Electrical Flow Primitive and Fast Graph Algorithms

Aleksander Mądry

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

These are exciting times for fast graph algorithms

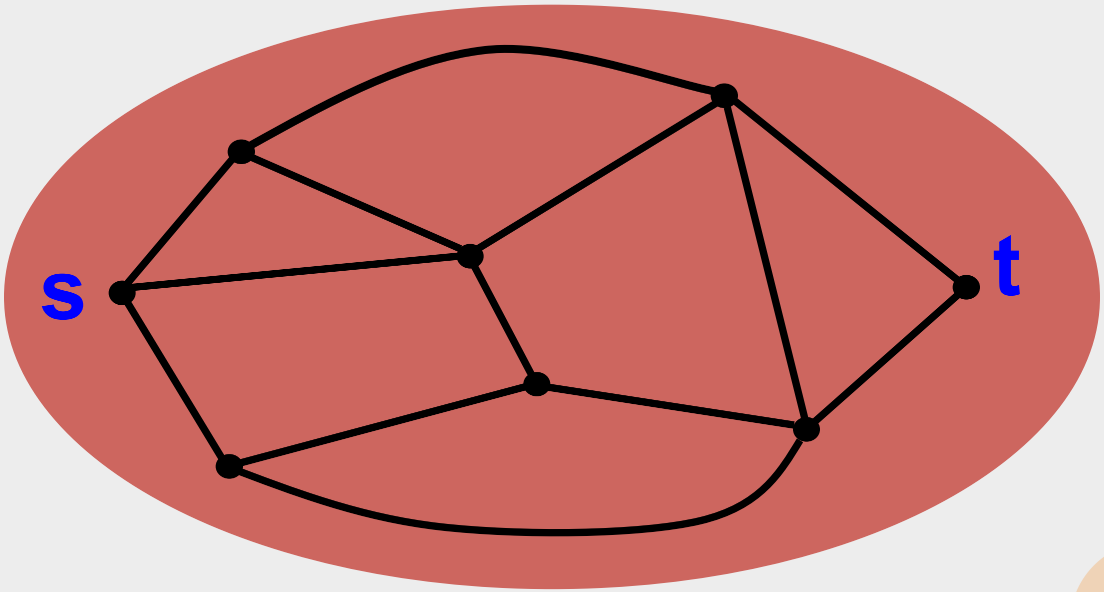
Last several years brought faster algorithms for a number of fundamental graph problems

Central notion: Electrical flows

This talk: A quick tour through these algorithmic applications and the underlying connections

Electrical flows (Take I)

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

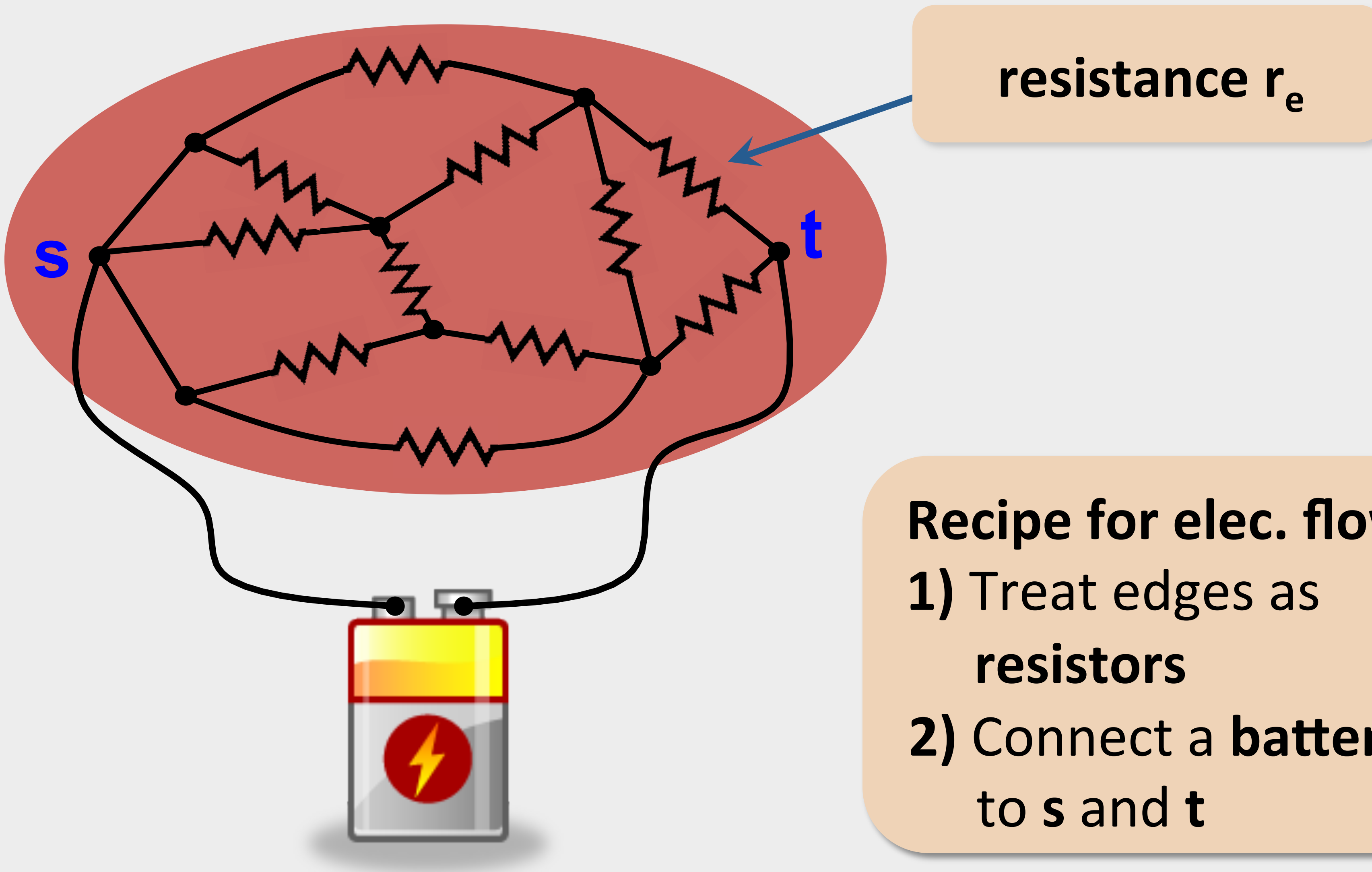


Recipe for elec. flow:

1) Treat edges as **resistors**

Electrical flows (Take I)

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



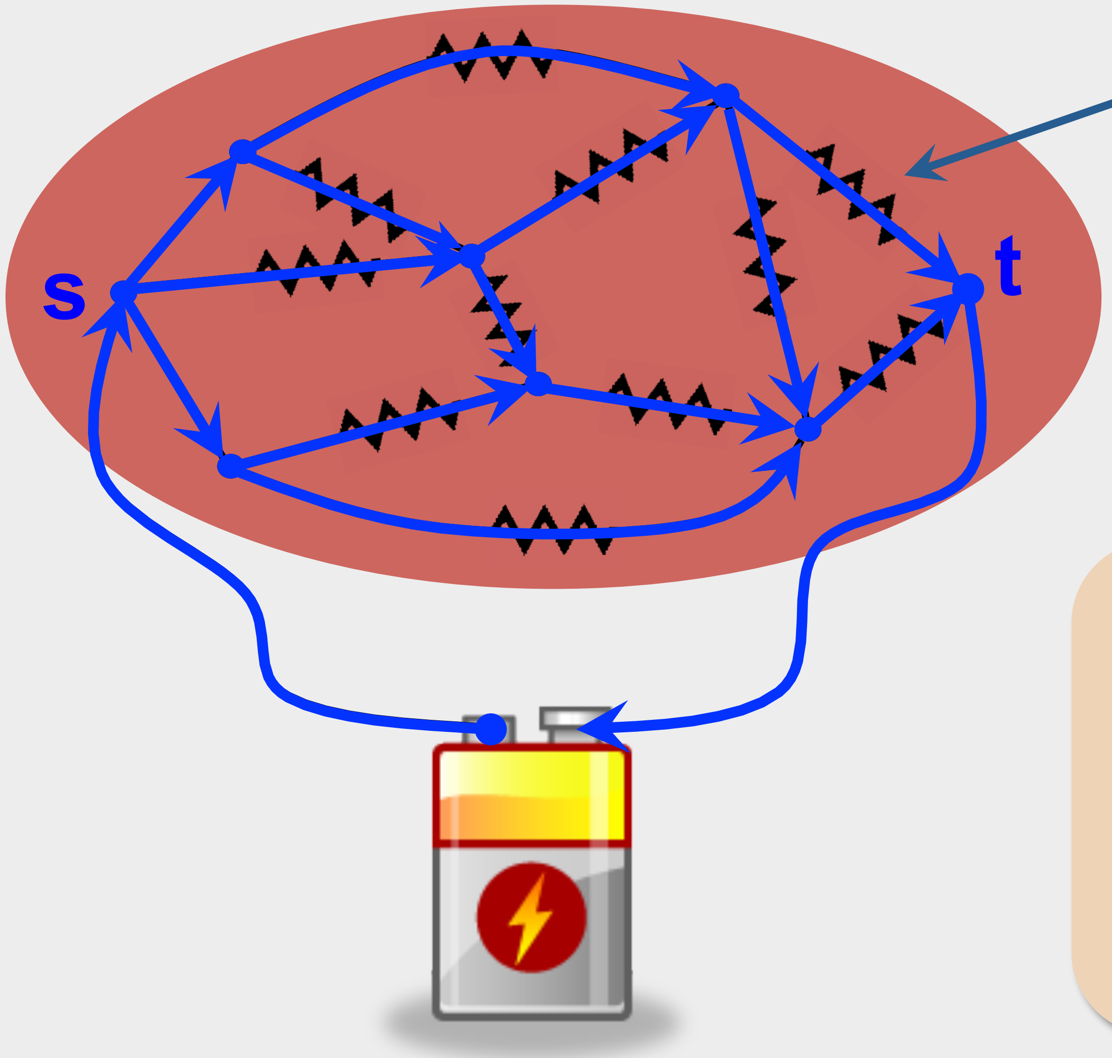
Recipe for elec. flow:

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**



resistance r_e

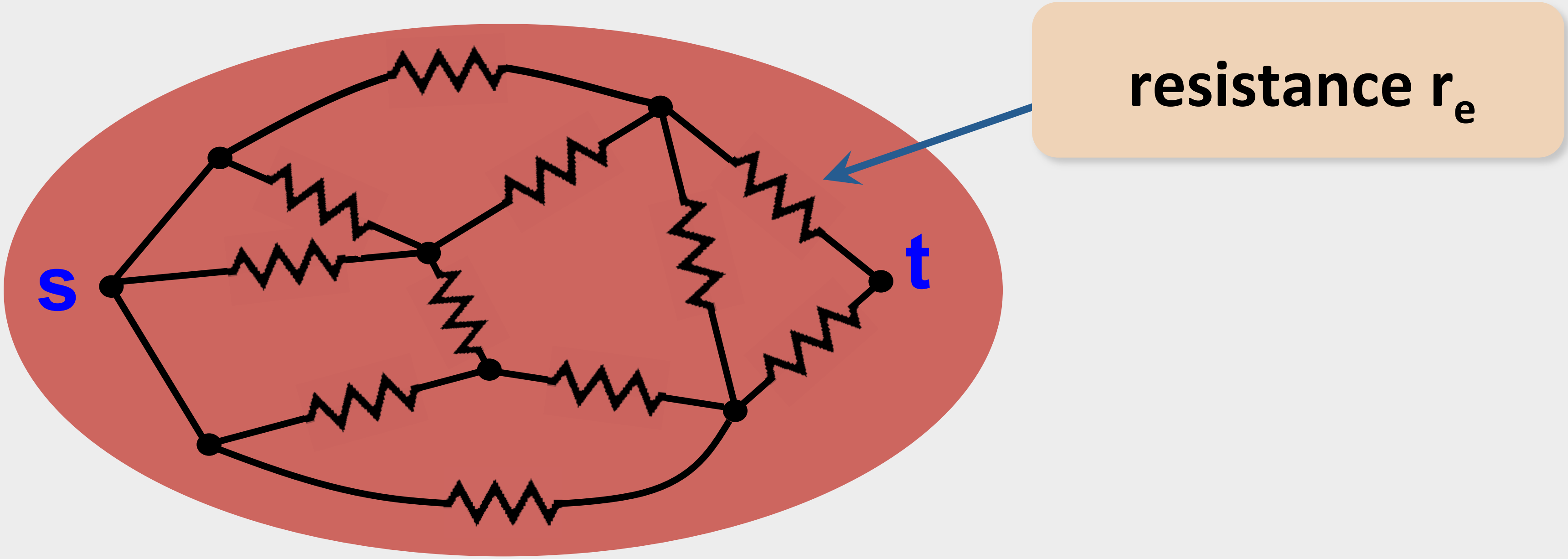
Recipe for elec. flow:

1) Treat edges as **resistors**

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

Electrical flows (Take II)

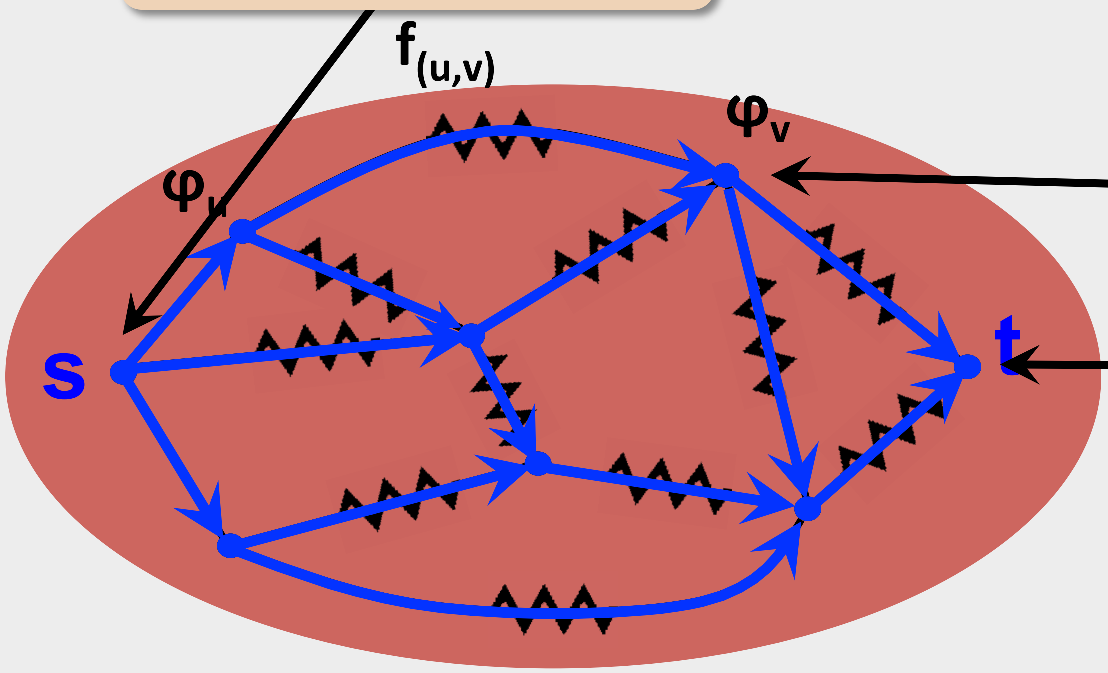
Input: **Undirected** graph **G**, **resistances** r_e, **source s** and **sink t**



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

Electrical flows (Take II)

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



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

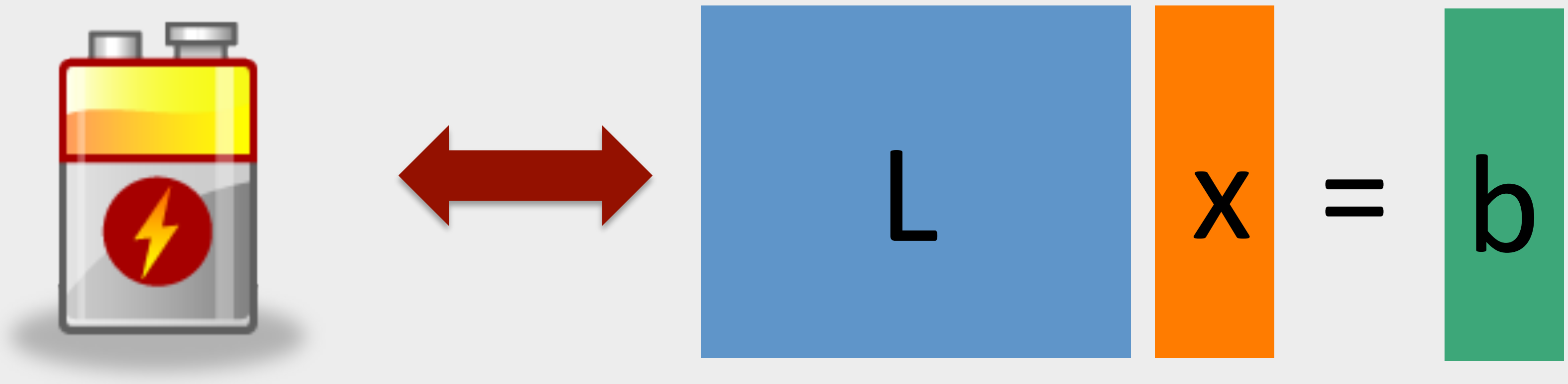
Find **vertex potentials φ_v** such that setting, for all **(u,v)**

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

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

Crucial connection: Laplacian systems

Computing electrical flows boils down to solving **Laplacian systems**



[ST ’04, KMP ’10, KMP ’11, KOSZ ’13, LS ’13, CKPPR ‘14]:
Laplacian systems can be (essentially) solved in **nearly-linear** time

Key consequence: Electrical flows become a powerful primitive

Electrical Flows and the Maximum Flow Problem

Connection I: Energy minimization

Principle of least energy:
Electrical flows = ℓ_2-minimization

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

$$E(f) = \Sigma_e r_e f(e)^2$$

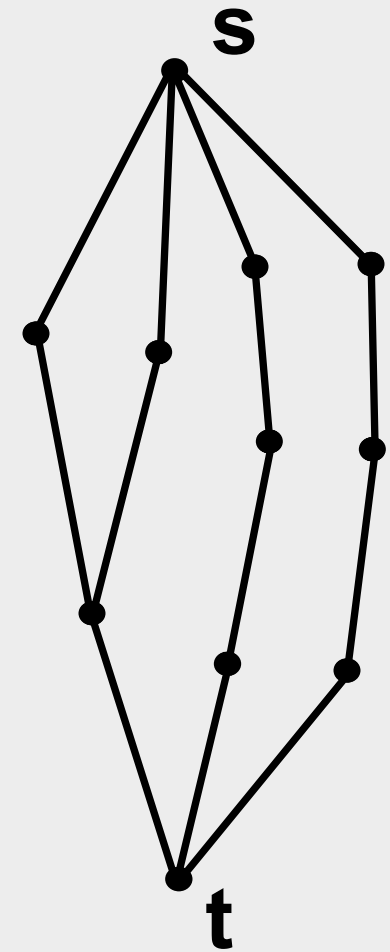
among all **s-t** flows **f** of value **F**

Application:
Turning the fast **electrical flow** computation into a fast **max flow** approx.

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

Assume: F* known (via binary search)

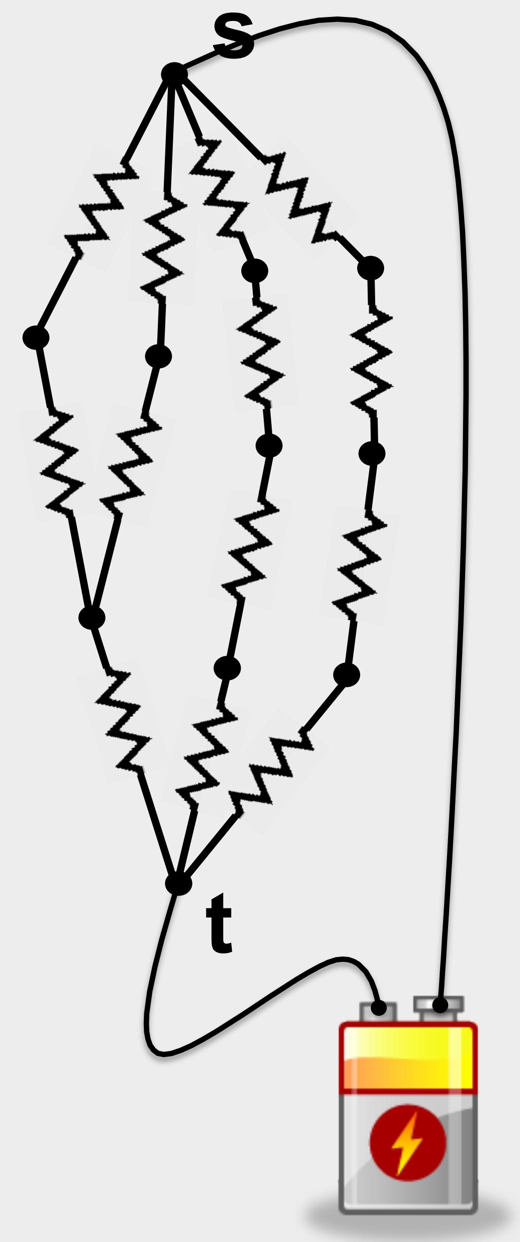
→ Treat edges as resistors of resistance **1**



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

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

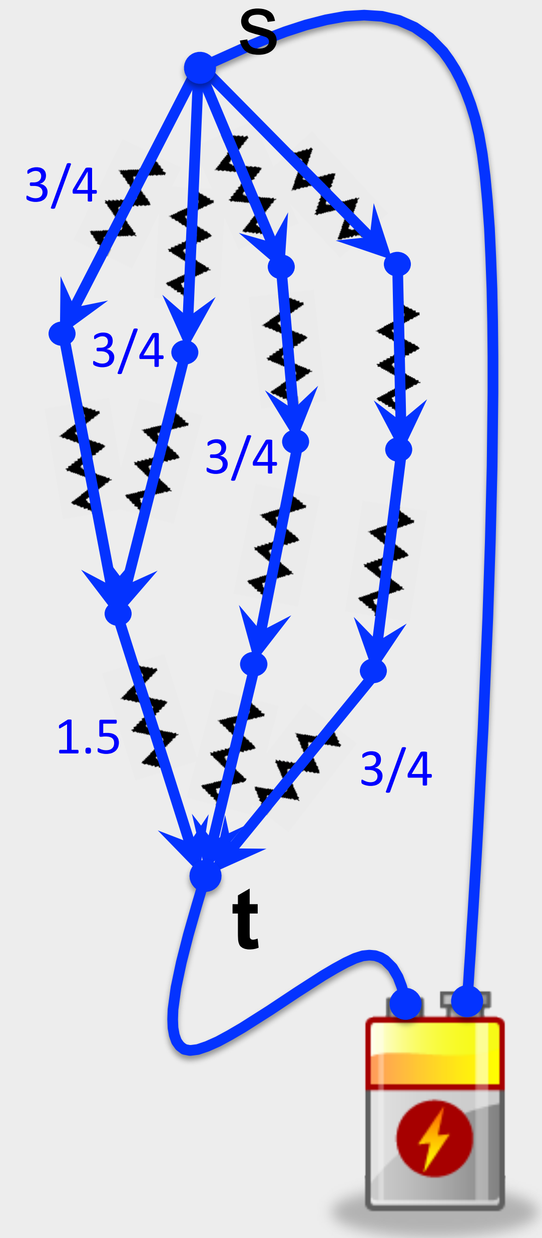
→ Treat edges as resistors of resistance **1**
→ Compute electrical flow of value $\mathbf{F}^*$



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 **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: F* known (via binary search)

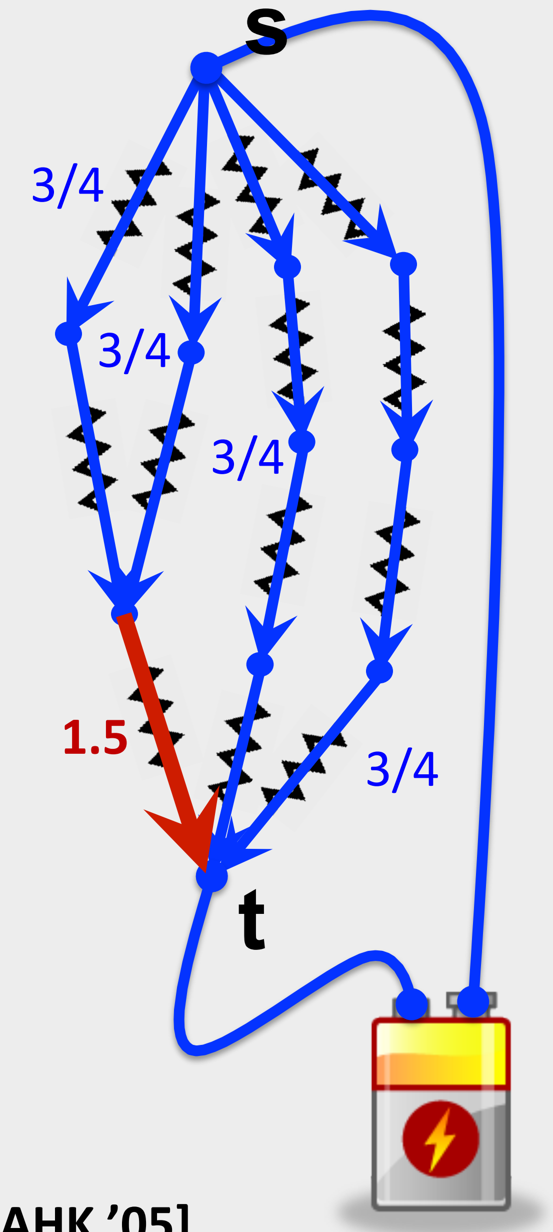
→ Treat edges as resistors of resistance **1**

→ Compute electrical flow of value **F***
(This flow has **no leaks**, but **can overflow** some edges)

→ To fix that: **Increase resistances** on the overflowing edges

Repeat

→ **At the end:** Take an **average** of all the flows as the final answer



Evolution of resistances:
Based on Multiplicative Weight Update method

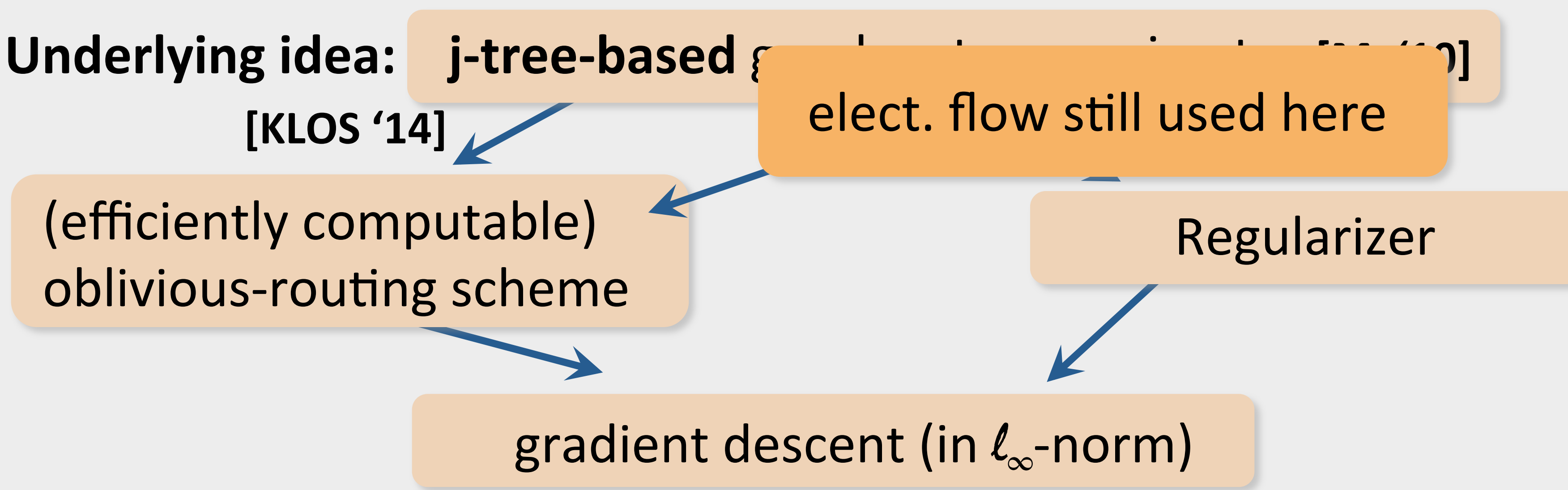
[FS ’97, PST ’95, AHK ’05]

This approach yields an **(1+ε)**-approx. to undirected max flow in **$\tilde{O}(m^{4/3}\varepsilon^{-3})$** time algorithm

[Lee Srivastava Rao '13]: A different perspective with a better dependence on **ε** for unit capacity graphs

[Kelner Miller Peng '12]: **(1+ε)**-approx. to undirected **k**-commodity flow in **$\tilde{O}(m^{4/3}\,\mathrm{poly}(k,\varepsilon^{-1}))$** time

[Sherman '13, Kelner Lee Orecchia Sidford '14]:
(1+ε)-approx. to undirected **k**-commodity flow in O**($m^{1+o(1)}\,k^2\,\varepsilon^{-2}$)** time



Electrical Flows and the Interior-point Methods

(Path-following) Interior-point method (IPM)

[Dikin '67, Karmarkar '84, Renegar '88,…]

A powerful framework for solving general LPs (and more)

LP: **min $c^T x$**
s.t. $Ax = b$
$x \geq 0$

Idea: Take care of "hard" constraints by adding a "barrier" to the objective

"**easy**" constraints (use projection)

"**hard**" constraints

(Path-following) Interior-point method (IPM)

[Dikin '67, Karmarkar '84, Renegar '88,…]

A powerful framework for solving general LPs (and more)

LP(μ): $\min c^T x - \mu \sum_i \log x_i$

s.t. $Ax = b$

~~$x \geq 0$~~

Idea: Take care of "hard" constraints by adding a "barrier" to the objective

Observe: The barrier term enforces $x \geq 0$ implicitly

Furthermore: for large **μ**, **LP(μ)** is easy to solve and

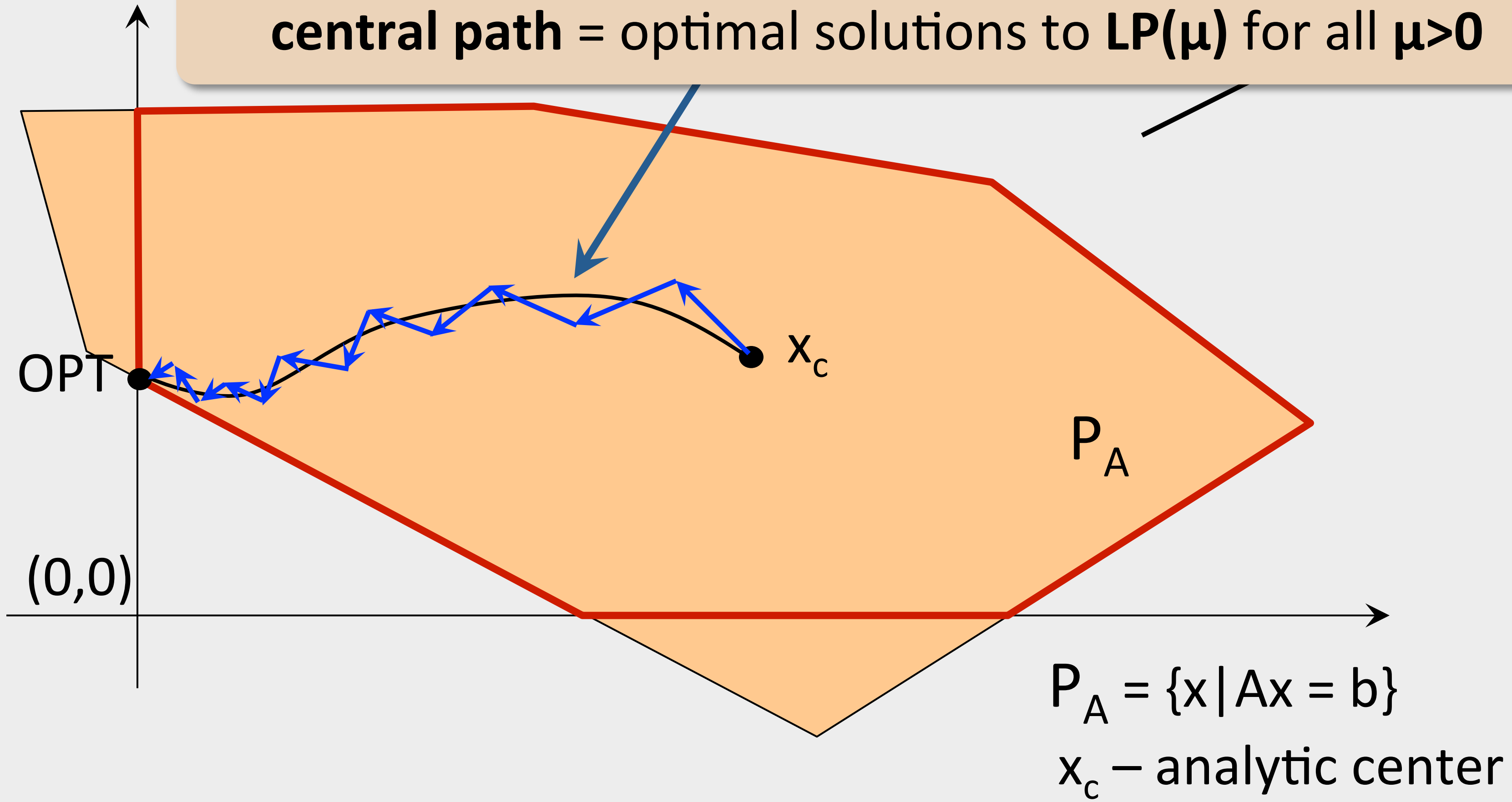
LP(μ) → original **LP**, as $\mu \to 0^+$

Path-following routine:

→ Start with (near-)optimal solution **x(μ)** to **LP(μ)** for large **μ>0**

→ Take an **improvement step** that gradually reduces **μ** while maintaining the (near-)optimality of **x(μ)** (wrt current **μ**)

central path = optimal solutions to **LP(μ)** for all **μ>0**



Path-following routine:

→ Start with (near-)optimal solution **x(μ)** to **LP(μ)** for large **μ>0**

→ Take an **improvement step** that gradually reduces **μ** while maintaining the (near-)optimality of **x(μ)** (wrt current **μ**)

How to guide our improvement steps?

Use electrical flows!

Step 1: Descent ("Predict")

Solve: $\min \sum_i r_i u_i^2$

s.t. $Au = b$

x = current solution
(near-optimal wrt **LP(μ)**)

$r_i = 1/x_i^2$

Set: $x_i' = (1-\delta)\, x_i + \delta\, u$

Effect: Decreasing **μ** at the e
of optimality (centrality) of

Formulation of electrical flow problem
if **A** = edge-vertex incidence matrix

How to guide our improvement steps?

Use electrical flows!

Step 2: Centering ("Correct")

x = current solution
(near-optimal wrt **LP(μ)**)

$\mathbf{r_i = 1/x_i^2}$

Set: $\mathbf{x_i' = (1-\delta)\, x_i + \delta\, u}$
$\mathbf{\mu' = (1-\delta)\, \mu}$

How to guide our improvement steps?

Use electrical flows!

Step 2: Centering ("Correct")

Solve: $\min \sum_i r_i' \hat{u}_i^2$
s.t. $A\hat{u} = b'$

x = current solution
(near-optimal wrt **LP(μ)**)

$r_i = 1/x_i^2$ $\quad r_i' = 1/(x_i')^2$

Set: $x_i' = (1-\delta)\, x_i + \delta\, u$
$\mu' = (1-\delta)\, \mu$

How to guide our improvement steps?

Use electrical flows!

Step 2: Centering ("Correct")

Solve: $\min \sum_i r_i' \hat{u}_i^2$
s.t. $A\hat{u} = b'$

x = current solution (near-optimal wrt **LP(μ)**)

$r_i = 1/x_i^2$ $r_i' = 1/(x_i')^2$

Set: $x_i' = (1-\delta)\, x_i + \delta\, u$
$\mu' = (1-\delta)\, \mu$
$x'' = (1-\delta)\, x_i' + \delta\, \hat{u}$

Effect: **x''** is near-optimal (centered) again and **μ** decreased

Can show: Setting $\delta \approx m^{-1/2}$ suffices (**m** = # of variables)
→ $O(m^{1/2} \log \varepsilon^{-1})$ iterations gives an **ε-optimal** solution **[Renegar '88]**

But: Understanding the electrical flow connection brings much more

Beyond the classical analysis of IPMs

Can show: Setting $\boldsymbol{\delta} \approx \mathbf{m}^{-1/2}$ suffices ($\mathbf{m}$ = # of variables)
→ $\mathbf{O(m^{1/2} \log \varepsilon^{-1})}$ iterations gives an **ε-optimal** solution **[Renegar '88]**

1) When solving **flow** problems → each iteration takes $\tilde{\mathbf{O}}(\mathbf{m})$ time

[Daitch Spielman '08]: $\tilde{\mathbf{O}}(\mathbf{m^{3/2} \log \varepsilon^{-1}})$ time alg. for "all" flow problems

2) Optimal setting of $\boldsymbol{\delta}$ corresponds to certain $\boldsymbol{\ell_2}$**- vs.** $\boldsymbol{\ell_4}$**-norm** interplay

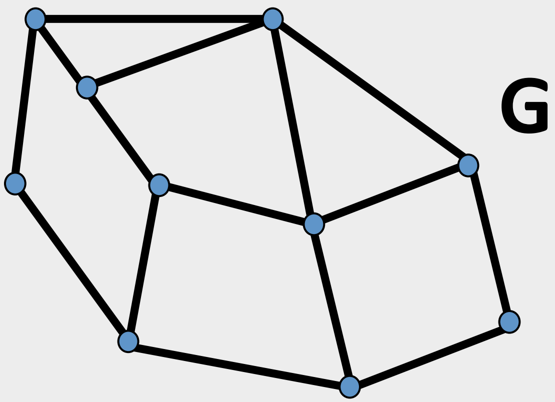
[M. '13]: $\tilde{\mathbf{O}}(\mathbf{m^{10/7}})$ time alg. for **unit-capacity** max flow
(via perturbation + preconditioning of intermediate sol./central path)

[Lee Sidford '13]: $\tilde{\mathbf{O}}(\mathbf{mn^{1/2} \log \varepsilon^{-1}})$ time alg. for "all" flow problems
(via a careful choice and reweighting of the barriers)

Electrical Flows, Random Walks, and Random Spanning Tree Generation

Random Spanning Trees

Goal: Output an uniformly random spanning tree

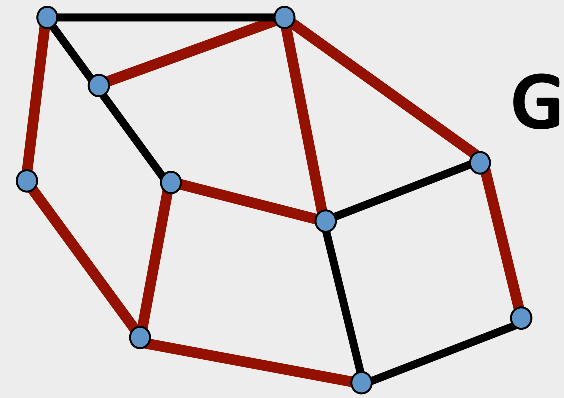


More precisely:

$\mathcal{T}$(**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}

More generally: Given weight w_e for each edge **e**

Task: Output a tree **T** with prob. proportional to $\Pi_e w_e$

Connection II: Random Spanning Trees

Key quantity: Effective resistance (between **s** and **t**)

$$R_{eff}(s,t) = \chi_{st}^T L^+ \chi_{st}$$

Vector with **1** at **t**, **-1** at **s** and **0**s everywhere else

Pseudo-inverse of the Laplacian

Connection II: Random Spanning Trees

Key quantity: Effective resistance (between **s** and **t**)

$$R_{eff}(s,t) = \chi_{st}^T L^+ \chi_{st}$$

Alternatively: $R_{eff}(s,t)$ **=** potential difference $\phi_t - \phi_s$ induced by an electrical flow **f** that sends a **unit current** from **s** to **t**

Yet differently (when all r_e=1 and (s,t) is an edge):
$R_{eff}(s,t)$ **=** the amount of flow f_{st} sent directly over the edge **(s,t)** by the **unit-value** electrical **s-t** flow **f**

Matrix Tree theorem [Kirchoff 1847]

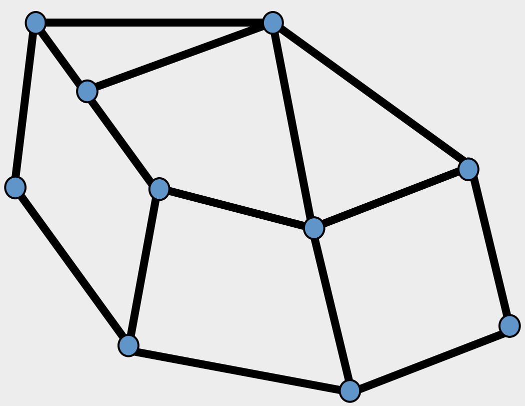
Pr[e in a rand. tree**]** = $w_e R_{eff}(e)$ where $r_e = 1/w_e$

Generating a Random Spanning Tree with Electr. Flows

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

→ For each $\mathbf{e_i}$:

- Compute $\mathbf{R_{eff}(e_i)}$ 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.

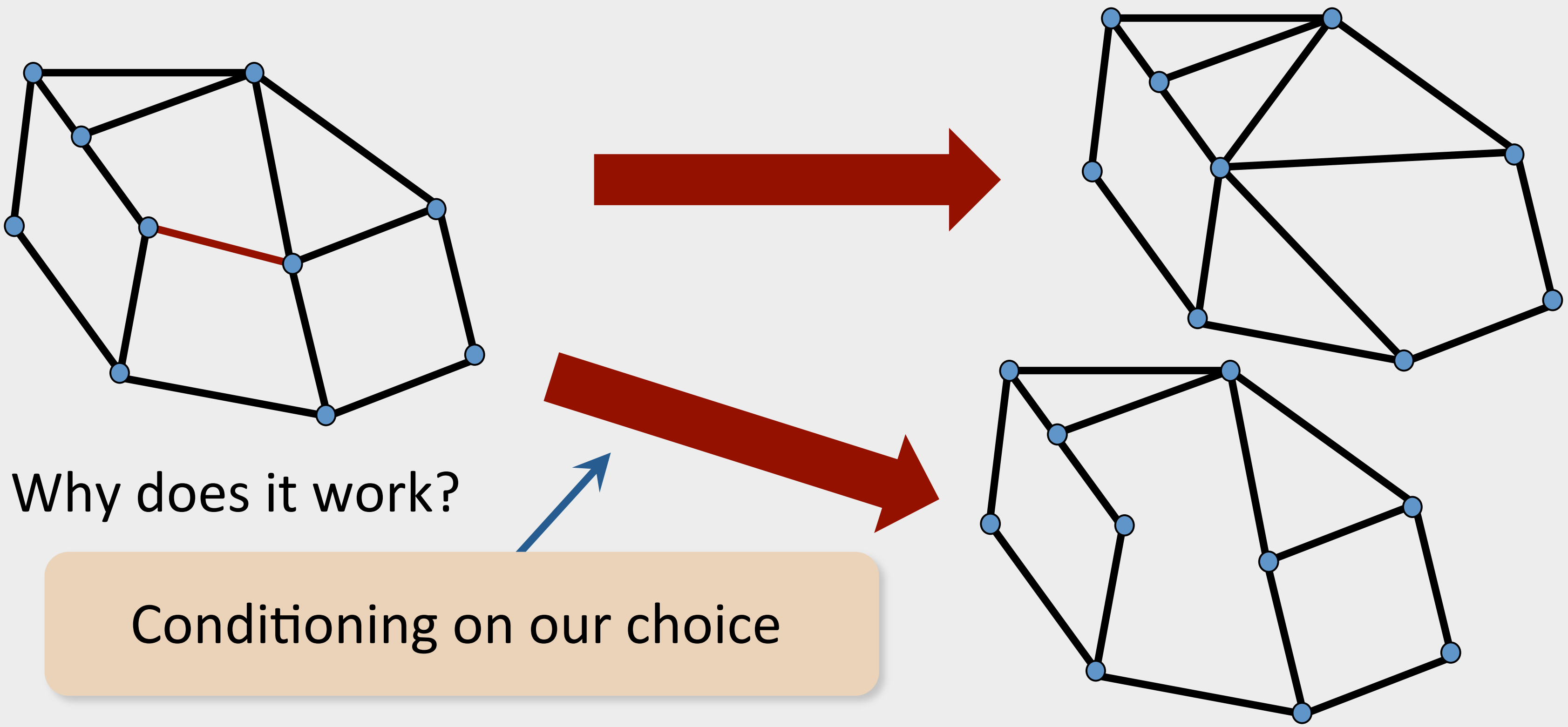


Generating a Random Spanning Tree with Electr. Flows

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

- Compute $\mathbf{R_{eff}(e_i)}$ 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**



Why does it work?

Conditioning on our choice

Generating a Random Spanning Tree with Electr. Flows

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

But: $\mathbf{R_{eff}(e) = \chi_e^T L^+ \chi_e}$ → Use Laplacian solver

Slight difficulty: Need to solve a Laplacian system ~~**exactly**~~

[Propp ‘0

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

Generating a Random Spanning Tree with Electr. Flows

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

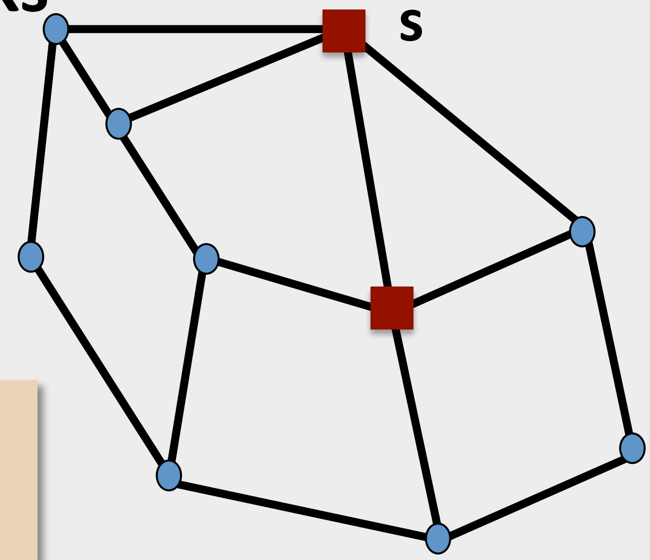
But: $\mathbf{R_{eff}(e) = \chi_e^T L^+ \chi_e}$ → Use Laplacian solver

Slight difficulty: Need to solve a Laplacian system ~~exactly~~

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

Can we do better? **Yes!** (At least when the graph is sparse.)

Rand. Spanning Trees and Random Walks



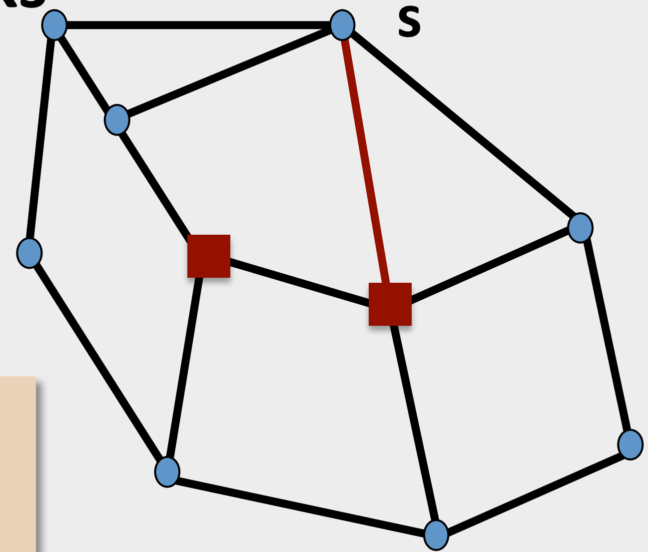
[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ 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**

Rand. Spanning Trees and Random Walks

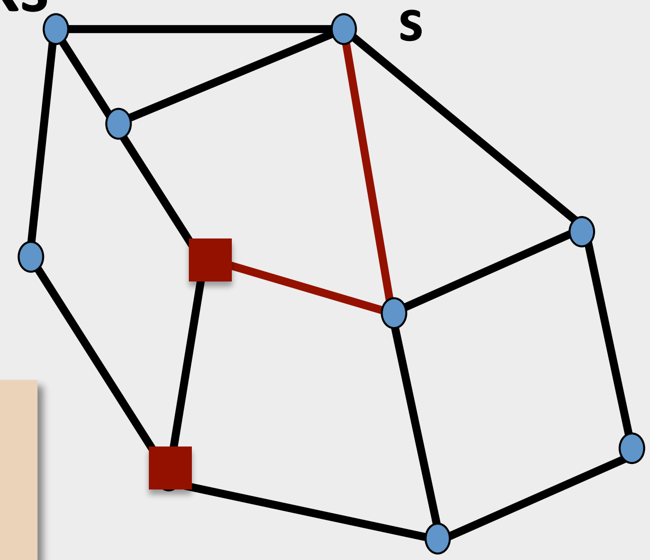
[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ 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**



Rand. Spanning Trees and Random Walks

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

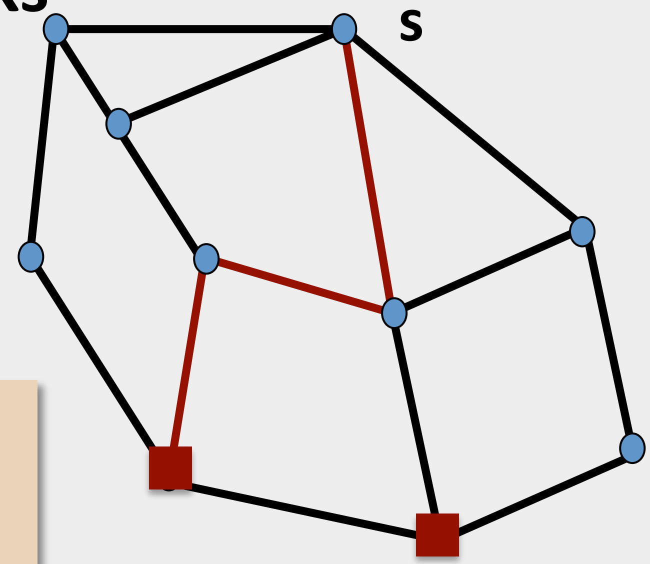


→ Start a random walk at some vertex **s**

→ Whenever visiting a new vertex **v**,
add to **T** the edge through which we visited

→ Output **T**

Rand. Spanning Trees and Random Walks



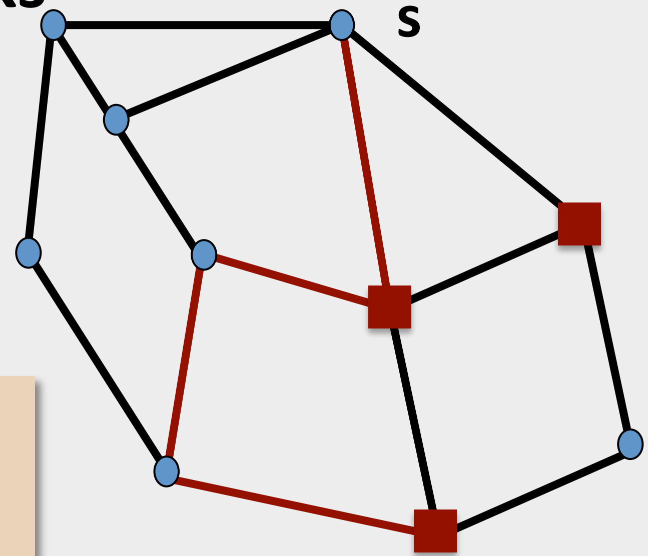
[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ 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**

Rand. Spanning Trees and Random Walks

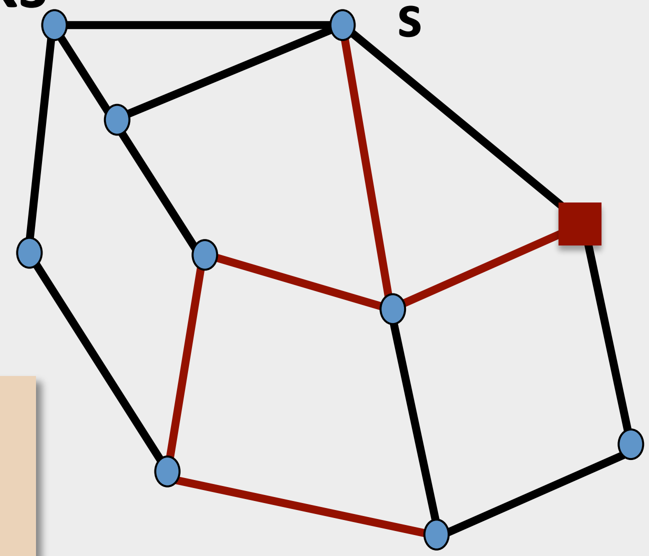


[Broder '89, Aldous '90]: Generate random spanning tree using random walks

→ 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**

Rand. Spanning Trees and Random Walks

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

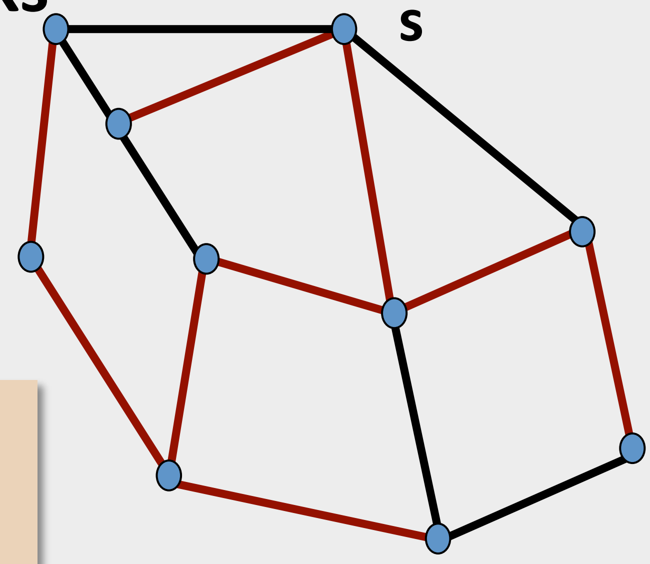


→ Start a random walk at some vertex **s**

→ Whenever visiting a new vertex **v**, add to **T** the edge through which we visited

→ Output **T**

Rand. Spanning Trees and Random Walks



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

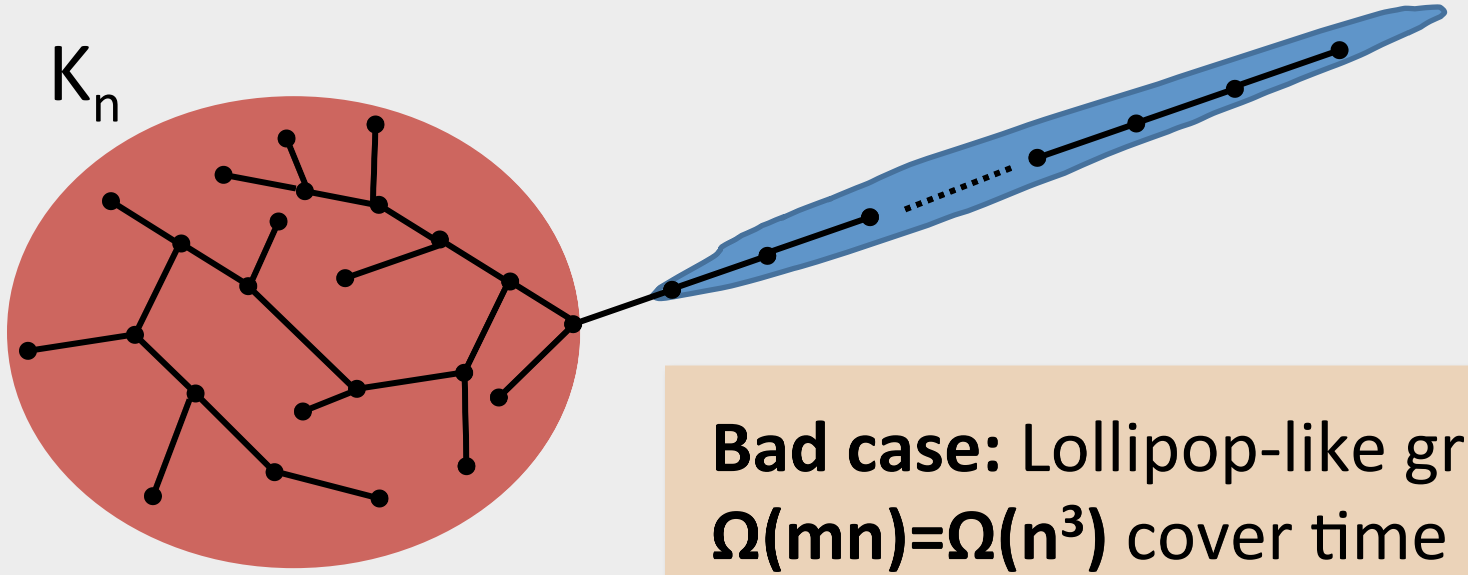
- → Start a random walk at some vertex **s**
- → Whenever visiting a new vertex **v**, add to **T** the edge through which we visited
- → Output **T**

Why does it work? Magic! (aka coupling from the past)

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?



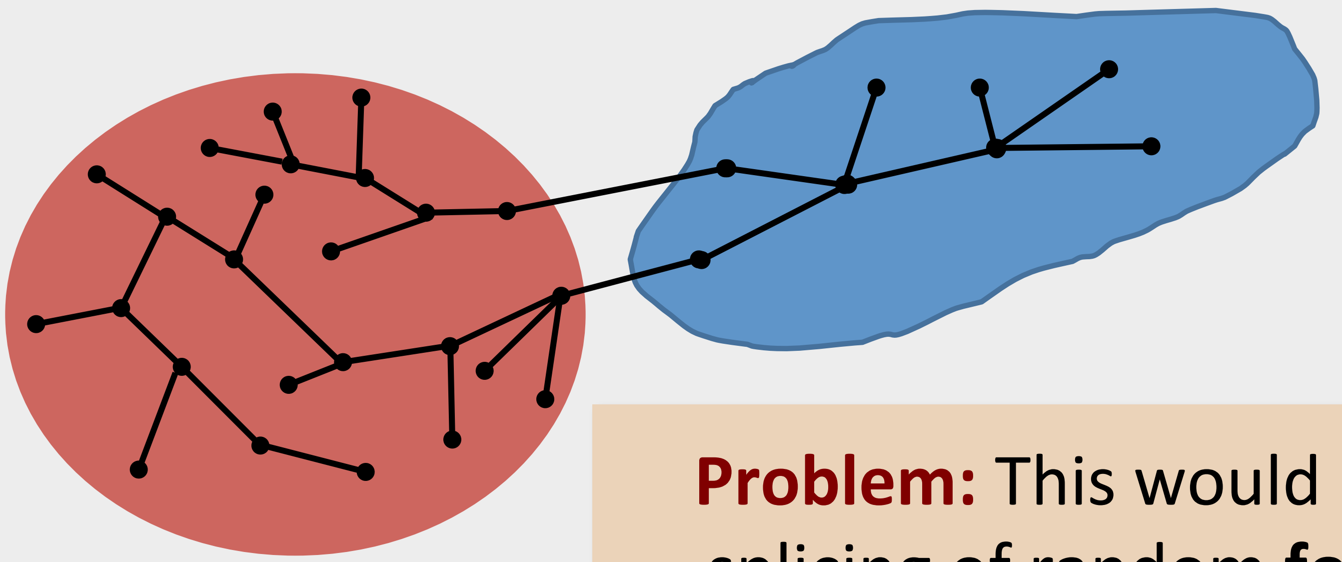
Bad case: Lollipop-like graph
$\Omega(mn)=\Omega(n^3)$ cover time

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**

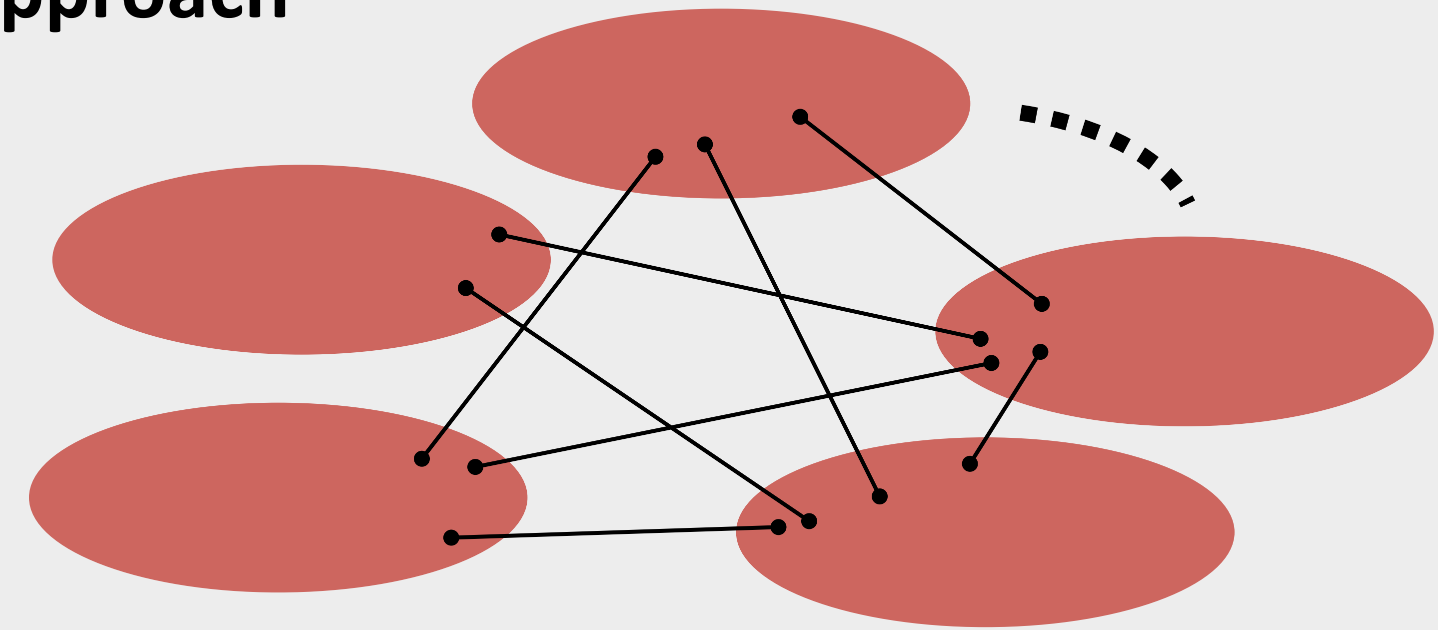
What happens: The walk resides mainly in K_n - the path-like part is covered only after a lot of attempts

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]



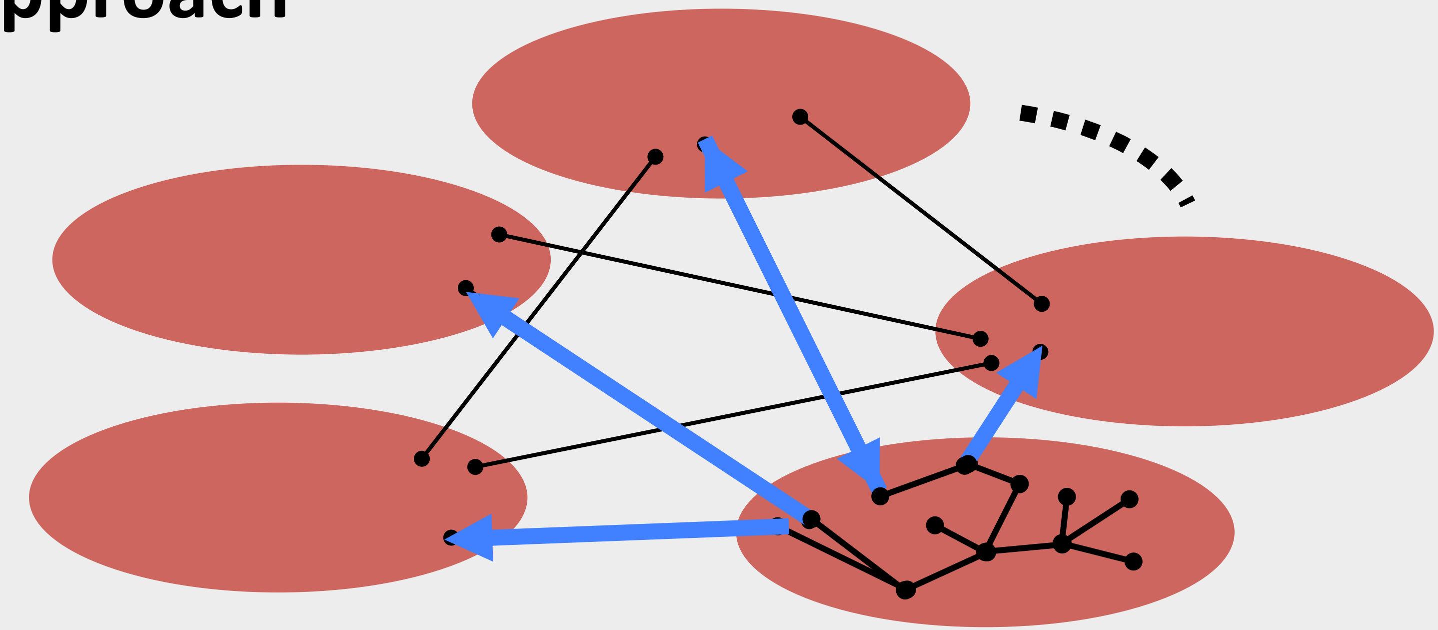
Decompose the graph into pieces with:

→ Low diameter each

→ Small "interface"

Different Approach

[Kelner M. '09]



Decompose the graph into pieces with:

→ Low diameter each

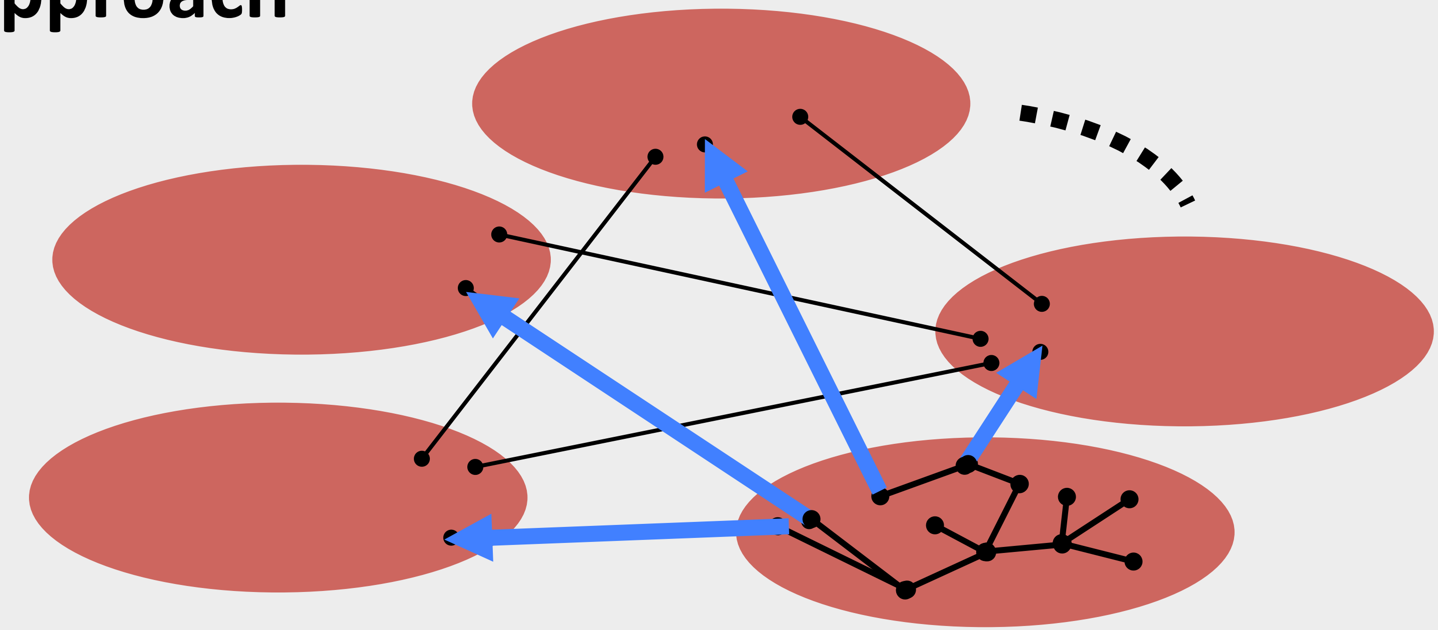
→ Small "interface"

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

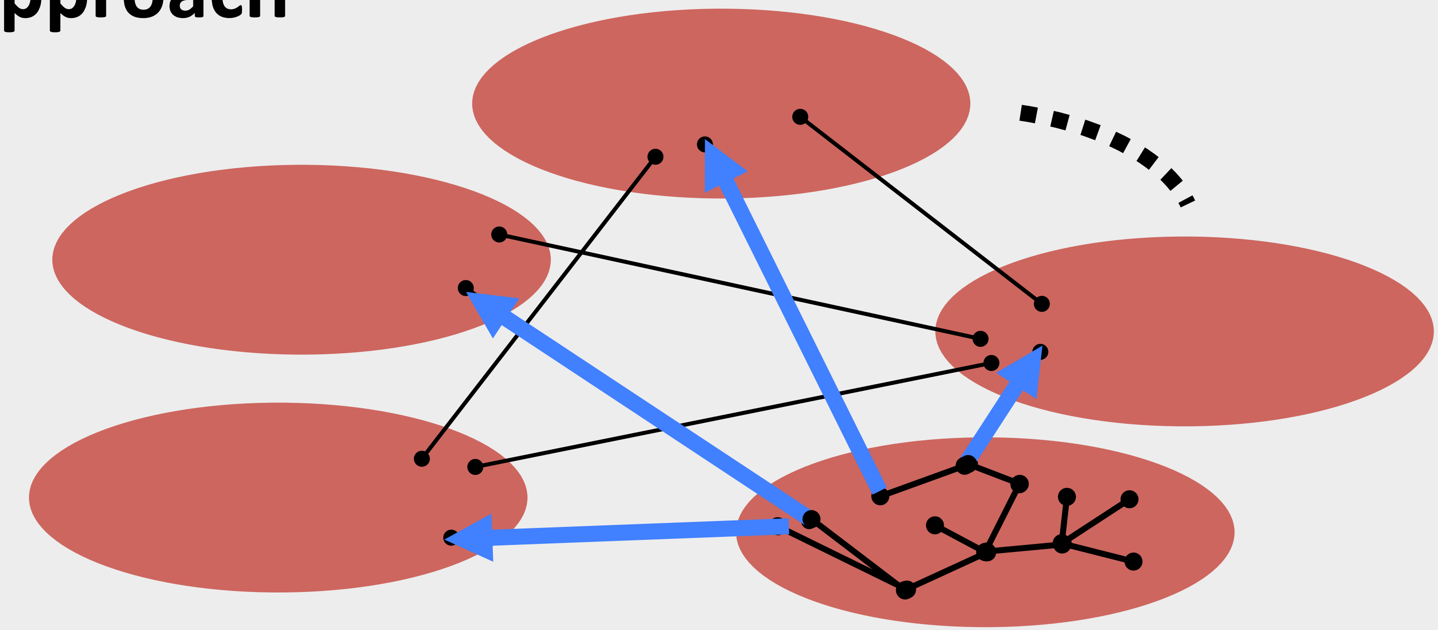
→ Small "interface"

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

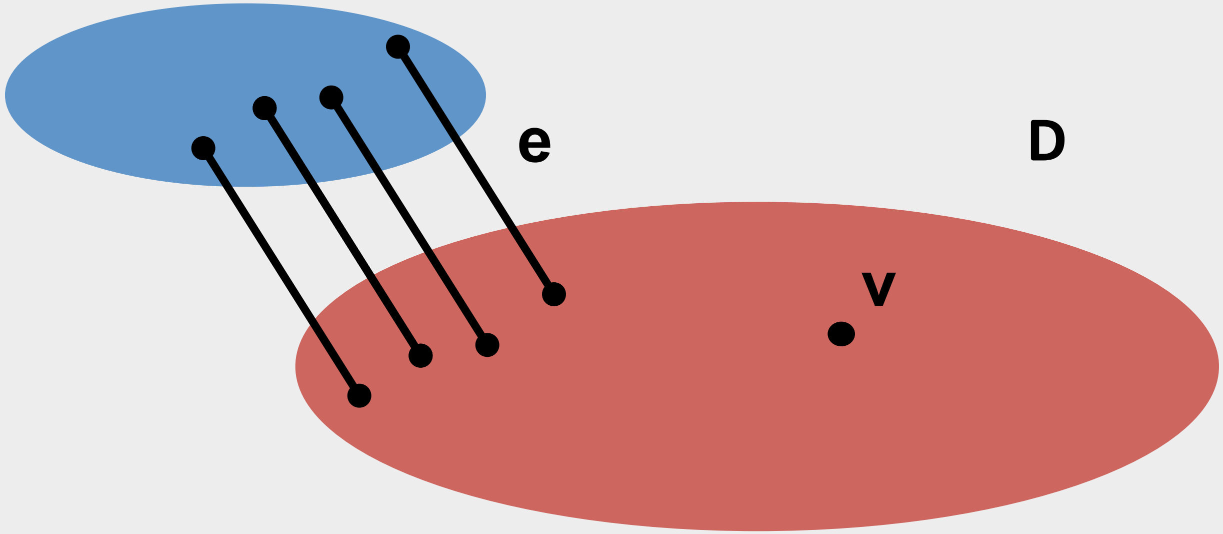
Modification: When simulating the random walk, **shortcut** revisits to pieces that were already explored in full

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

Missing element: How to compute the shortcutting jumps?

Different Approach

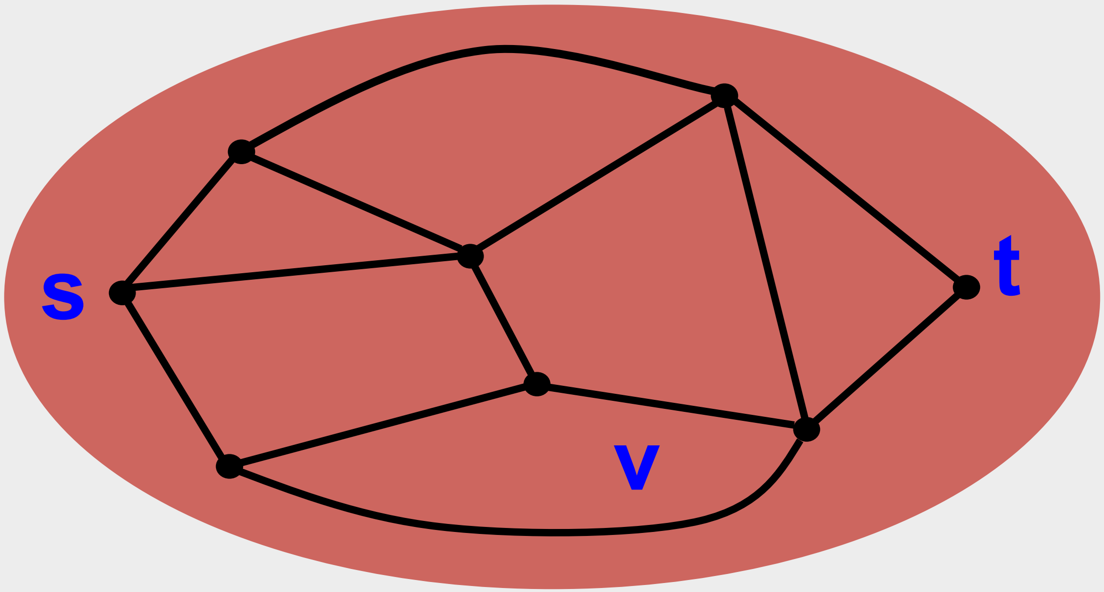
[Kelner M. '09]



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

Electrical flows can help us compute that!

Connection III: Absorption of Random Walks



Want to compute:
$q_{st}(v)$ = prob. that a random walk started at **v** reaches **t** before reaching **s**

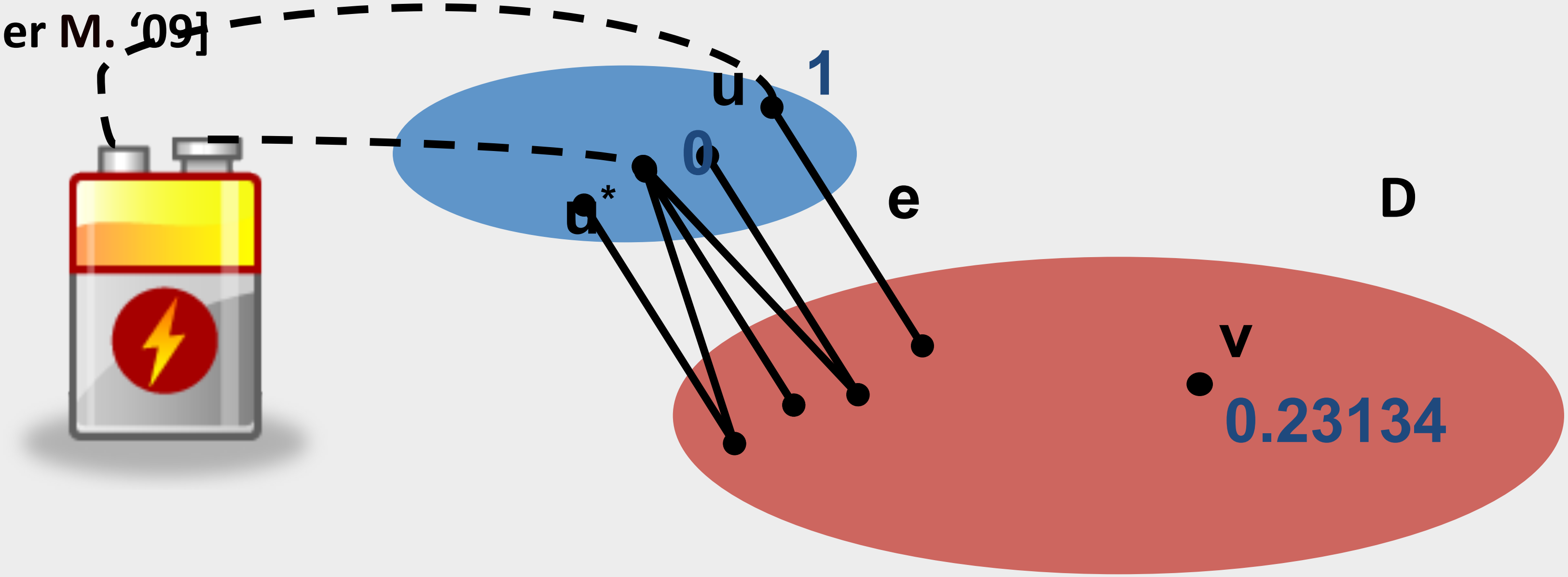
To compute $q_{st}(v)$:
→ Compute an electrical **s-t** flow
→ Normalize the voltage potentials **ϕ** so as **ϕ_s=0** and **ϕ_t=1**

Then: **$q_{st}(v) = \phi_v$**

Result: We can (approx.) compute **$q_{st}(v)$** for **all v** in **$\tilde{O}(m)$** time

Different Approach

[Kelner M. '09]



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

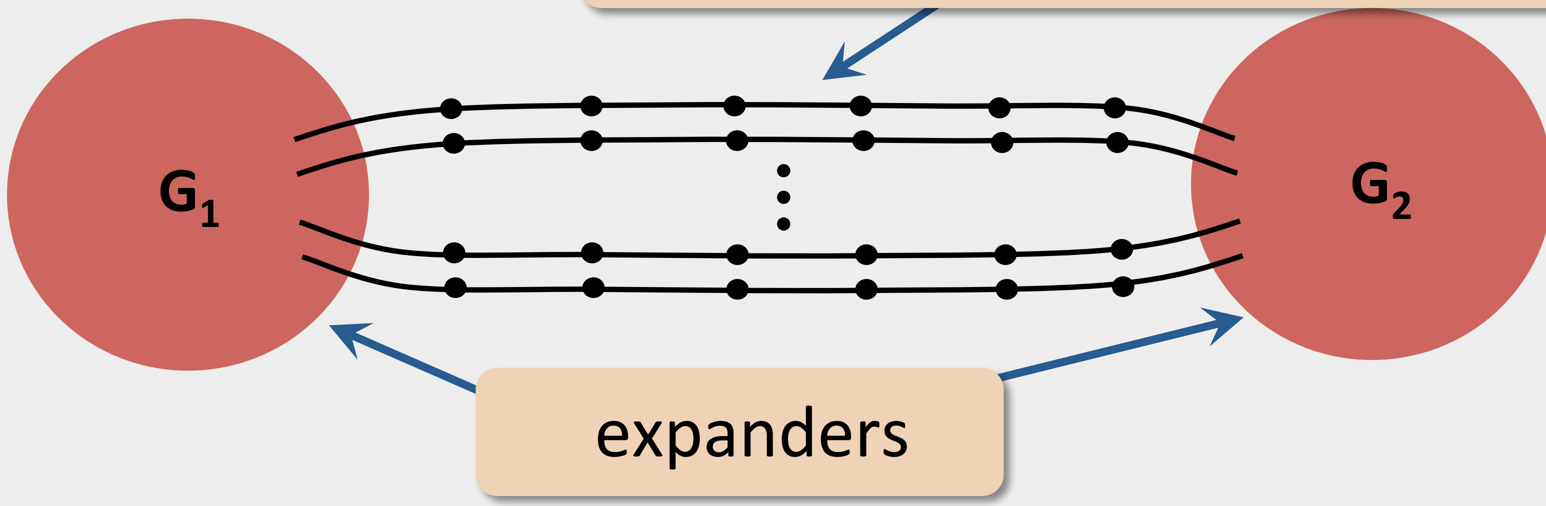
Observe: P_D(e,v) = q_{uu^*}(v)

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

Putting it all together: Generation of a random spanning tree in **$\tilde{O}$(mn½)** time

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

[M. Straszak Tarnawski '14]



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

To overcome this: Work with the "right" metric.

Connection IV: Cover and Commute Times of Random Walks

Recall (effective resistance between **s** and **t):**
R_{eff}(s,t) = potential difference **ϕ_t- ϕ_s** induced by
an electrical flow **f** that sends a **unit current** from **s** to **t**

Can show: For any two vertices **s** and **t**

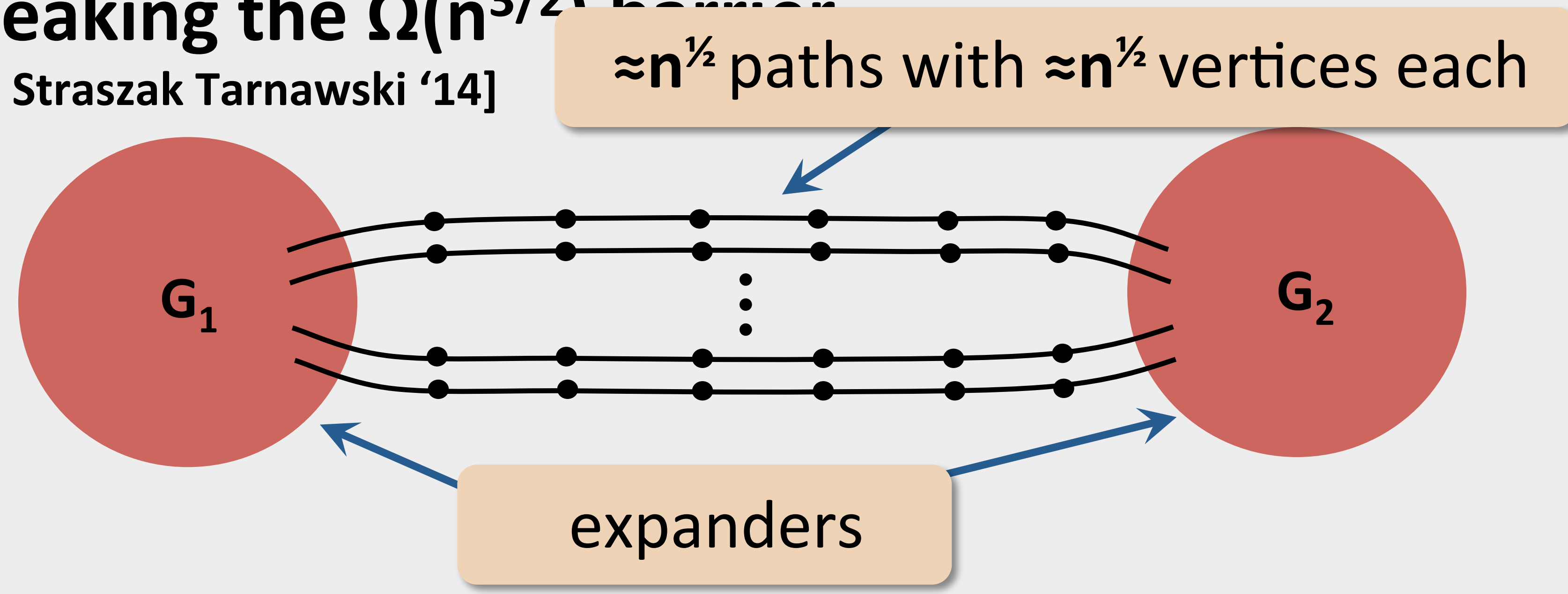
CommuteTime(s,t) = 2m R_{eff}(s,t)

[Matthews '88]: For any subgraph **D** of **effective resistance diameter γ**

CoverTime(D) = O(m γ)

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

[M. Straszak Tarnawski '14]



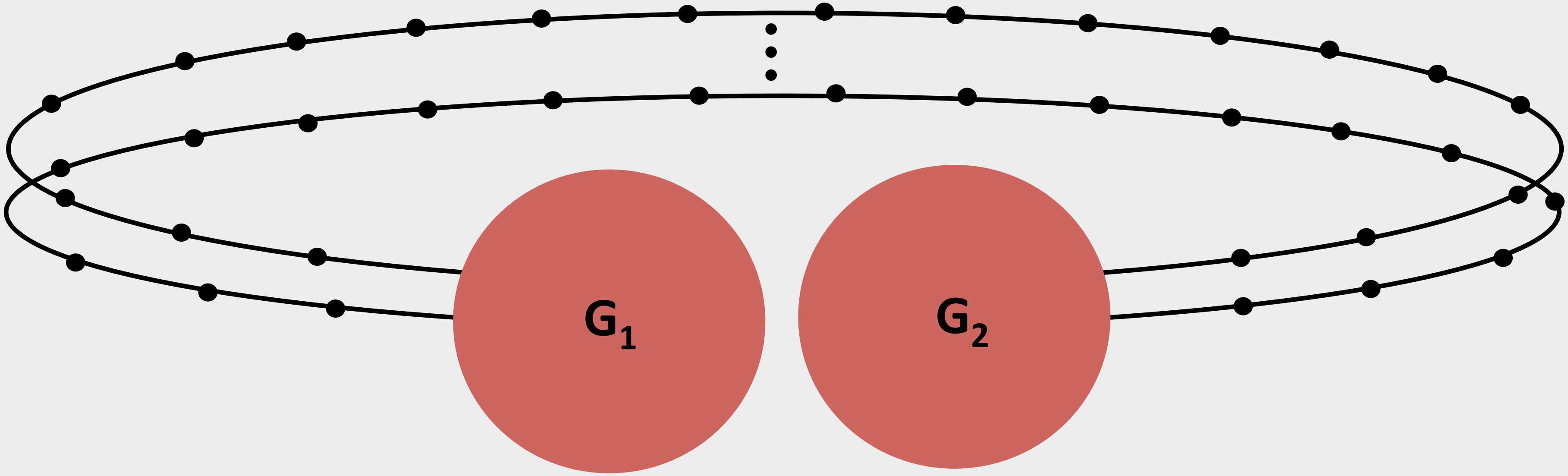
Problem: This graph has an $\Omega(n^{3/2})$ cover time and there is no nice way to cut it

To overcome this: Work with the "right" metric.

This graph looks much nicer in **effective resistance metric** (given by $L^{-1/2}$) than in the graph distance metric

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

[M. Straszak Tarnawski '14]



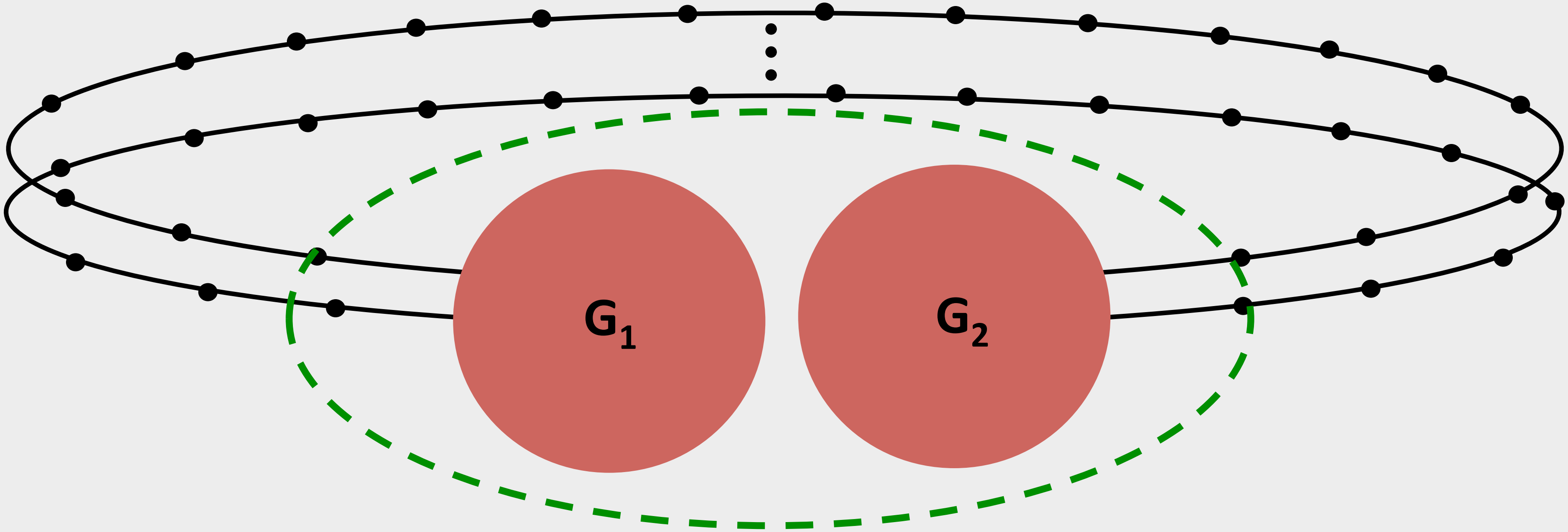
Problem: This graph has an $\boldsymbol{\Omega(n^{3/2})}$ cover time and there is no nice way to cut it

To overcome this: Work with the "right" metric.

This graph looks much nicer in **effective resistance metric** (given by $\mathbf{L}^{-1/2}$) than in the graph distance metric

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

[M. Straszak Tarnawski '14]



This identifies a **large, low-diameter region** in the **effective resistance metric** with **"simple" exterior**

Effective resistance diam. vs. Cover time connection:

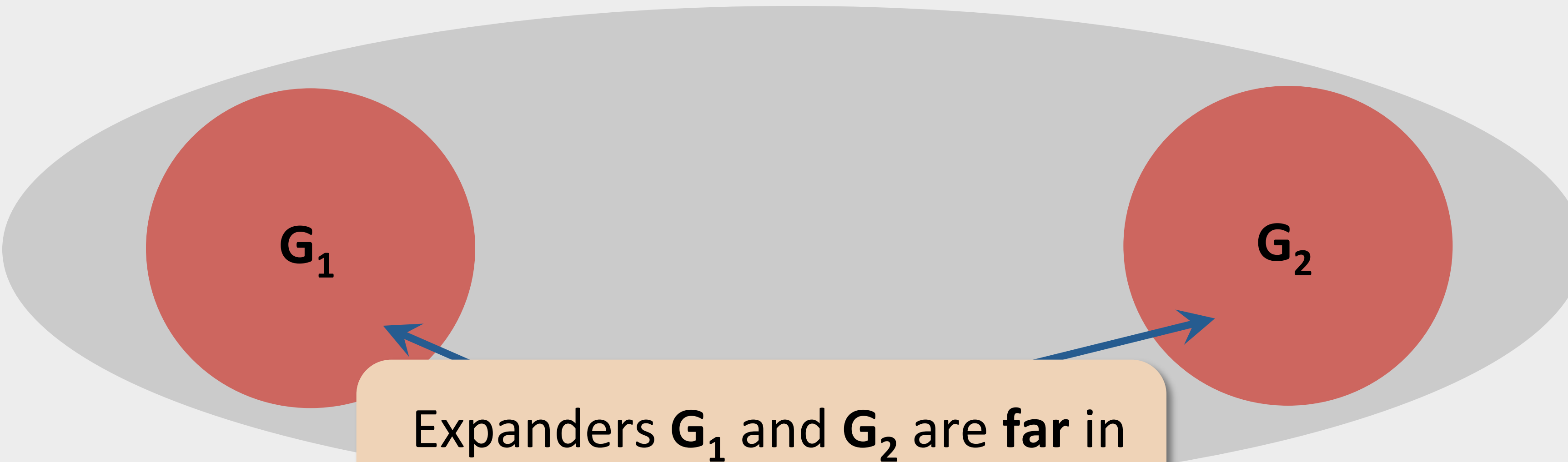
→ Figure out how the tree looks like in this region quickly

→ **Condition** on this choice (by modifying the graph) and proceed to next phase (we made a lot of progress!)

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

[M. Straszak Tarnawski '14]

Missing element: What if the exterior of our large low-diameter region is **not** "simple"?



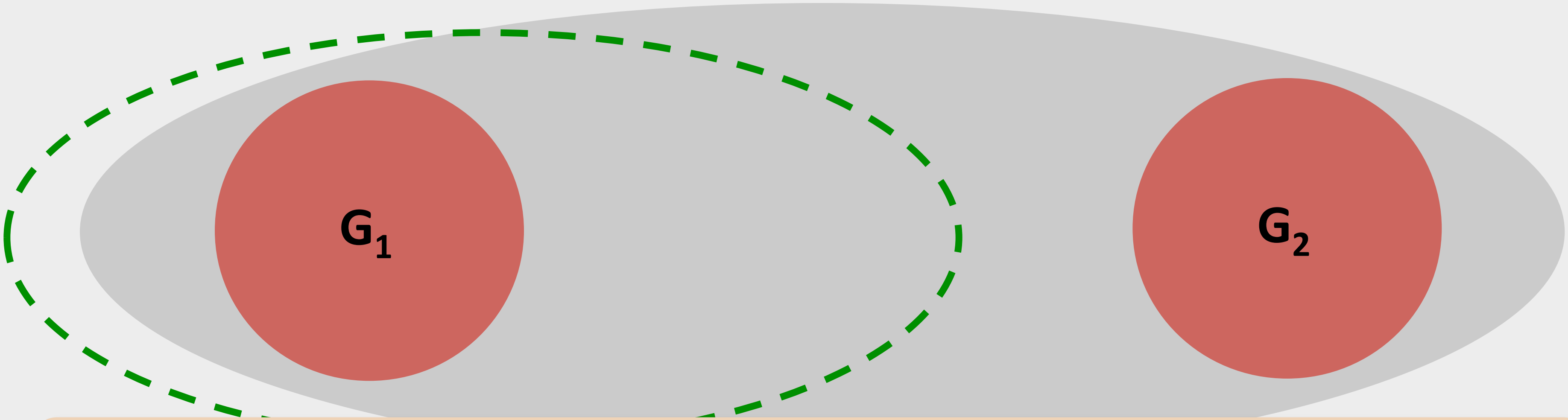
Expanders $\mathbf{G_1}$ and $\mathbf{G_2}$ are **far** in the effective resistance metric

Tie effect. resist. to graph cuts: Argue that there exists a **good cut** separating the two regions

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

[M. Straszak Tarnawski '14]

Missing element: What if the exterior of our large low-diameter region is **not** "simple"?



Putting it all together: An $O(m^{4/3+o(1)})$ time sampling algorithm

Tie effect. resist. to graph cuts: Argue that there exists a **good cut** separating the two regions

Electrical Flows and Cuts

Electrical Flows and Sparsification [Spielman Srivastava '08]

A simple sparsification algorithm:

→ Given graph **G**, **compute** effective resistances $\mathbf{R_{eff}(e)}$ of all edges

→ **Sample ≈ ε^{-2}n log n** edges **independently** (with replacements) and proportionally to their effective resistances $\mathbf{R_{eff}(e)}$

→ Take **H = union** of all the sampled edges (after reweighting by an inverse of sampling probability)

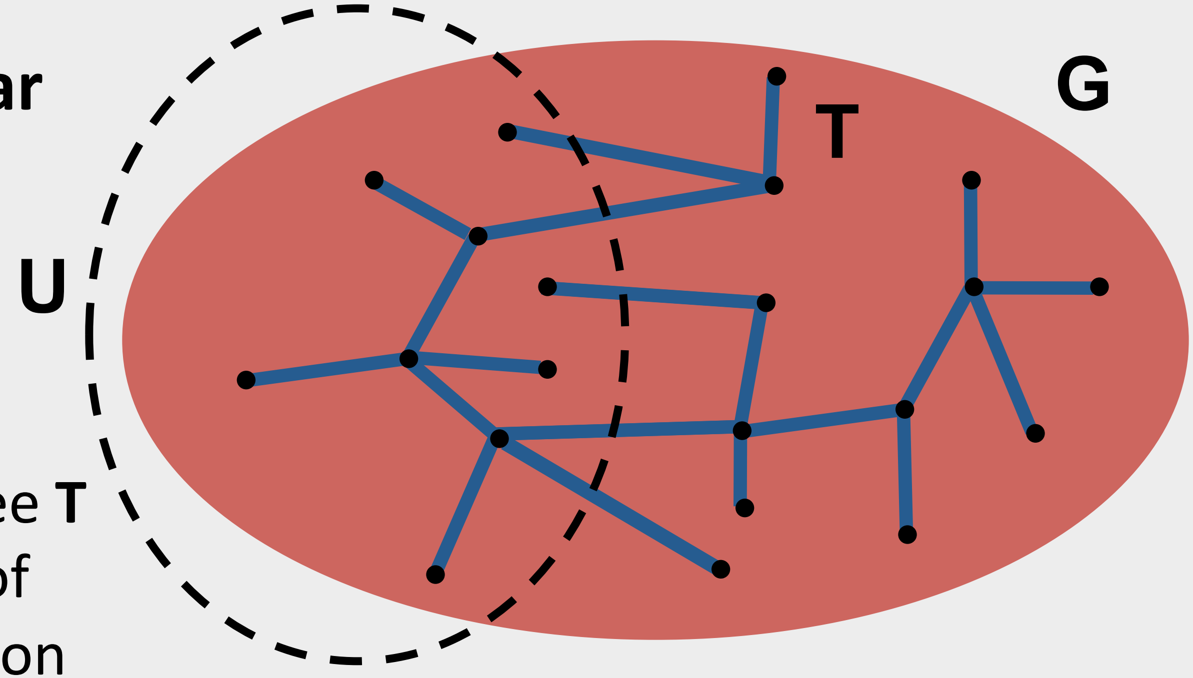
[Spielman Srivastava '08]: Whp **H** preserves all the cuts of **G** up to a multiplicative error of **(1+ε)**

Bottom line: $\mathbf{R_{eff}(e)}$ is a good measure of importance of an edge **e** wrt connectivity.

Electrical Flows and Thin Trees

[Asadpour Goemans M. Oveis Gharan Saberi '10]

Assume: G k-regular
and **k-connected**



Think: Finding such tree **T** is an "extreme" form of (upper) cut sparsification

Spanning tree T is **α-thin** iff

$\delta_T(U) \leq \alpha\, \delta_G(U)/k$, for every cut **U**

[Asadpour Goemans M. Oveis Gharan Saberi '10]:

Ability to find **α-thin trees** = **O(α)-approx.** to ATSP

Electrical Flows and Thin Trees

[Asadpour Goemans M. Oveis Gharan Saberi '10]

How to find good **α**-thin trees?

Independent sampling: Gives **α=Θ(log n)**

A better way:
(1) Compute weights $\mathbf{w_e}$ so as **Pr[e** in a rand. tree**]** close to **uniform**
(2) Sample a random spanning tree with respect to these weights

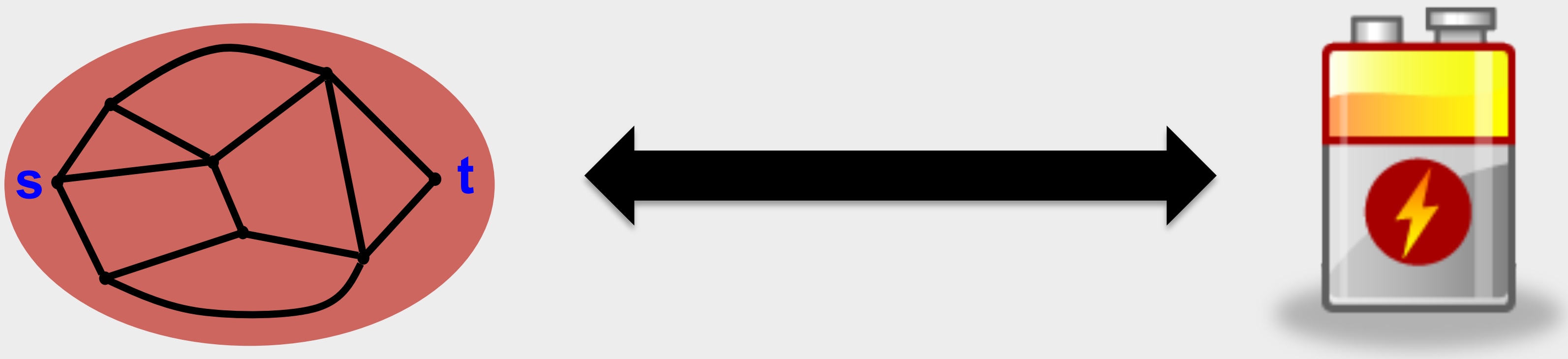
(1) → All cuts are good in expectation

Negative correlation of random tree edge sampling
+ Karger's "trick" → Gives **α=Θ(log n/log log n)**
Can we do better?

[Markus Spielman Srivastava '13]+[Harvey Olver '14]:
Can get **α=O(1)** for a **spectral** analogue of the question

Conclusions

Electrical Flows and Graph Algorithms



Electrical flows: A powerful new tool for fast graph alg.

→ Wealth of connections to graphs

→ Very fast computation

Key question: Where else can electrical flows primitive be useful?

Electrical Flows and Graph Algorithms

Key question: Where else can electrical flows primitive be useful?

- → Advancing our understanding of the convergence of interior-point methods? Inspire new types of IPMs?
- → Better "spectral" graph algorithms? (Stability is the key?)
- → What other properties of random walks we can get a hold on using electrical flows?
- → Effective resistance metric as a basic way of looking at graphs? (Seems more robust than shortest-path metric)
- → Theory of directed flows/walks?

Thank you

Questions?