Fast Generation of Random Spanning Trees and the Effective Resistance Metric

Jakub Tarnawski

joint work with Aleksander Mądry and Damian Straszak

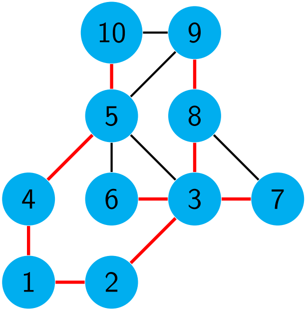
EPFL, Lausanne, Switzerland
University of Wrocław, Poland

Problem: given an undirected graph $G = (V, E)$, sample a spanning tree T uniformly at random.

Notation

$\mathcal{T}(G)$: set of spanning trees of G

Output every tree T with prob. $1/|\mathcal{T}(G)|$.



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

- Matrix-tree-theorem methods
  - $\mathcal{O}(mn^{\omega})$ (Guenoche 1983)
  - $\mathcal{O}(n^{\omega})$ (Colbourn et al. 1996)
- Random-walk methods
  - $\mathcal{O}(mn)$ (Aldous 1990, Broder 1989)
  - $\widetilde{\mathcal{O}}(m\sqrt{n})$ (Kelner-Mądry 2009)

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

- Matrix-tree-theorem methods
  - $\mathcal{O}(mn^\omega)$ (Guenoche 1983)
  - $\mathcal{O}(n^\omega)$ (Colbourn et al. 1996)
- Random-walk methods
  - $\mathcal{O}(mn)$ (Aldous 1990, Broder 1989)
  - $\widetilde{\mathcal{O}}(m\sqrt{n})$ (Kelner-Mądry 2009)
  - $\widetilde{\mathcal{O}}(m^{4/3})$ (this work)

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

- Matrix-tree-theorem methods (good for dense graphs)
  - $\mathcal{O}(mn^{\omega})$ (Guenoche 1983)
  - $\mathcal{O}(n^{\omega})$ (Colbourn et al. 1996)
- Random-walk methods (good for sparse graphs)
  - $\mathcal{O}(mn)$ (Aldous 1990, Broder 1989)
  - $\widetilde{\mathcal{O}}(m\sqrt{n})$ (Kelner-Mądry 2009)
  - $\widetilde{\mathcal{O}}(m^{4/3})$ (this work)

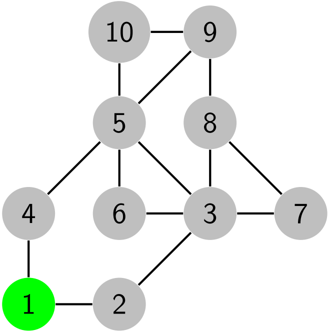
Remark: $|\mathcal{T}(G)|$ can be as large as n^{n-2}.

- Matrix-tree-theorem methods (good for dense graphs)
  - $\mathcal{O}(n^{3.37})$ (Guenoche 1983)
  - $\mathcal{O}(n^{2.37})$ (Colbourn et al. 1996)
- Random-walk methods (good for sparse graphs)
  - $\mathcal{O}(n^{2.00})$ (Aldous 1990, Broder 1989)
  - $\widetilde{\mathcal{O}}(n^{1.50})$ (Kelner-Mądry 2009)
  - $\widetilde{\mathcal{O}}(n^{1.33})$ (this work)

assuming $m = \mathcal{O}(n)$.
Fastest known algorithm for $m \leqslant \mathcal{O}(n^{1.5})$.

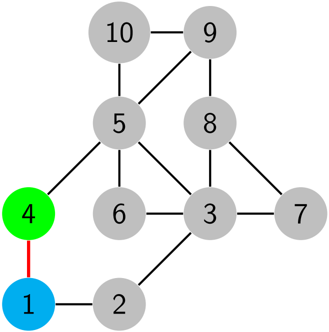
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



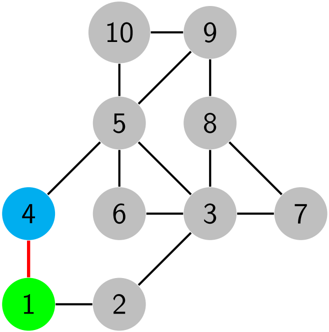
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



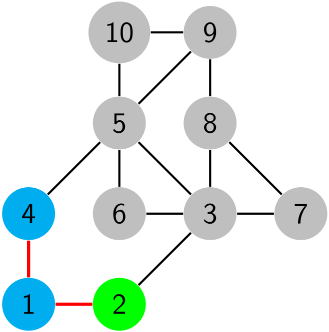
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



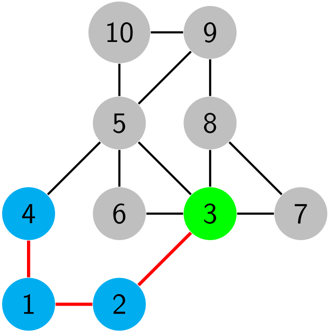
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit, add to T the edge through which it was hit.
- Once G is covered, output T.



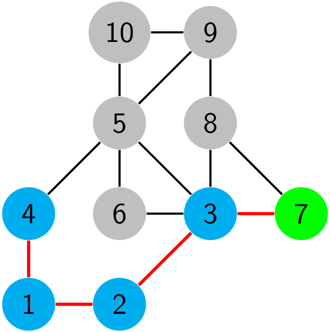
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



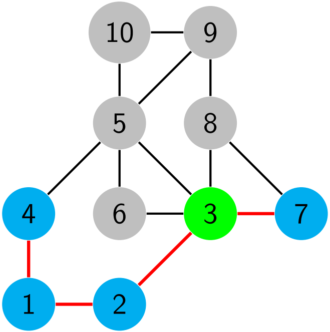
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



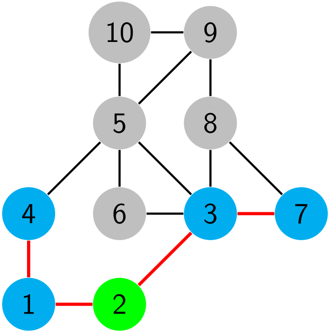
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



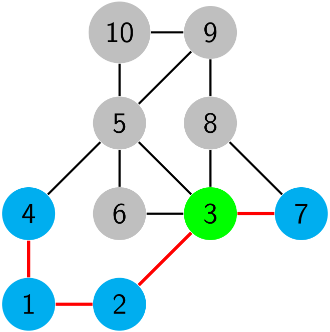
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



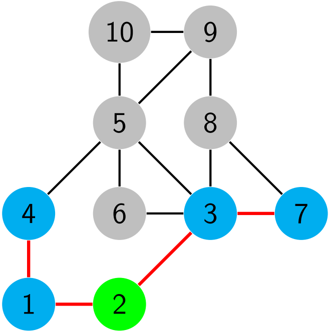
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



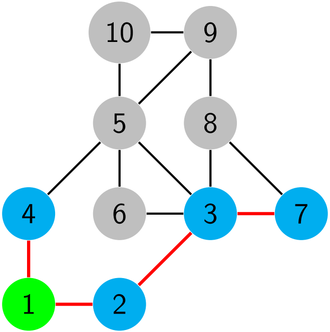
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



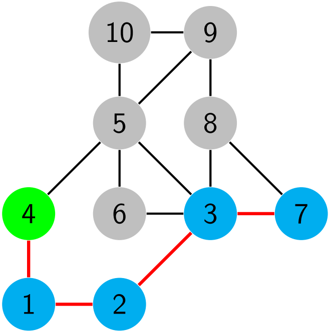
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



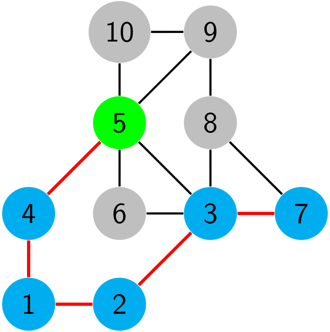
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



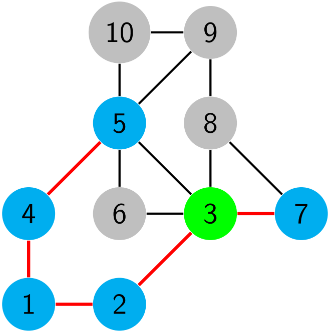
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



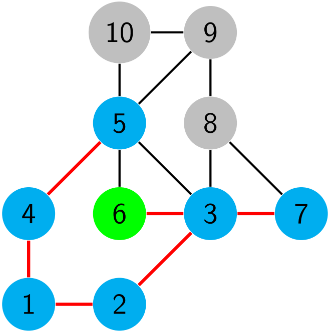
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



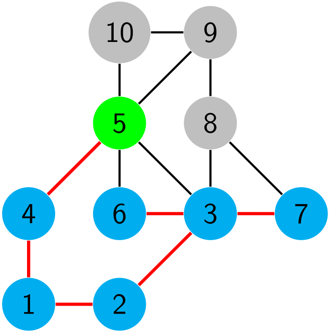
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



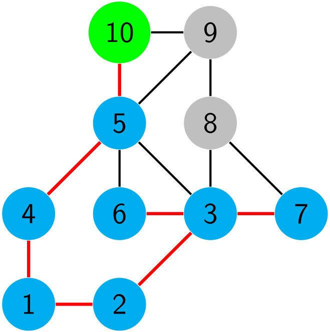
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



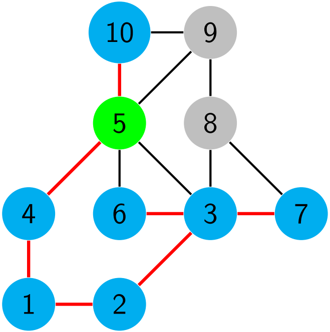
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



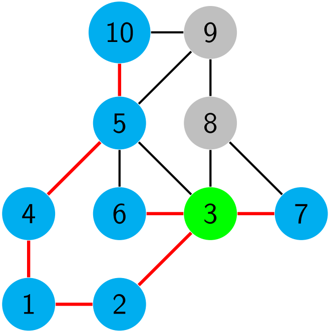
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



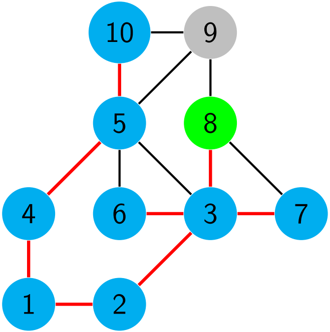
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



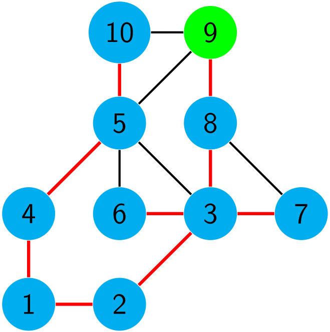
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



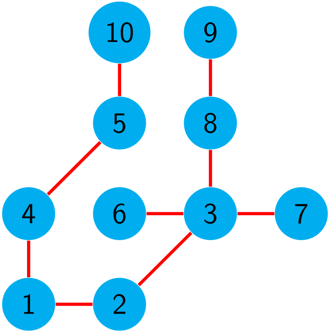
Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.



Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit, add to T the edge through which it was hit.
- Once G is covered, output T.

Theorem (Aldous 1990, Broder 1989)

The distribution of T is uniform on $\mathcal{T}(G)$.

Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.

Theorem (Aldous 1990, Broder 1989)

The distribution of T is uniform on $\mathcal{T}(G)$.

Time $= \mathcal{O}(\text{cover time}) = \mathcal{O}(mn)$.

Random-walk method for sampling RSTs

- Run a random walk on G.
- Whenever a new vertex is hit,
  add to T the edge through which it was hit.
- Once G is covered, output T.

Theorem (Aldous 1990, Broder 1989)

The distribution of T is uniform on $\mathcal{T}(G)$.

Time $= \mathcal{O}(\text{cover time}) = \mathcal{O}(mn)$.
Question: do we need to simulate this process in full?

Improving upon $\mathcal{O}(mn)$

(Kelner-Mądry, FOCS 2009)

Improving upon $\mathcal{O}(mn)$
(Kelner-Mądry, FOCS 2009)

First step: find a bad example.

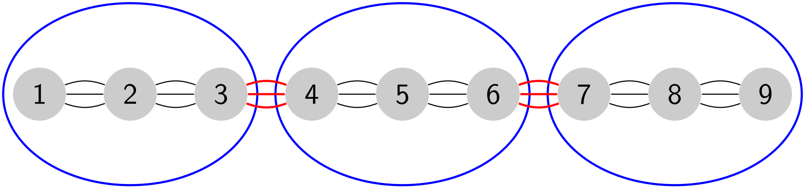
$$\text{cov}(G) = \Theta(n^2)$$

Out of the $\mathcal{O}(nm)$ steps of the walk, we only care about $n-1$ (those that visit a new vertex).

$$\mathrm{cov}(G) = \Theta(n^2)$$

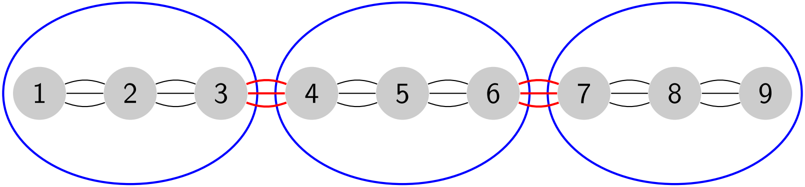
Out of the $\mathcal{O}(nm)$ steps of the walk, we only care about $n-1$ (those that visit a new vertex).

Issue: too much walking over already-explored parts.
(We gain no new information this way.)



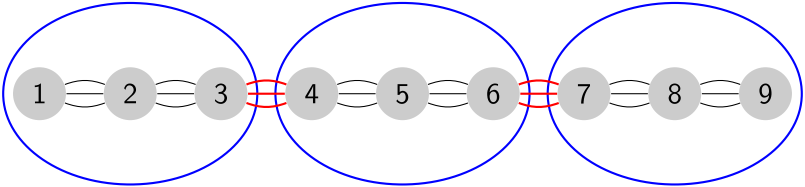
Plan:

- after we cover a blue subgraph we don't want to traverse it anymore
- whenever we return there, we'd rather just *know* through which edge we will exit, and exit (*shortcutting the walk*)



Requirements:

- we want to keep the cover time of each blue subgraph low
- walking the red edges is costly – we want to reduce their number

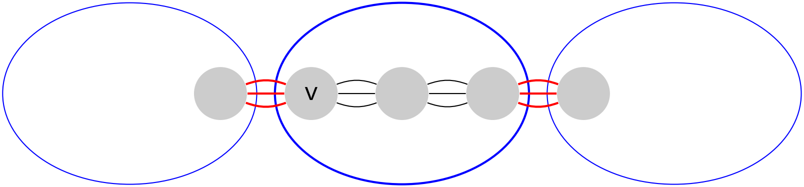


Fact (Leighton, Rao 1999)

Can partition G into regions such that:

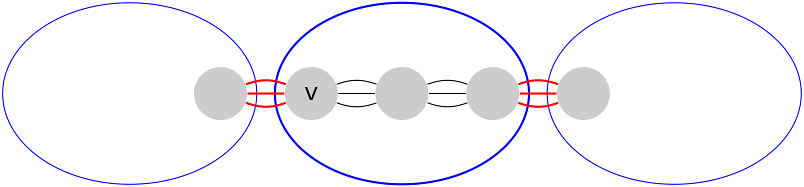
- *diameters of regions are small ($\sqrt{m}$),*
- *number of cut edges is small ($\sqrt{m}$).*

- Walking over each region until it is covered takes $\widetilde{\mathcal{O}}(m^{3/2})$ steps in total.
- Walking the red edges until G is covered takes $\widetilde{\mathcal{O}}(m^{3/2})$ steps.



Task: for a vertex v from the region and an edge e from the region's boundary, compute $P_v(e)$: probability that a walk started at v will exit the region through e.

This can be done using electrical flows and fast Laplacian solvers, also in time $\widetilde{\mathcal{O}}(m^{3/2})$.



Task: for a vertex v from the region and an edge e from the region's boundary, compute $P_v(e)$: probability that a walk started at v will exit the region through e.

This can be done using electrical flows and fast Laplacian solvers, also in time $\widetilde{\mathcal{O}}(m^{3/2})$.

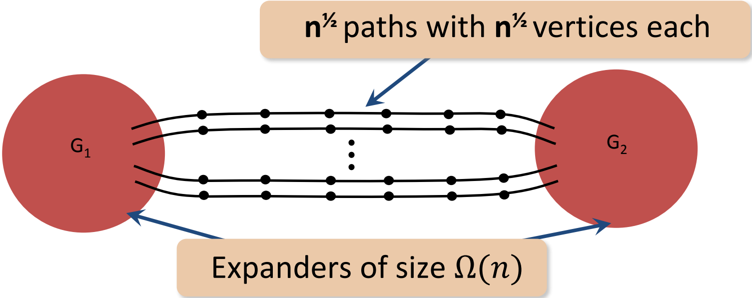
Theorem (Kelner, Mądry 2009)

One can sample a uniformly random spanning tree in time $\widetilde{\mathcal{O}}(m^{3/2})$. Can be improved to $\widetilde{\mathcal{O}}(m\sqrt{n})$.

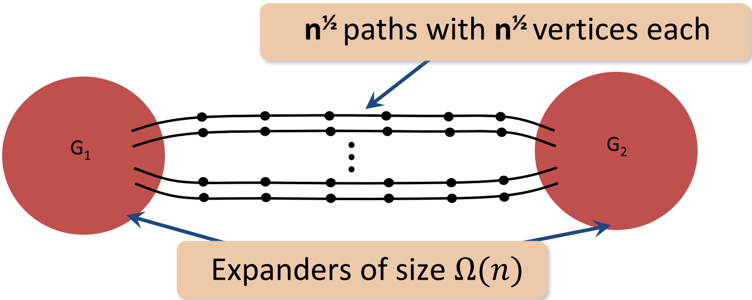
Improving upon $\widetilde{\mathcal{O}}(m\sqrt{n})$
(this work)

Improving upon $\widetilde{\mathcal{O}}(m\sqrt{n})$
(this work)

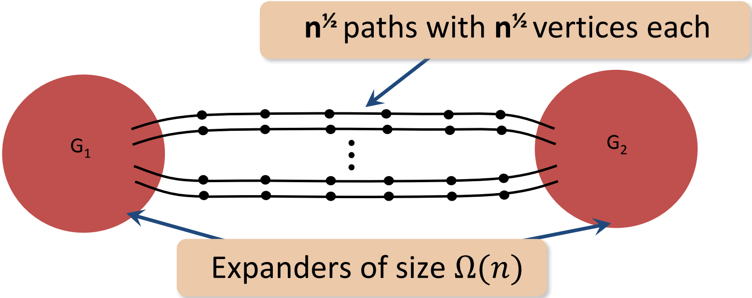
First step: find a bad example.



- Diameter: $\Theta(\sqrt{n})$
- Cover time: $\Theta(n^{3/2})$ – Kelner-Mądry gives no improvement over simple random-walk algorithm



- Diameter: $\Theta(\sqrt{n})$
- Cover time: $\Theta(n^{3/2})$ – Kelner-Mądry gives no improvement over simple random-walk algorithm
- No nice cut: any cut cuts either at least $\sqrt{n}$ edges or leaves regions of diameter at least $\sqrt{n}$



- Diameter: $\Theta(\sqrt{n})$
- Cover time: $\Theta(n^{3/2})$ – Kelner-Mądry gives no improvement over simple random-walk algorithm
- No nice cut: any cut cuts either at least $\sqrt{n}$ edges or leaves regions of diameter at least $\sqrt{n}$
- **Source of problem**: G_1 and G_2
  - are **far away from each other**
  - have large min-cut

Are G_1 and G_2 *really* far away from each other?

Are G_1 and G_2 *really* far away from each other?
We look at *distance* because it upper-bounds *cover time*:

Matthews bound

$\mathrm{cov}(G) \leqslant \widetilde{\mathcal{O}}(m \cdot \mathrm{diam}(G))$

Are G_1 and G_2 *really* far away from each other?
We look at *distance* because it upper-bounds *cover time*:

Matthews bound

$\mathrm{cov}(G) \leqslant \widetilde{\mathcal{O}}(m \cdot \mathrm{diam}(G))$

Key change: employ new notion of distance which captures cover time more tightly:

Are G_1 and G_2 *really* far away from each other?
We look at *distance* because it upper-bounds *cover time*:

Matthews bound

$\mathrm{cov}(G) \leqslant \widetilde{\mathcal{O}}(m \cdot \mathrm{diam}(G))$

Key change: employ new notion of distance which captures cover time more tightly: **effective resistance**.

Are G_1 and G_2 *really* far away from each other?
We look at *distance* because it upper-bounds *cover time*:

Matthews bound

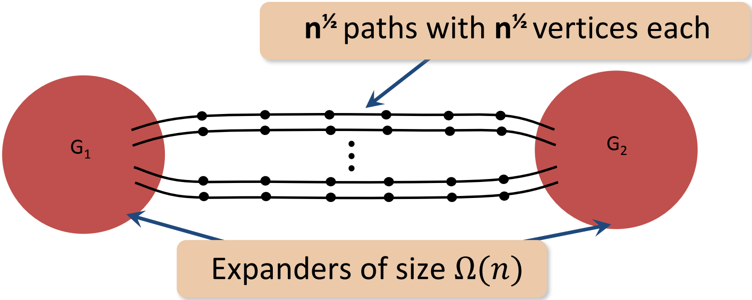
$\mathrm{cov}(G) \leqslant \widetilde{\mathcal{O}}(m \cdot \mathrm{diam}(G))$

Key change: employ new notion of distance which captures cover time more tightly: **effective resistance**.

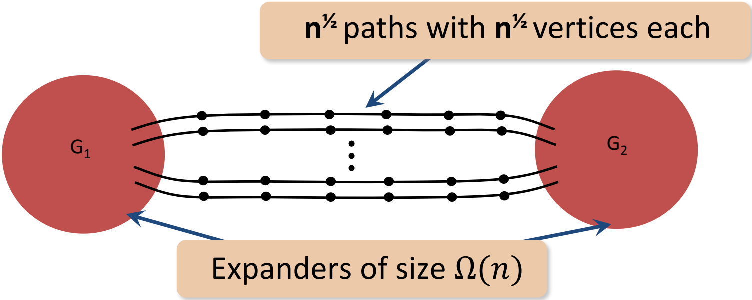
Tighter bound

$\mathrm{cov}(G) = \widetilde{\Theta}(m \cdot \mathrm{diam}_{\mathrm{eff}}(G)) \leqslant \widetilde{\mathcal{O}}(m \cdot \mathrm{diam}(G))$

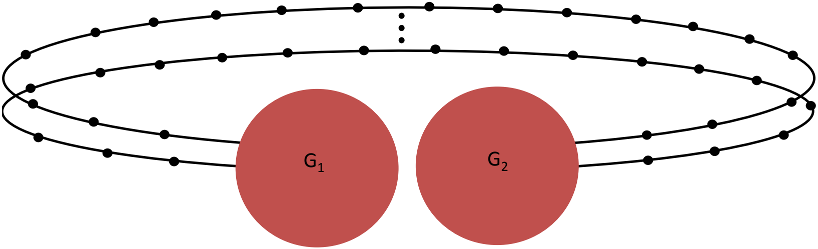
where $\mathrm{diam}_{\mathrm{eff}}(G) = \max_{s,t \in G} R_{\mathrm{eff}}(s,t)$.



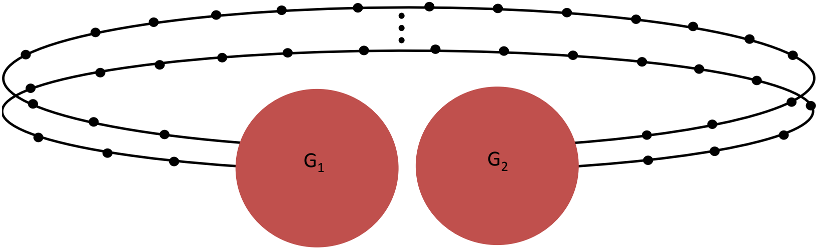
Before: G_1 and G_2 are **far away** in the graph-distance metric.



Now: G_1 and G_2 are **close** in the effective-resistance metric.
(They are connected by many paths.)



Now: G_1 and G_2 are **close** in the effective-resistance metric.
(They are connected by many paths.)



Now: G_1 and G_2 are **close** in the effective-resistance metric.
(They are connected by many paths.)

And we should treat $G_1 \cup G_2$ as one region!
The exterior of $G_1 \cup G_2$ is easy to partition nicely.

We obtained a nice region D:

- large
- low effective-resistance-diameter
- its exterior can be partitioned nicely

We obtained a nice region D:

- large
- low effective-resistance-diameter
- its exterior can be partitioned nicely

previously	now
diam(G) high	$\mathrm{diam}_{\mathrm{eff}}(D)$ low
cov(G) high	cov(D) low
stop the walk once G covered	stop the walk once D covered
slow	fast
learn T	only learn $T \cap D$
done in one shot	not done yet – but much progress

We obtained a nice region D:

- large
- low effective-resistance-diameter
- its exterior can be partitioned nicely

previously	now
diam(G) high	$\mathrm{diam}_{\mathrm{eff}}(D)$ low
cov(G) high	cov(D) low
stop the walk once G covered	stop the walk once D covered
slow	fast
learn T	only learn $T \cap D$
done in one shot	not done yet – but much progress

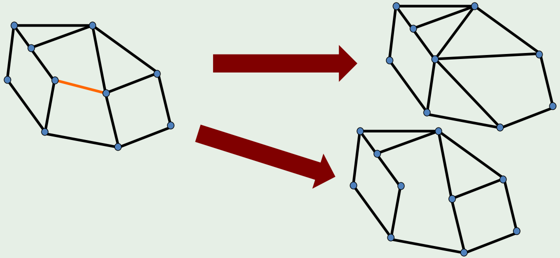
We learn $T \cap D$. How to use this knowledge?

We **condition** on the choice of $T \cap D$.

We **condition** on the choice of $T \cap D$.

Example

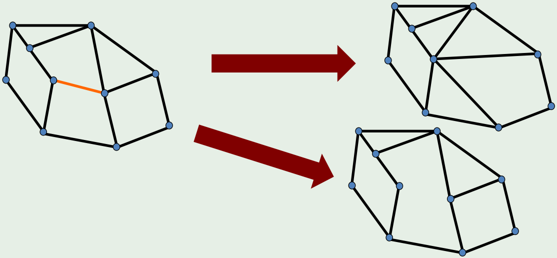
If $D = \{e\}$, then $T \cap D$ is either $\{e\}$ or $\emptyset$:



We **condition** on the choice of $T \cap D$.

Example

If $D = \{e\}$, then $T \cap D$ is either $\{e\}$ or $\emptyset$:



Our algorithm

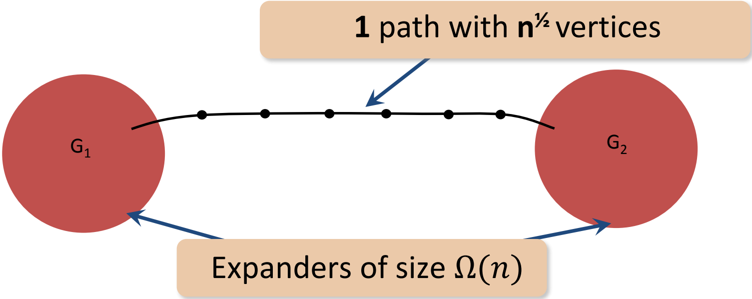
- Find nice region D.
- Sample $T \cap D$
  (run random walk with shortcutting until D is covered).
- Condition on this choice, and repeat.
  Interior of D (large) is eradicated – lots of progress!

Outstanding issue:

what if there is no such nice region D?

Outstanding issue:

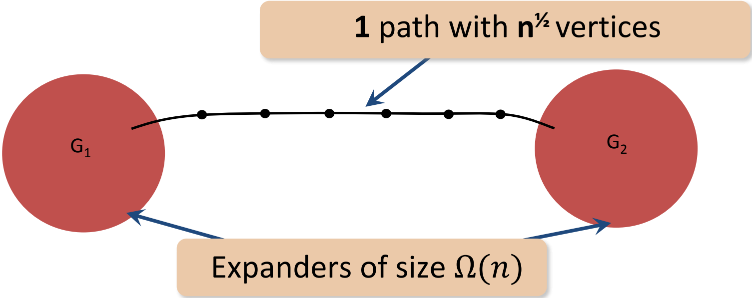
what if there is no such nice region D?



G_1 and G_2 are no longer close even in effective-resistance metric.

Outstanding issue:

what if there is no such nice region D?



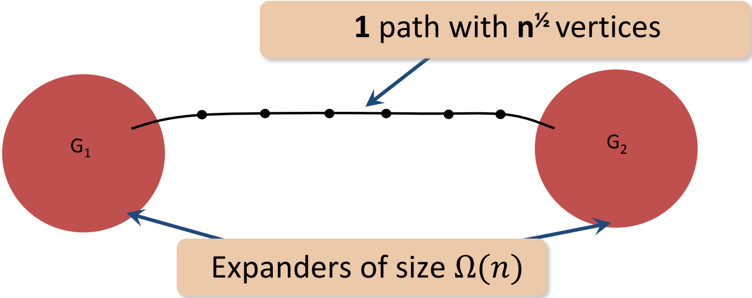
G_1 and G_2 are no longer close even in effective-resistance metric.

Can show: we can always either

- find a nice region D, or
- identify two large regions G_1 and G_2 which are far away in the effective-resistance metric.

Outstanding issue:

what if there is no such nice region D?



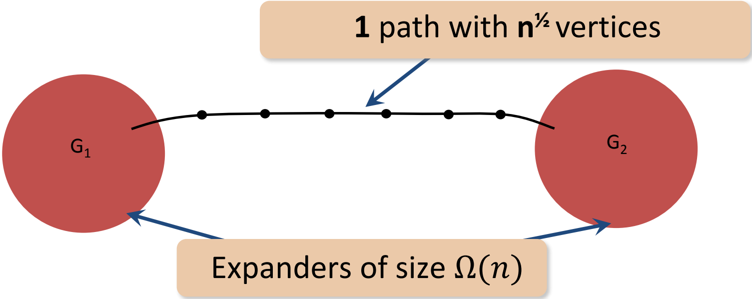
G_1 and G_2 are no longer close even in effective-resistance metric.

Can show: we can always either

- find a nice region D, or
- identify two large regions G_1 and G_2 which are far away in the effective-resistance metric. Then they have a small min-cut!

Outstanding issue:

what if there is no such nice region D?



G_1 and G_2 are no longer close even in effective-resistance metric.

Lemma (R_{eff} vs. Cuts)

If $R_{eff}(G_1, G_2) \geqslant m^{1/3}$, then $mincut(G_1, G_2) \leqslant m^{1/3}$.

Roughly speaking, we make this cut and recurse on both halves.

Theorem (Mądry, Straszak, T. 2015)

One can generate a uniformly random spanning tree in expected time $\widetilde{\mathcal{O}}(m^{4/3})$.

Open questions:

- for non-sparse graphs: improve $\widetilde{\mathcal{O}}(m^{4/3})$ to $\widetilde{\mathcal{O}}(mn^{1/3})$
  - (like Kelner-Mądry improve $\widetilde{\mathcal{O}}(m^{3/2})$ to $\widetilde{\mathcal{O}}(mn^{1/2})$)
  - would give a single algorithm best for all regimes of sparsity
  - seems to require fast approximation of vertex cuts
- faster algorithms
- other applications of:

Lemma (R_{eff} vs. Cuts)

$mincut(v_1, v_2) \leqslant \sqrt{\frac{m}{R_{eff}(v_1, v_2)}}$

Thank you!