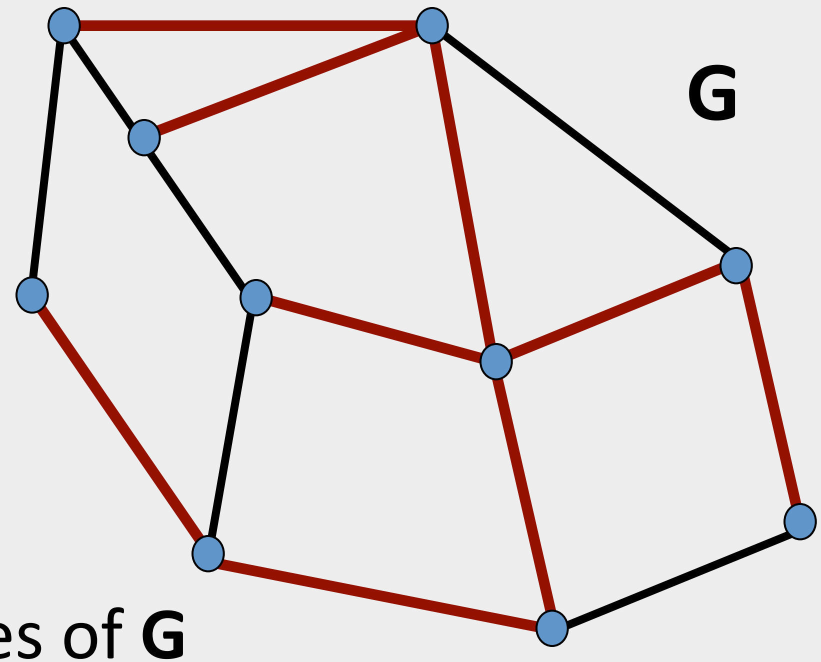


More precisely:

$\mathcal{T}$**(G) =** set of all spanning trees of **G**

Random Spanning Trees

Goal: Output an uniformly random spanning tree



More precisely:

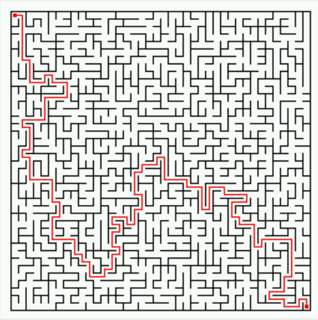
$\mathcal{T}$**(G) =** set of all spanning trees of **G**

Task: Output a tree **T** with prob. $|\mathcal{T}(G)|^{-1}$

Note: $|\mathcal{T}(G)|$ can be as large as n^{n-2}

Why Random Spanning Trees?

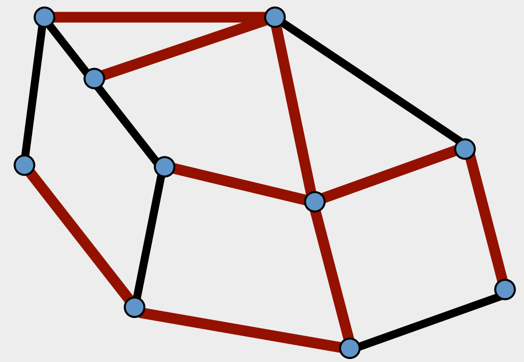
- Fundamental probabilistic object in graph theory (study dates back to 1800s **[K 1847]**)
- Applications in computer networks, statistical physics, computational biology
- **Deep connections to electrical flows and graph structure**:
  → Efficient sparsifiers **[GRV '09]**
  → Thin trees/ Approx. of ATSP **[AGM.OS '10]**
- Recreation! (Generation of random maze puzzles)



How to Generate a Random Spanning Tree?

Matrix Tree theorem **[Kirchoff 1847]**

Pr[e in a rand. tree**]** = **Reff(e)**



Resulting algorithm:

Effective resistance of **e**

→ Order edges **e_1, e_2,…,** g empty

→ For each **e_i**:

- Compute **Reff(e_i)** and add **e_i** to **T** with that probability
- Update **G** by **contracting e_i** if **e** in **T** and **removing** it o.w.

→ Output **T**

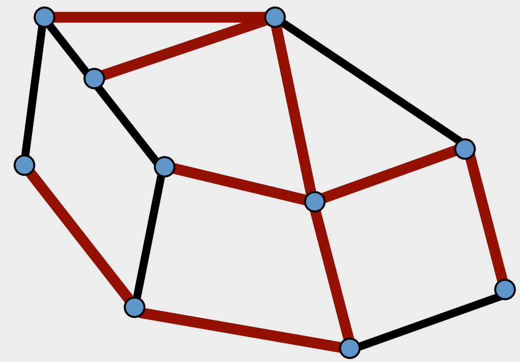
Why does it work?

Conditioning on our choice

How to Generate a Random Spanning Tree?

Matrix Tree theorem [Kirchoff 1847]

Pr[e in a rand. tree**]** = **Reff(e)**



Resulting algorithm:

→ Order edges $\mathbf{e_1, e_2, \ldots, e_m}$ arbitrarily and start with **T** being empty
→ For each $\mathbf{e_i}$:
- Compute **Reff($\mathbf{e_i}$)** and add $\mathbf{e_i}$ to **T** with that probability
- Update **G** by **contracting** $\mathbf{e_i}$ if **e** in **T** and **removing** it o.w.

→ Output **T**

Running time?

Bottleneck: Computing **Reff($\mathbf{e_i}$)**

But: $\mathbf{Reff(e) = \chi_e^T L^{-1} \chi_e}$

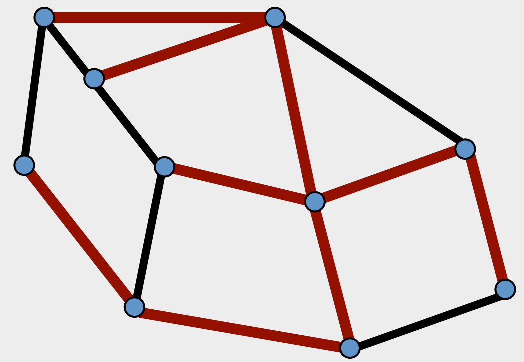
→ Need to solve a Laplacian system (~~exactly!~~)

Resulting runtime: min($\mathbf{m\, n^{\omega}}$, $\mathbf{\tilde{O}(m^2)}$)

How to Generate a Random Spanning Tree?

Matrix Tree theorem **[Kirchoff 1847]**

Pr[e in a rand. tree**]** = **Reff(e)**



Resulting algorithm:

→ Order edges $\mathbf{e_1, e_2, \ldots, e_m}$ arbitrarily and start with **T** being empty
→ For each $\mathbf{e_i}$:

- Compute **Reff($\mathbf{e_i}$)** and add $\mathbf{e_i}$ to **T** with that probability
- Update **G** by **contracting** $\mathbf{e_i}$ if **e** in **T** and **removing** it o.w.

→ Output **T**

Running time?

But: $\mathbf{Reff(e) = \chi_e^T L^{-1} \chi_e}$

Bottleneck: Computing **Reff($\mathbf{e_i}$)**

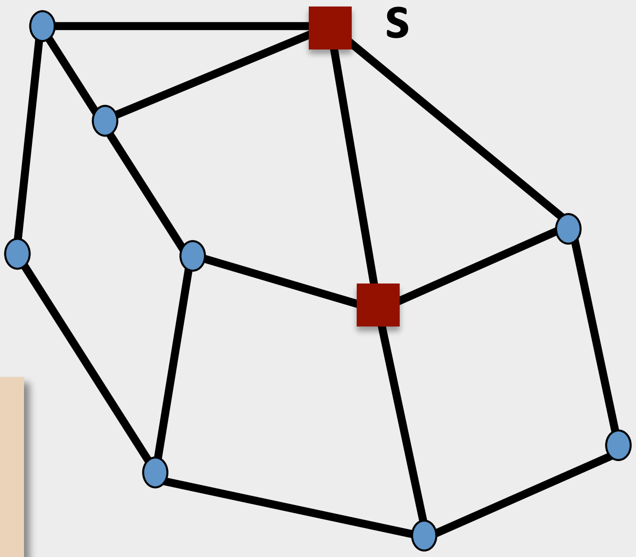
→ Need to solve a Laplacian system (~~exactly~~!)

Resulting runtime: min($\mathbf{n^\omega}$, $\mathbf{\tilde{O}(m^2)}$) **[CMN '96]**

Can we do better?

[Broder '89, Aldous '90]: Generate random spanning tree using random walks

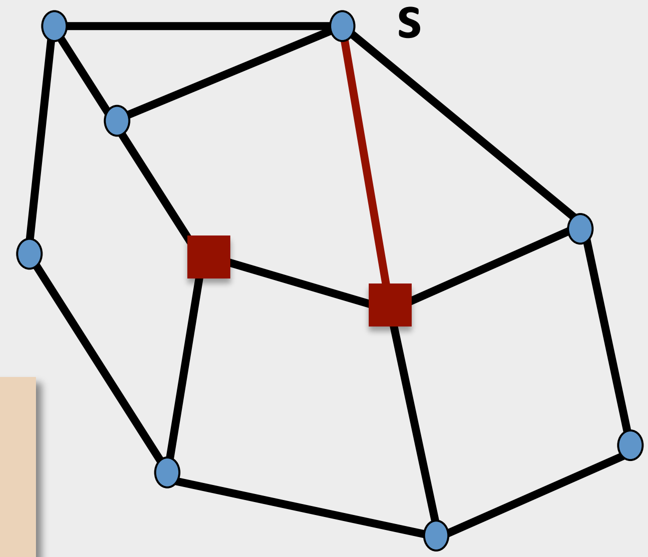
→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited
→ Output **T**



Can we do better?

[Broder '89, Aldous '90]: Generate random spanning tree using random walks

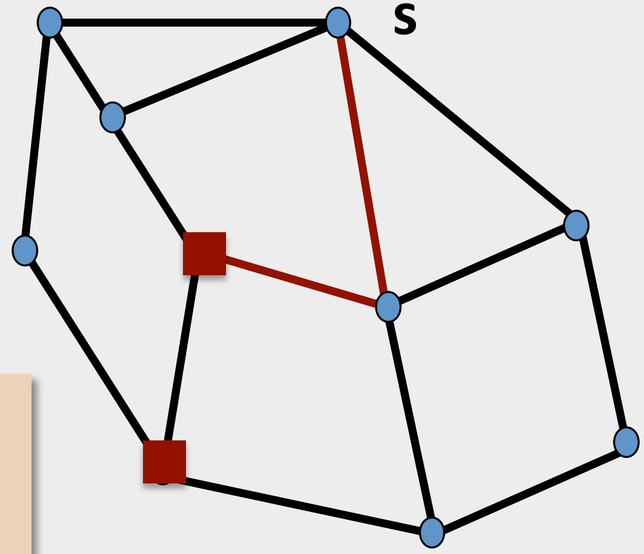
→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited
→ Output **T**



Can we do better?

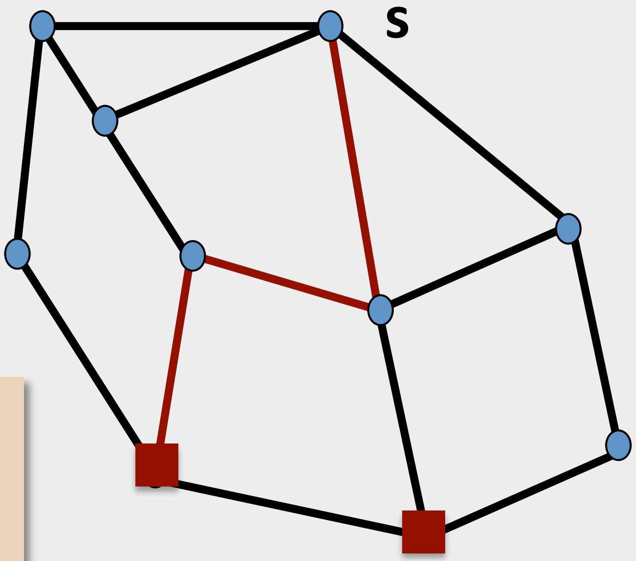
[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**, add to **T** the edge through which we visited
→ Output **T**



Can we do better?

[Broder ’89, Aldous ’90]: Generate random spanning tree using random walks

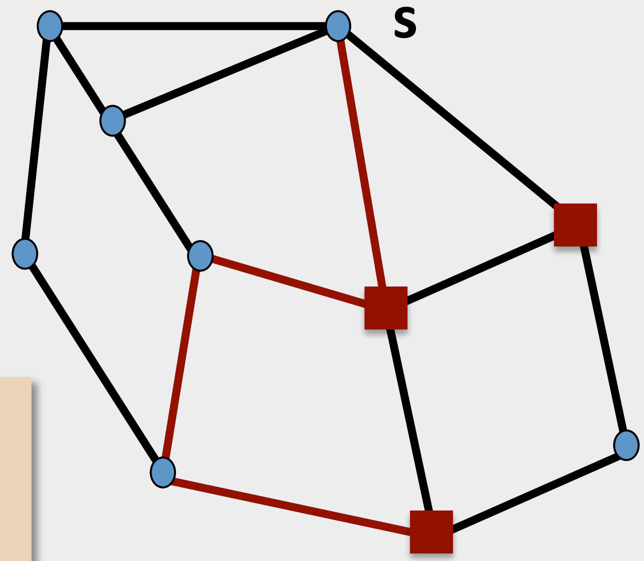


→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited
→ Output **T**

Can we do better?

[Broder '89, Aldous '90]: Generate random spanning tree using random walks

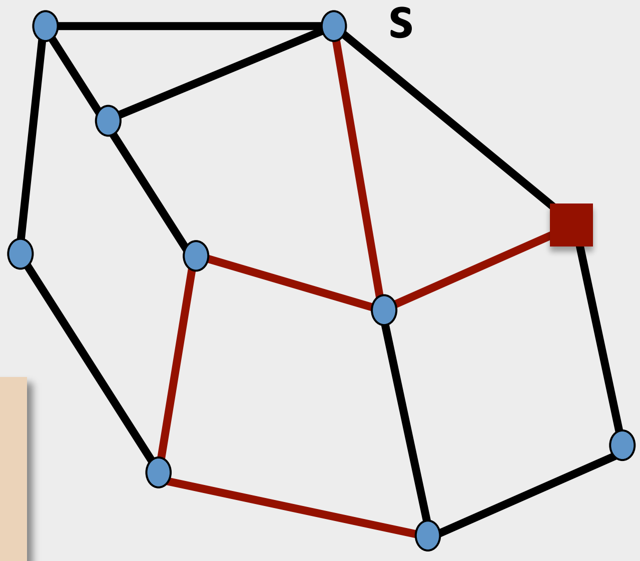
→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited
→ Output **T**



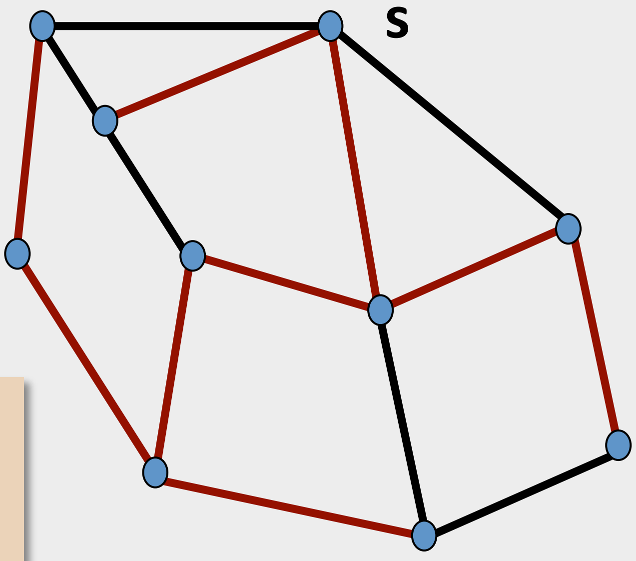
Can we do better?

[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited
→ Output **T**



Can we do better?



[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ Start a random walk at some vertex **s**
→ Whenever visiting a new vertex **v**, add to **T** the edge through which we visited
→ Output **T**

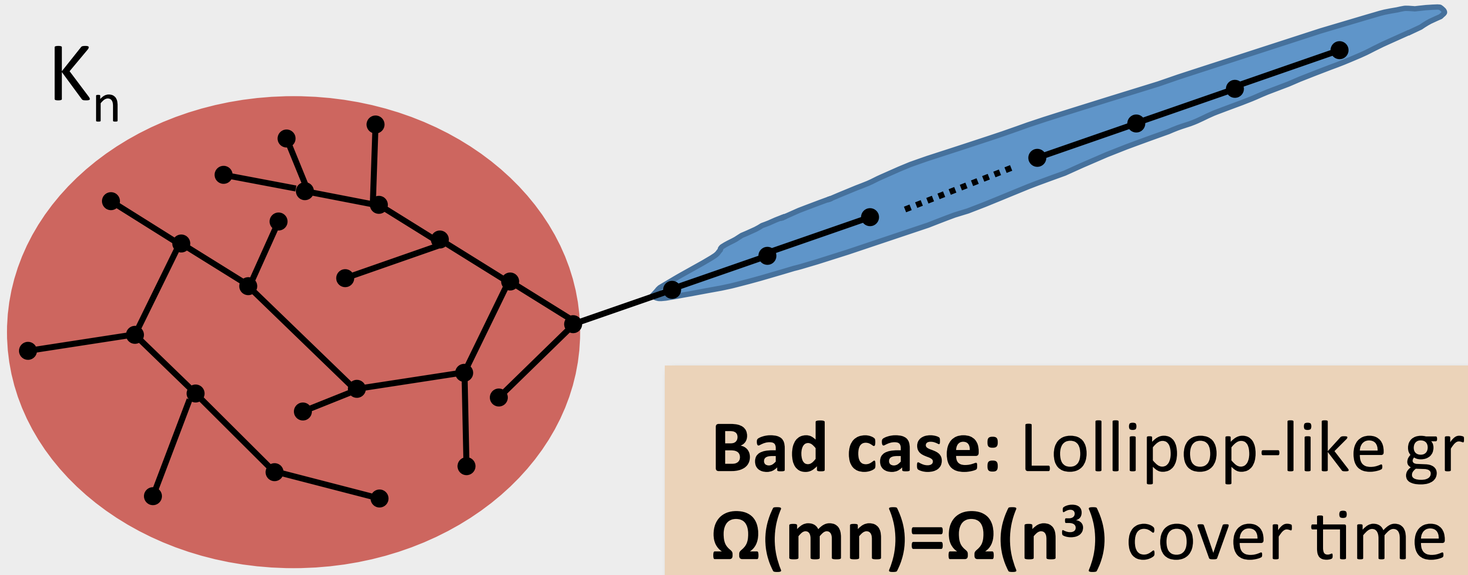
Why does it work? Magic!

Running time?

O(cover time) = O(mn)

[W '96]: Can get **O(mean hitting time)** but still **O(mn)** in the worst case

Can we improve upon that?



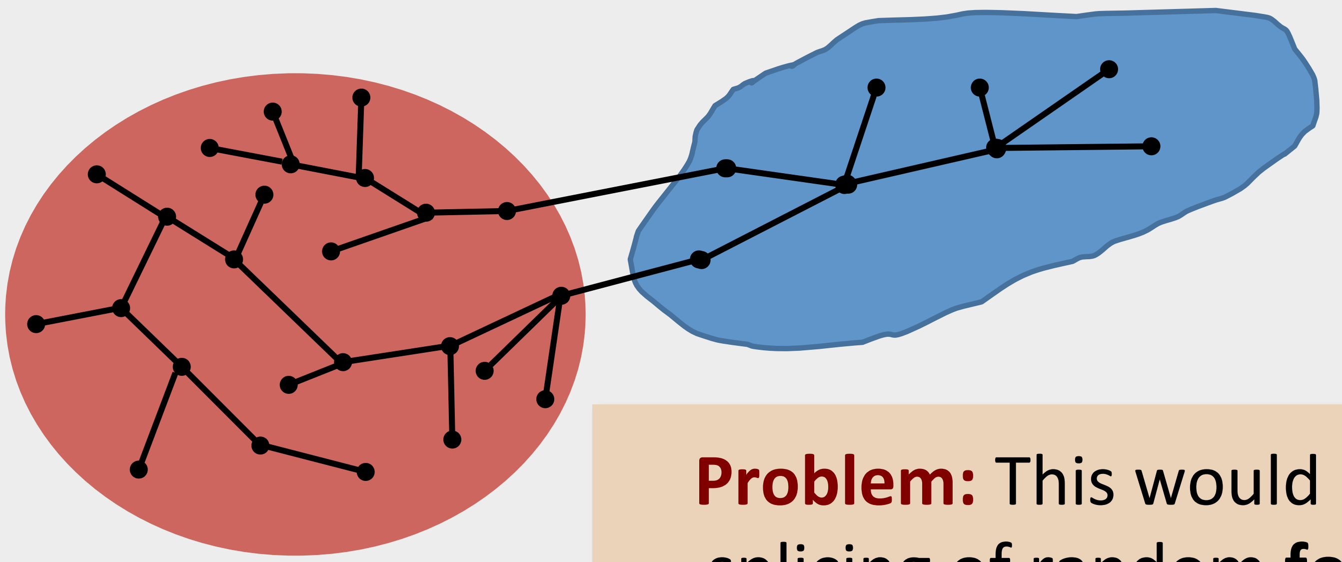
Bad case: Lollipop-like graph
$\Omega(mn)=\Omega(n^3)$ cover time

What happens: The walk resides mainly in K_n - the path-like part is covered only after a lot of attempts

Observe: We know how the tree looks like in K_n very early on

Idea: Cut the graph into pieces with good cover time and find trees in each piece separately

Can we improve upon that?



Problem: This would require splicing of random **forests**

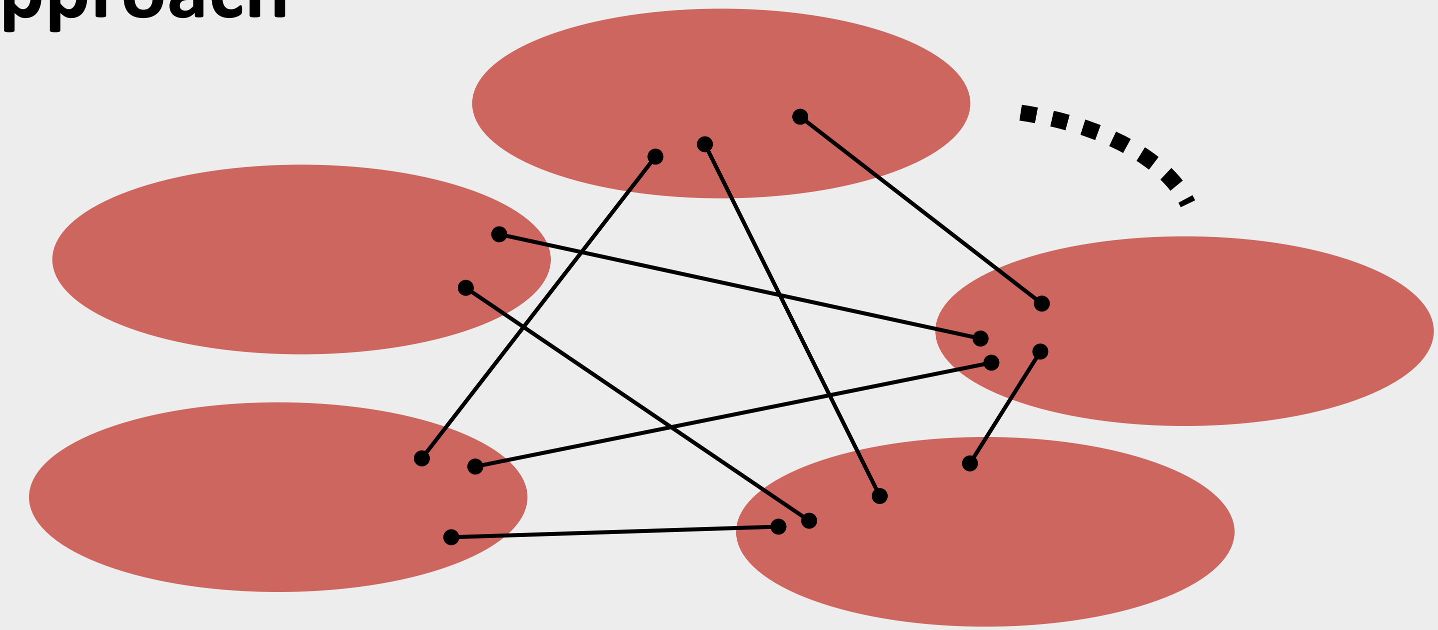
What happens: The walk resides mainly in K_n - the path-like part is covered only after a lot of attempts

Observe: We know how the tree looks like in K_n very early on

Idea: Cut the graph into pieces with good cover time and find trees in each piece separately

Different Approach

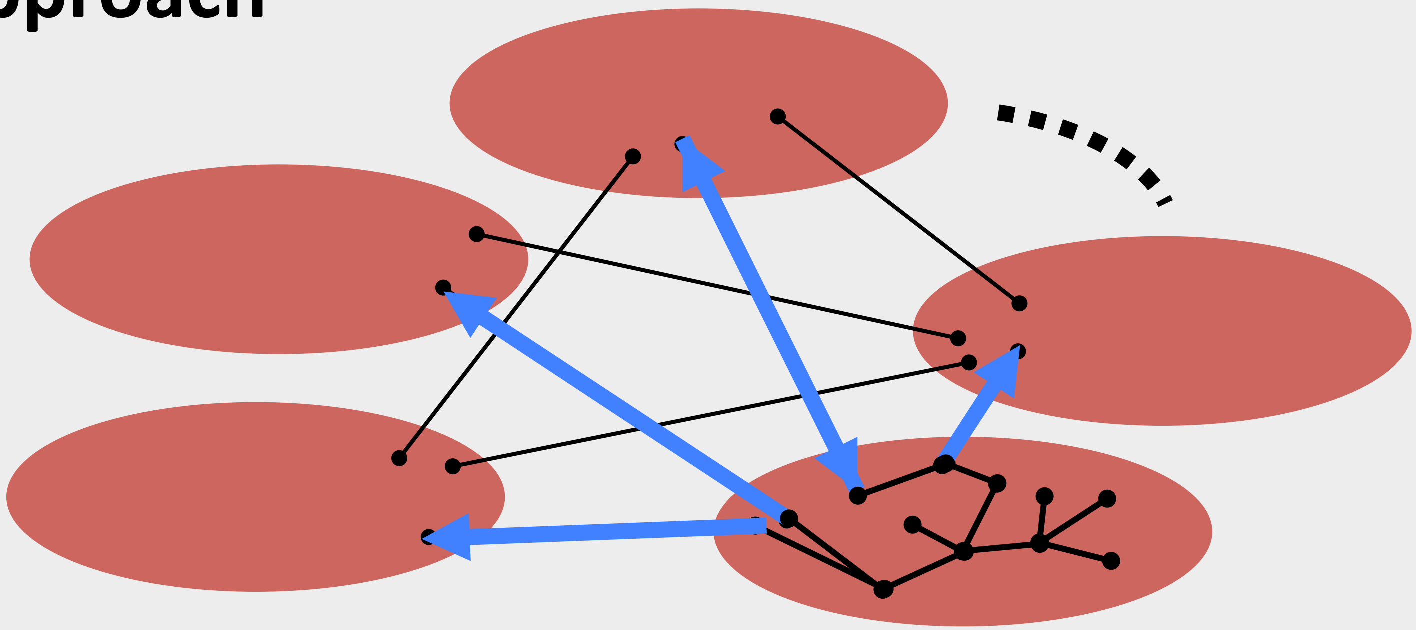
[Kelner M. '09]



Decompose the graph into pieces with:
→ Low diameter each
→ Small "interface"

Different Approach

[Kelner M. '09]



Decompose the graph into pieces with:

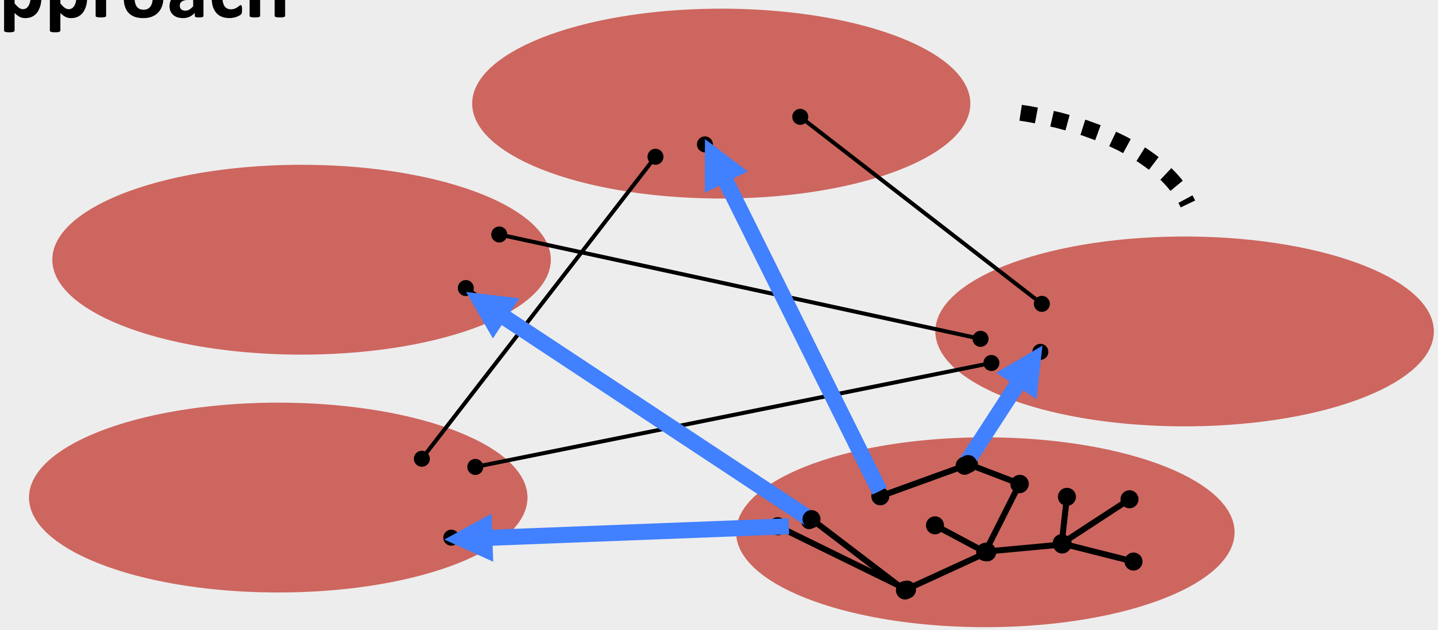
→ Low diameter each

→ Small "interface"

Modification: When simulating the random walk, **shortcut** revisits to pieces that were already explored in full

Note: We still retain enough information to output the final tree

Different Approach
[Kelner M. '09]



Decompose the graph into pieces with:

→ Low diameter each = we cover each piece relatively quickly

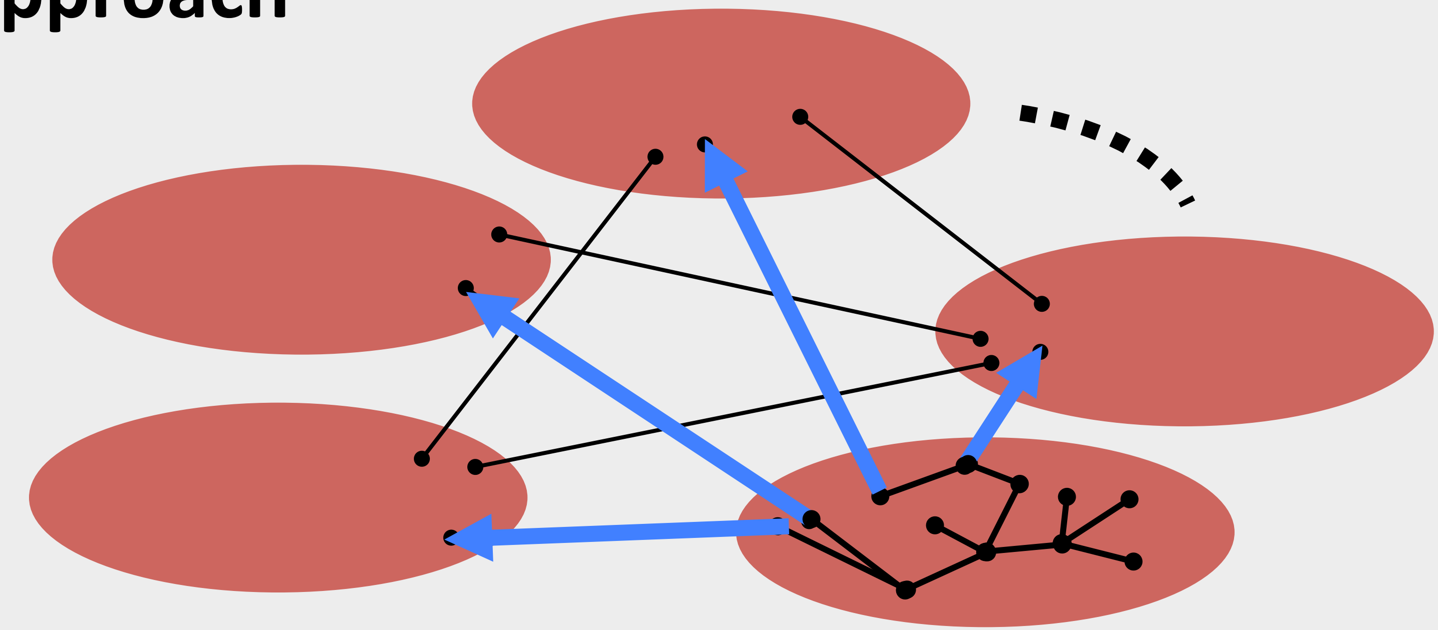
→ Small "interface"

Modification: When simulating the random walk, **shortcut** revisits to pieces that were already explored in full

Note: We still retain enough information to output the final tree

Different Approach

[Kelner M. '09]



Decompose the graph into pieces with:

→ Low diameter each = we cover each piece relatively quickly

→ Small "interface" = we do not walk too much over that interface

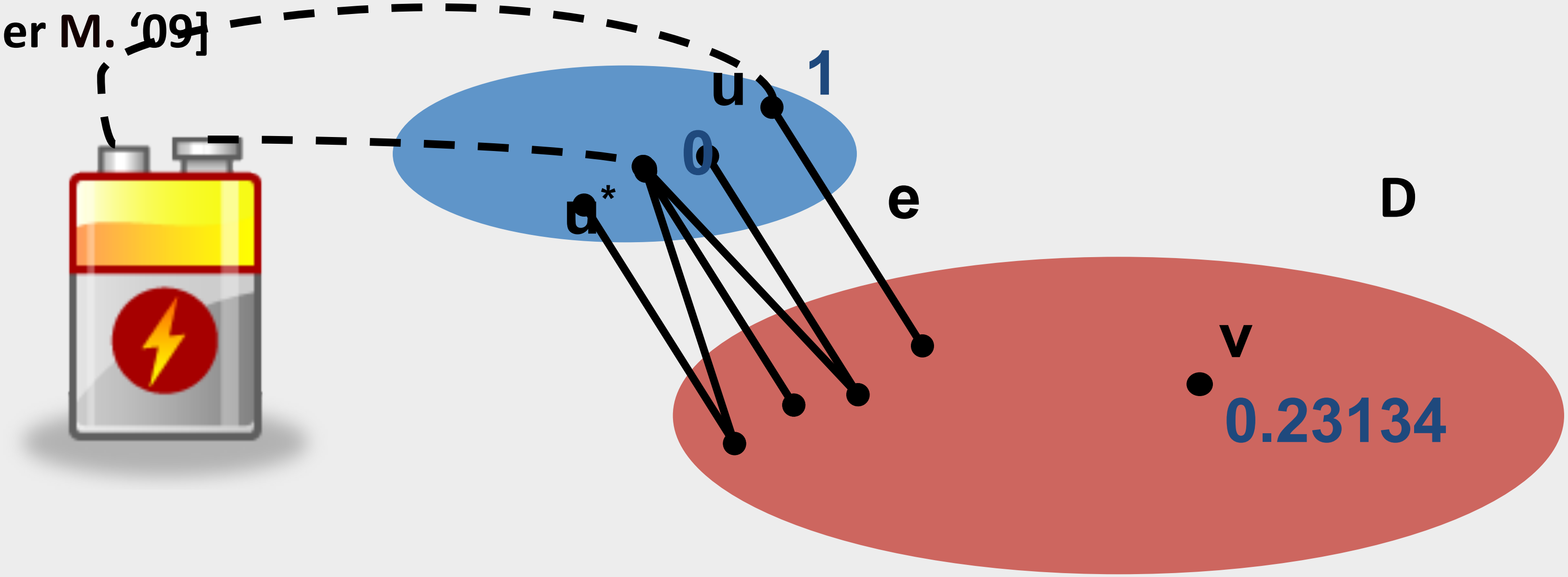
Modification: When simulating the random walk, **shortcut** revisits to pieces that were already explored in full

Note: We still retain enough information to output the final tree

Missing element: How to compute the shortcutting jumps?

Different Approach

[Kelner M. '09]



Need: P_D(e,v) = prob. we exit **D** via edge **e** after entering through **v**

Electrical flows/Laplacian solvers can compute that!

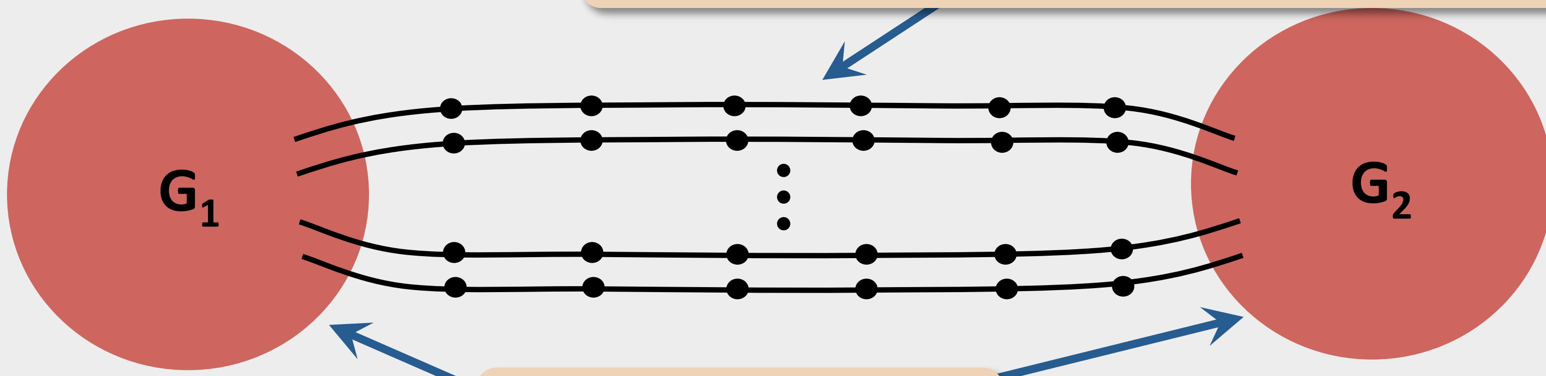
[Propp '09]: Computing good approx. to voltages suffices

Putting it all together: Generation of a random spanning tree in **$\tilde{O}(mn^{1/2})$** time

Breaking the $\Omega(n^{3/2})$ barrier

[M. Straszak Tarnawski '14]

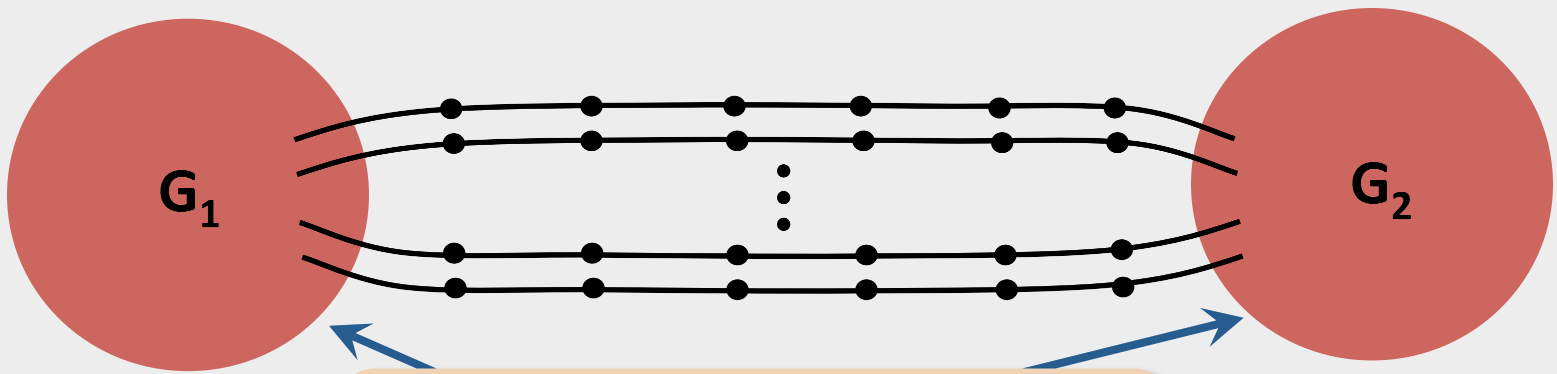
$\approx$**n**^{½} paths with $\approx$**n**^{½} vertices each



expanders

Breaking the $\Omega(n^{3/2})$ barrier

[M. Straszak Tarnawski '14]



Probl ... ver time

... it

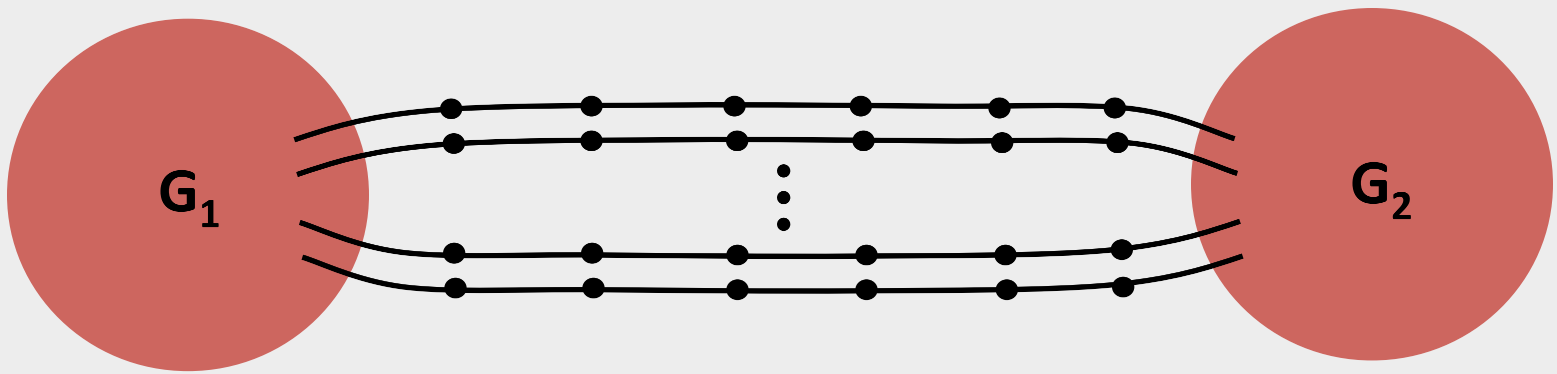
G_1 and **G_2** are close in effective resistance metric

To overcome this:

→ **Work with the "right" metric:** effective resistance metric (given by $\mathbf{L}^{-½}$) instead of the graph distance metric

Breaking the $\Omega(n^{3/2})$ barrier

[M. Straszak Tarnawski '14]



Problem: This graph has an $\mathbf{\Omega(n^{3/2})}$ cover time and there is no nice way to cut it

To overcome this:

→ **Work with the "right" metric:** effective resistance metric (given by $L^{-½}$) instead of the graph distance metric

region with small effect. resist. diameter

Result: An $\mathbf{O(n^{4/3+o(1)})}$ time sampling algorithm

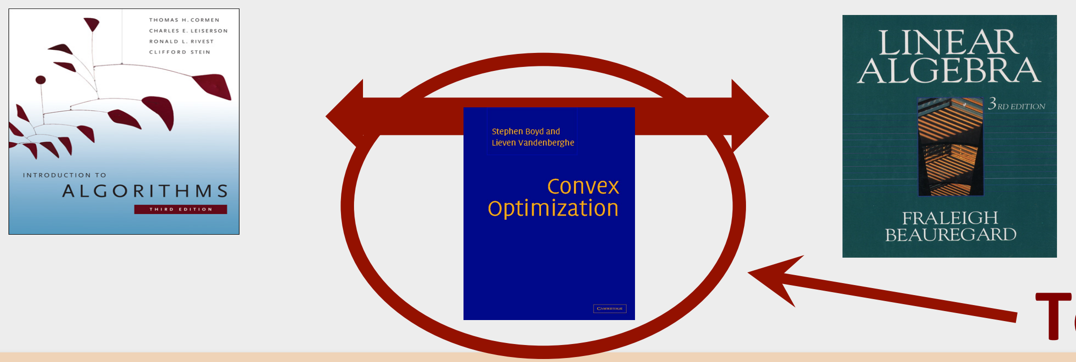
→ **Tie effect. resist. to graph cuts:** Show that any two large regions separated in effect. resist. metric have a good cut

Wrapping Up

We have seen two examples of electrical flows being a key **algorithmic** primitive

(There is more and will be even more in the future)

Merging combinatorial and continuous perspective was crucial for achieving success here



Tomorrow

Ultimate goal: Forging next generation toolkit for graph algorithms

→ Capable of making progress on some longstanding challenges
→ Compatible with "approximate but quick" regime of big graphs

Thank you