Random Spanning Trees

Bhishmaraj S

Chennai Mathematical Institute

bhishma@cmi.ac.in

June 12, 2020

Overview

1. Introduction
   - Problem Definition
   - Applications
   - A bird's eye view of existing algorithms
2. Random Walk Based Algorithms
   - Aldous-Broder Algorithm
   - Wilson's Algorithm
3. Determinant Based Methods
   - Electric Networks
   - Harvey, Xu
4. Conclusion

Problem Definition

Given an undirected connected graph $G = (V, E)$, sample a spanning tree T with probability $\frac{1}{|\mathcal{T}|}$ where $\mathcal{T}$ denotes the set of all spanning trees of G.

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation

Applications

Sampling spanning trees pops up in surprising problems in TCS such as

- Constructing expanders ([GRV09], [FGRV14])
- Approximation algorithms for TSP([GSS11], [AGM$^+$17])
- Graph Sparsification ([FH10], [DK16])
- Analysis of network reliability ([Col87],[NC90], [CDM88])
- Sequence shuffling problem in Bioinformatics ([KMUW96])

Maze Generation [1]

[1] https://en.wikipedia.org/wiki/Maze_generation_algorithm

Proposed Algorithms

In this talk we would review some of the algorithms proposed for this problem and get into details of [HX16]

- **Random Walk Based** - Simulate variants of random walk on the input graph
- **Determinant Based** - Based on Kirchoff Matrix Tree theorem and involve computing determinants of the laplacian matrix
- **Approximation Algorithms**

Proposed Algorithms

In this talk we would review some of the algorithms proposed for this problem and get into details of [HX16]

- **Random Walk Based** - Simulate variants of random walk on the input graph
- **Determinant Based** - Based on Kirchoff Matrix Tree theorem and involve computing determinants of the laplacian matrix
- **Approximation Algorithms**

Proposed Algorithms

In this talk we would review some of the algorithms proposed for this problem and get into details of [HX16]

- **Random Walk Based** - Simulate variants of random walk on the input graph
- **Determinant Based** - Based on Kirchoff Matrix Tree theorem and involve computing determinants of the laplacian matrix
- **Approximation Algorithms**

Proposed Algorithms

In this talk we would review some of the algorithms proposed for this problem and get into details of [HX16]

- **Random Walk Based** - Simulate variants of random walk on the input graph
- **Determinant Based** - Based on Kirchoff Matrix Tree theorem and involve computing determinants of the laplacian matrix
- **Approximation Algorithms**

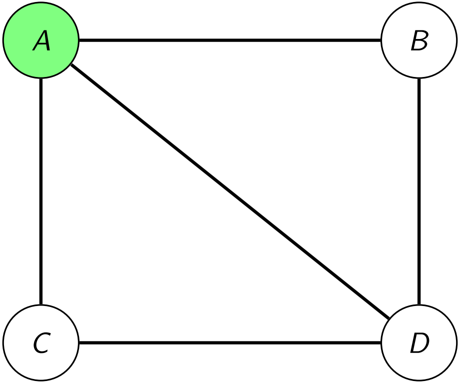
Aldous-Broder Algorithm

Andrei Broder and David Aldous independently invented the following algorithm

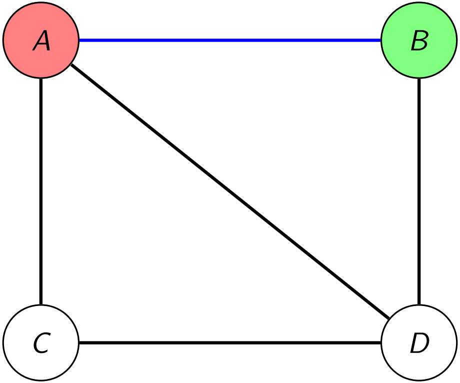
Input: $G = (V, E)$
Output: A random spanning tree
Choose a starting vertex s arbitrarily
$T_V \leftarrow \{s\}, T_E \leftarrow \emptyset$
while $|T_V| < |V$ **do**
    $next =_{u.a.r} N(s)$
    if $next \notin T_V$ **then**
        $T_V = T_V \cup \{next\}$
        $T_E = T_E \cup \{(s, next)\}$
    end
    $s = next$
end
return $T = (T_V, T_E)$

Algorithm 1: Aldous-Broder Algorithm

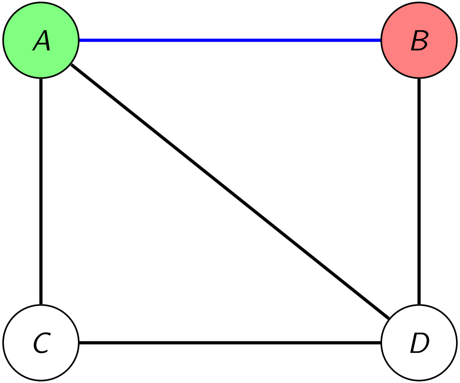
Aldous-Broder Algorithm example



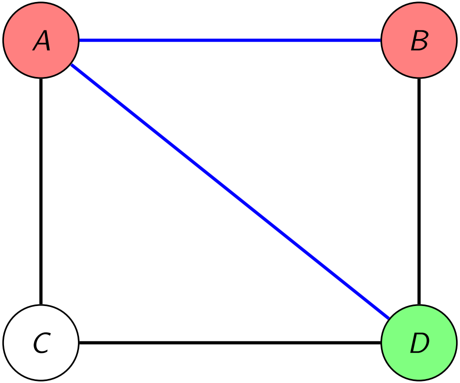
Aldous-Broder Algorithm example



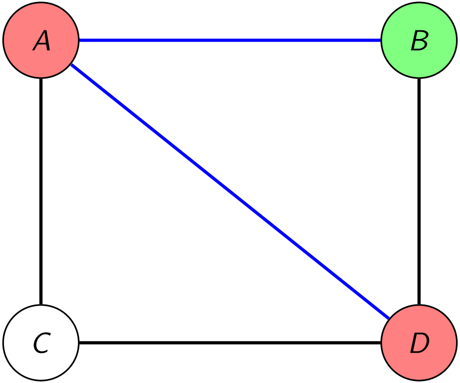
Aldous-Broder Algorithm example



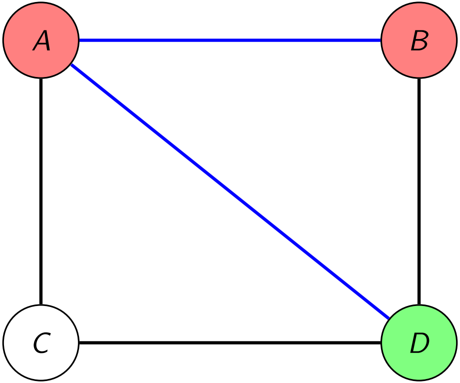
Aldous-Broder Algorithm example



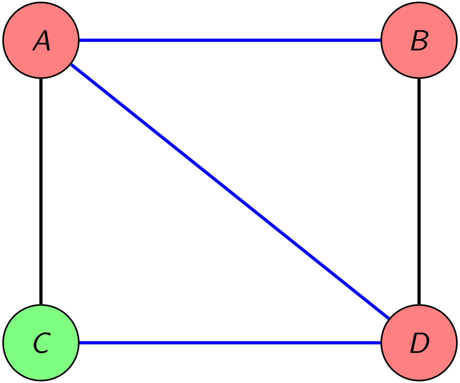
Aldous-Broder Algorithm example



Aldous-Broder Algorithm example



Aldous-Broder Algorithm example



Running Time

Cover Time

$cov_G(u) :=$ The expected number of steps for a random walk starting at u to visit all the vertices in G

Cover Time of G

$cov_G := \max_{u \in V_G} cov_G(u)$

It is known that $cov_G = \mathcal{O}(|V|\,|E|) = \mathcal{O}(|V|^3)$

Running Time

Cover Time

$cov_G(u) :=$ The expected number of steps for a random walk starting at u to visit all the vertices in G

Cover Time of G

$cov_G := \max_{u \in V_G} cov_G(u)$

It is known that $cov_G = \mathcal{O}(|V|\,|E|) = \mathcal{O}(|V|^3)$

Running Time

Cover Time
$cov_G(u) :=$ The expected number of steps for a random walk starting at u to visit all the vertices in G

Cover Time of G
$cov_G := \max_{u \in V_G} cov_G(u)$

It is known that $cov_G = \mathcal{O}(|V|\ |E|) = \mathcal{O}(|V|^3)$

Running Time

Cover Time
$cov_G(u) :=$ The expected number of steps for a random walk starting at u to visit all the vertices in G

Cover Time of G
$cov_G := \max_{u \in V_G} cov_G(u)$

It is known that $cov_G = \mathcal{O}(|V|\ |E|) = \mathcal{O}(|V|^3)$

Wilson's Algorithm

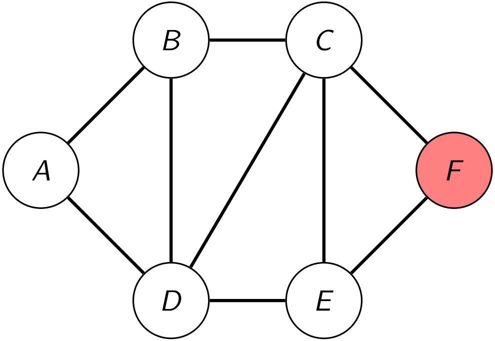
In 1996 Wilson proposed a variant of random walk called loop erased random walk .

Input: $G = (V, E)$ and root $r \in V$
Output: Parent pointer array called *next*

inTree$[i] \leftarrow$ *False*, $\forall\ i \neq r$
inTree$[r] \leftarrow$ *True*
next$[r] \leftarrow$ NULL
for $i \leftarrow 1$ *to* n **do**
  $u = i$
  while $\neg inTree[u]$ **do**
    next$[u] =_{u.a.r} N(u)$
    $u =$ next$[u]$
  end
  $u = i$
  while $\neg inTree[u]$ **do**
    inTree$[u] =$ *True*
    $u =$ next$[u]$
  end
end
return *next*

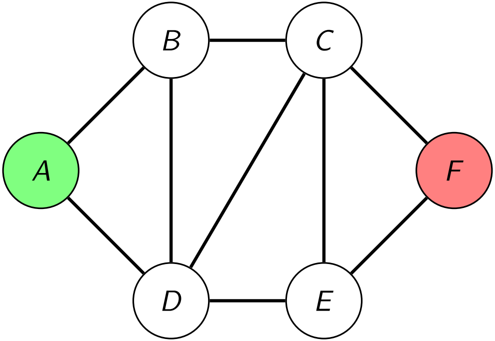
Algorithm 2: Wilson's Algorithm

Wilson's algorithm example

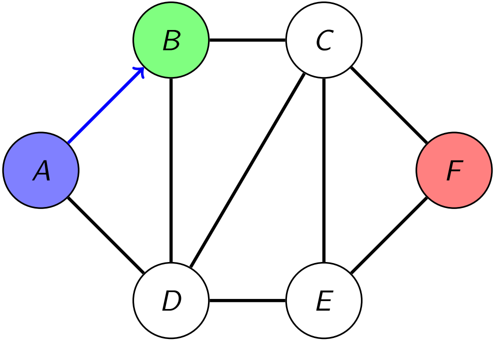


Wilson's algorithm example

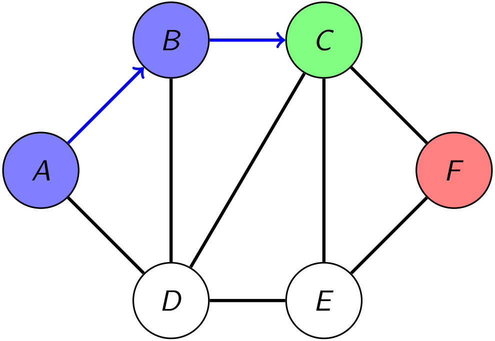
Start at A



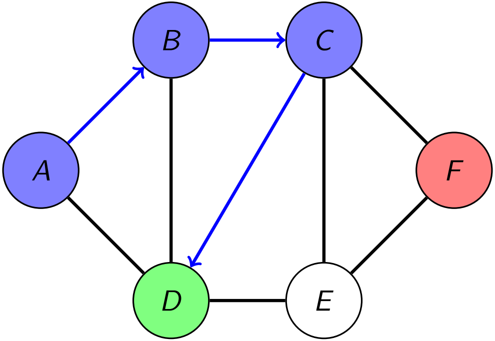
Wilson's algorithm example



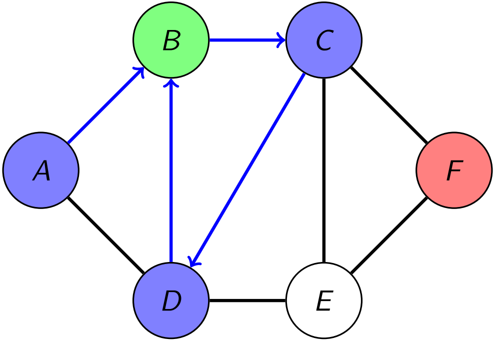
Wilson's algorithm example



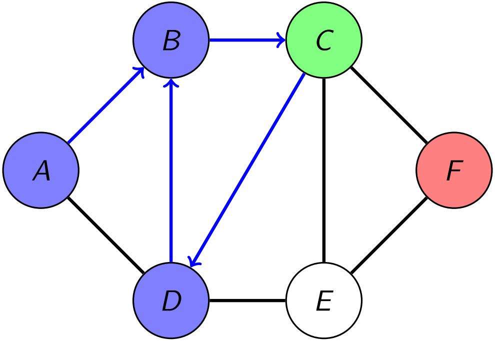
Wilson's algorithm example



Wilson's algorithm example

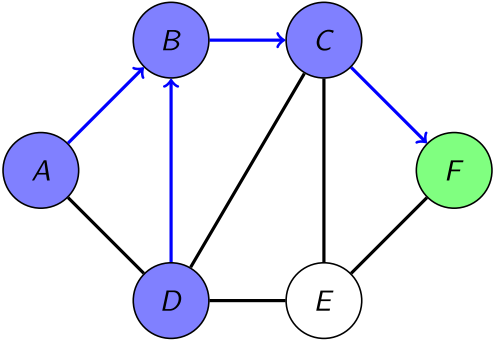


Wilson's algorithm example



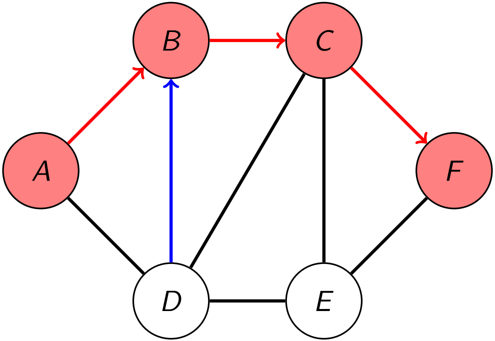
Wilson's algorithm example

Notice the **next(C)** has changed from D to F.



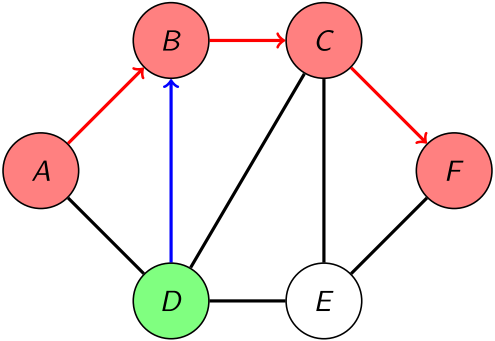
Wilson's algorithm example

Since a vertex already in the tree has been reached (namely F), starting from A we trace the successors and set their **inTree** value to *True*

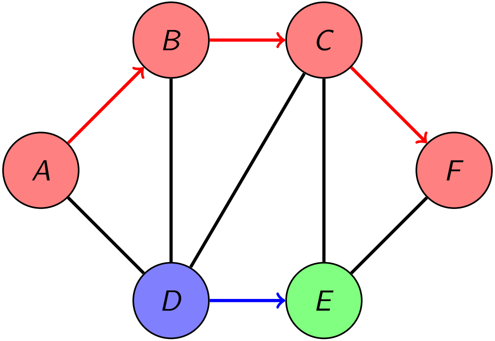


Wilson's algorithm example

Since B, C are already in the tree they will be skipped and now will start at D

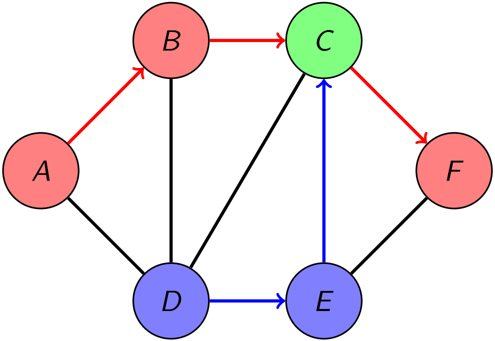


Wilson's algorithm example



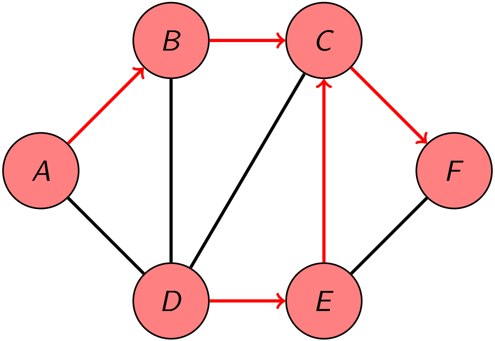
Wilson's algorithm example

Since C is already in the tree, the random walk stops and the algorithm retraces from D and includes the vertices into the tree



Wilson's algorithm example

Since C is already in the tree, the random walk stops and the algorithm retraces from D and includes the vertices into the tree



Laplacian of a graph

For an undirected unweighted graph $G = (V, E)$ the Laplacian L_G is a $|V| \times |V|$ matrix defined as

$$L_G(i,j) = \begin{cases} -1 & \text{if } (i,j) \in E \\ deg(i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

It can also be seen that

$$L_G = D - A$$

where D is a diagonal matrix with diagonal entries as degree of the corresponding vertex. And A the adjacency matrix of G.

Weighted Laplacian

For an undirected weighted graph $G = (V, E)$ and a weight function $\mathbf{w} : E \to \mathbb{R}_{\geq 0}$ the Laplacian L_G is a $|V| \times |V|$ matrix defined as

$$L_G(i,j) = \begin{cases} -w(i,j) & \text{if } (i,j) \in E \\ \sum\limits_{(i,v)\in E} w(i,v) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Determinant based algorithm

Theorem (Kirchoff Matrix Tree Theorem)

The number of spanning trees in a graph G is $\mathbf{det}(L_G[i])$ (for any i) where L_G denotes the Laplacian of G and $L_G[i]$ denotes the matrix with i^{th} row and column removed.

Lemma

$$\tau(G) = \tau(G \backslash e) + \tau(G/e)$$

Where $\tau(G)$ denotes the number of spanning trees of G and $G \backslash e, G/e$ are the graph G with the edge e deleted and contracted respectively

Determinant based algorithm

Theorem (Kirchoff Matrix Tree Theorem)

The number of spanning trees in a graph G is $\mathbf{det}(L_G[i])$ (for any i) where L_G denotes the Laplacian of G and $L_G[i]$ denotes the matrix with i^{th} row and column removed.

Lemma

$$\tau(G) = \tau(G \backslash e) + \tau(G/e)$$

Where $\tau(G)$ denotes the number of spanning trees of G and $G \backslash e, G/e$ are the graph G with the edge e deleted and contracted respectively

This can be used to give this simple algorithm. Proposed by [?]

Input: $G = (V, E)$
Output: Set of edges corresponding to a random spanning tree

for $e = (u, v) \in E$ **do**
    /* A' denotes matrix A with row and column 1 removed */
    if $(X \sim Bernoulli(\mathbf{det}(L'_{G/e})/\mathbf{det}(L'_G)) = 1$ **then**
        Add edge e to the spanning tree;
        $G = G/e$;
    else
        $G = G \setminus e$;
    end
end

Algorithm 3: Sampling uniform spanning tree using chain rule

$$\text{Running Time - } \mathcal{O}(|E| \cdot |V|^3)$$

Electric Network

Let $G = (V, E)$ be an undirected weighted graph with weight function $\mathbf{w} : E \to \mathbb{R}_{\geq 0}$. Now the electric network of G has edges replaced by a resistor with resistance $r_e = 1/\mathbf{w}(e), \forall e \in E$.

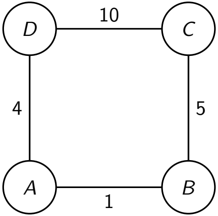
(a) The original graph G

(b) The electric network version of G

Figure: An example of a graph and it's corresponding electric network

Electric Network

Let $G = (V, E)$ be an undirected weighted graph with weight function $\mathbf{w} : E \to \mathbb{R}_{\geq 0}$. Now the electric network of G has edges replaced by a resistor with resistance $r_e = 1/\mathbf{w}(e), \forall e \in E$.



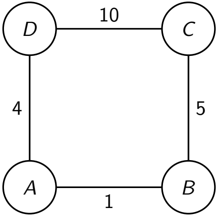
(a) The original graph G

(b) The electric network version of G

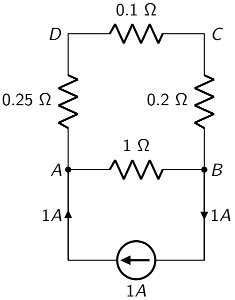
Figure: An example of a graph and it's corresponding electric network

Electric Network

Let $G = (V, E)$ be an undirected weighted graph with weight function $\mathbf{w} : E \to \mathbb{R}_{\geq 0}$. Now the electric network of G has edges replaced by a resistor with resistance $r_e = 1/\mathbf{w}(e), \forall e \in E$.

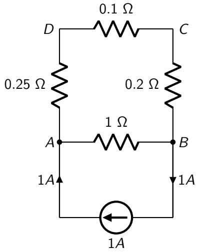


(a) The original graph G



(b) The electric network version of G

Figure: An example of a graph and it's corresponding electric network



By Ohm's law we have

$$i_{AD} = 4\left(v_A - v_D\right)$$
$$i_{DC} = 10\left(v_D - v_C\right)$$
$$i_{CB} = 5\left(v_C - v_B\right)$$
$$i_{AB} = 1\left(v_A - v_B\right)$$

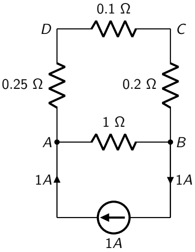
By Kirchoff's current law we have

$$i_{AD} + i_{AB} = 1$$
$$-i_{AD} + i_{DC} = 0$$
$$-i_{DC} + i_{CB} = 0$$
$$-i_{AB} - i_{CB} = -1$$

Now combining these two we get

$$5v_A - 1v_B - 0v_C - 1v_D = 1$$
$$-1v_A + 6v_B - 5v_C - 0v_D = -1$$
$$0v_A - 5v_B + 15v_C - 10v_D = 0$$
$$-4v_A - 0v_B - 10v_C + 14v_D = 0$$

The coefficients are exactly the Laplacian of G



By Ohm's law we have

$$i_{AD} = 4\,(v_A - v_D)$$
$$i_{DC} = 10\,(v_D - v_C)$$
$$i_{CB} = 5\,(v_C - v_B)$$
$$i_{AB} = 1\,(v_A - v_B)$$

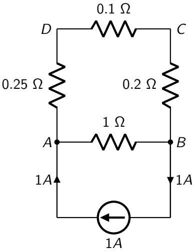
By Kirchoff's current law we have

$$i_{AD} + i_{AB} = 1$$
$$-i_{AD} + i_{DC} = 0$$
$$-i_{DC} + i_{CB} = 0$$
$$-i_{AB} - i_{CB} = -1$$

Now combining these two we get

$$5v_A - 1v_B - 0v_C - 1v_D = 1$$
$$-1v_A + 6v_B - 5v_C - 0v_D = -1$$
$$0v_A - 5v_B + 15v_C - 10v_D = 0$$
$$-4v_A - 0v_B - 10v_C + 14v_D = 0$$

The coefficients are exactly the Laplacian of G



By Ohm's law we have

$$
\begin{aligned}
i_{AD} &= 4\,(v_A - v_D)\\
i_{DC} &= 10\,(v_D - v_C)\\
i_{CB} &= 5\,(v_C - v_B)\\
i_{AB} &= 1\,(v_A - v_B)
\end{aligned}
$$

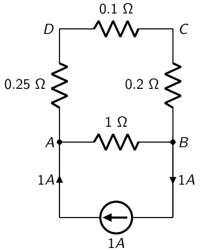
By Kirchoff's current law we have

$$
\begin{aligned}
i_{AD} + i_{AB} &= 1\\
-i_{AD} + i_{DC} &= 0\\
-i_{DC} + i_{CB} &= 0\\
-i_{AB} - i_{CB} &= -1
\end{aligned}
$$

Now combining these two we get

$$
\begin{aligned}
5v_A - 1v_B - 0v_C - 1v_D &= 1\\
-1v_A + 6v_B - 5v_C - 0v_D &= -1\\
0v_A - 5v_B + 15v_C - 10v_D &= 0\\
-4v_A - 0v_B - 10v_C + 14v_D &= 0
\end{aligned}
$$

The coefficients are exactly the Laplacian of G



By Ohm's law we have

$$i_{AD} = 4\,(v_A - v_D)$$
$$i_{DC} = 10\,(v_D - v_C)$$
$$i_{CB} = 5\,(v_C - v_B)$$
$$i_{AB} = 1\,(v_A - v_B)$$

By Kirchoff's current law we have

$$i_{AD} + i_{AB} = 1$$
$$-i_{AD} + i_{DC} = 0$$
$$-i_{DC} + i_{CB} = 0$$
$$-i_{AB} - i_{CB} = -1$$

Now combining these two we get

$$5v_A - 1v_B - 0v_C - 1v_D = 1$$
$$-1v_A + 6v_B - 5v_C - 0v_D = -1$$
$$0v_A - 5v_B + 15v_C - 10v_D = 0$$
$$-4v_A - 0v_B - 10v_C + 14v_D = 0$$

The coefficients are exactly the Laplacian of G

Let's make it formal

Let's focus only on unweighted graphs first.

Given an unweighted graph $G = (V, E)$ we associate a electrical network by replacing each edge with a resistor with **resistance** $1\ \Omega$.

A **current source** is introduced in each vertex, denoted as $\mathbf{c}_{\text{ext}} \in \mathbb{R}^n$.

This induces a **voltage** at each vertex and current at each edge. Let's denote it as $\mathbf{v} \in \mathbb{R}^n, \mathbf{i} \in \mathbb{R}^m$

Let's make it formal

Let's focus only on unweighted graphs first.

Given an unweighted graph $G = (V, E)$ we associate a electrical network by replacing each edge with a resistor with **resistance** $1\ \Omega$.

A **current source** is introduced in each vertex, denoted as $\mathbf{c}_{\text{ext}} \in \mathbb{R}^n$.

This induces a **voltage** at each vertex and current at each edge. Let's denote it as $\mathbf{v} \in \mathbb{R}^n, \mathbf{i} \in \mathbb{R}^m$

Let's make it formal

Let's focus only on unweighted graphs first.

Given an unweighted graph $G = (V, E)$ we associate a electrical network by replacing each edge with a resistor with **resistance** $1\ \Omega$.

A **current source** is introduced in each vertex, denoted as $\mathbf{c}_{\text{ext}} \in \mathbb{R}^n$.

This induces a **voltage** at each vertex and current at each edge. Let's denote it as $\mathbf{v} \in \mathbb{R}^n, \mathbf{i} \in \mathbb{R}^m$

Let's make it formal

Let's focus only on unweighted graphs first.

Given an unweighted graph $G = (V, E)$ we associate a electrical network by replacing each edge with a resistor with **resistance** $1\ \Omega$.

A **current source** is introduced in each vertex, denoted as $\mathbf{c}_{\text{ext}} \in \mathbb{R}^n$.

This induces a **voltage** at each vertex and current at each edge. Let's denote it as $\mathbf{v} \in \mathbb{R}^n, \mathbf{i} \in \mathbb{R}^m$

Let's make it formal

Incidence Matrix

Given an undirected unweighted graph $G = (V, E)$ with **arbitrary orientation** of edges. Let $B \in \mathbb{M}_{n \times m}$ called the edge-vertex **incidence matrix** defined as

$$B(i, e) = \begin{cases} 1 & \text{if } i \text{ is tail of } e \\ -1 & \text{if } i \text{ is head of } e \\ 0 & \text{otherwise} \end{cases}$$

Lemma

For a graph G with arbitrarily chosen incidence matrix B and Laplacian L,

$$B \cdot B^T = L$$

Let's make it formal

Incidence Matrix
Given an undirected unweighted graph $G = (V, E)$ with **arbitrary orientation** of edges. Let $B \in \mathbb{M}_{n \times m}$ called the edge-vertex **incidence matrix** defined as

$$B(i,e) = \begin{cases} 1 & \text{if } i \text{ is tail of } e \\ -1 & \text{if } i \text{ is head of } e \\ 0 & \text{otherwise} \end{cases}$$

Lemma

For a graph G with arbitrarily chosen incidence matrix B and Laplacian L,

$$B \cdot B^T = L$$

Kirchoff's current law

The algebraic sum of current into any vertex equals zero.

$$B \cdot \mathbf{i} = \mathbf{c}_{\text{ext}}$$

Ohm's law

$$V = I \cdot R$$

$$i_{xy} = \frac{v_x - v_y}{r_{xy}}$$

Since the resistance in the unweighted graph is 1 Ω we have

$$\mathbf{i} = B^T \cdot \mathbf{v}$$

Combining Ohm's law and Kirchoff's law we get

$$L \cdot \mathbf{v} = \mathbf{c}_{\text{ext}}$$

Pseudoinverse

Since $ker(L) = span(\mathbf{1})$

The equation $L \cdot x = b$ is not always solvable

Hence we work with a matrix called **pseudoinverse** (L^+) which is well defined for $ker(L)^\perp$

In the case $L \cdot \mathbf{v} = \mathbf{c}_{\text{ext}}$ there is a solution when $\langle \mathbf{c}_{\text{ext}}, \mathbf{1} \rangle = 0$

Effective Resistance

Definition - The potential difference across an edge $e = (x, y)$ when $1A$ is inducted at x and taken out at y. It is denoted as R_e^{eff}

So we have $\mathbf{c}_{\text{ext}} = e_x - e_y$ where e_i denotes a vector with 1 in the i^{th} index and 0 elsewhere.

$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot \mathbf{v}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot \mathbf{c}_{\text{ext}}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot (e_x - e_y)$$

Effective Resistance

Definition - The potential difference across an edge $e = (x, y)$ when $1A$ is inducted at x and taken out at y. It is denoted as R_e^{eff}

So we have $\mathbf{c}_{\text{ext}} = e_x - e_y$ where e_i denotes a vector with 1 in the i^{th} index and 0 elsewhere.

$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot \mathbf{v}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot \mathbf{c}_{\text{ext}}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot (e_x - e_y)$$

Effective Resistance

Definition - The potential difference across an edge $e = (x, y)$ when $1A$ is inducted at x and taken out at y. It is denoted as R_e^{eff}

So we have $\mathbf{c}_{\text{ext}} = e_x - e_y$ where e_i denotes a vector with 1 in the i^{th} index and 0 elsewhere.

$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot \mathbf{v}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot \mathbf{c}_{\text{ext}}$$
$$R_e^{\text{eff}} = (e_x - e_y)^T \cdot L^+ \cdot (e_x - e_y)$$

Electric Network and Spanning Tree

Theorem

Let T be a spanning tree chosen uniformly at random from all spanning trees in G. Then, the probability that an edge $e = (u, v)$ belongs to T is

$$\mathbb{P}[e \in T] = R_e^{eff} = (\chi_u - \chi_v)^T \; L_G^+ \; (\chi_u - \chi_v)$$

Naive algorithm using effective resistance

Input: $G = (V, E)$ and L_G^+
Output: Set of edges corresponding to a random spanning tree

```
for e = (u, v) ∈ E do
    R_e^eff = (χ_u − χ_v)^T L_G^+ (χ_u − χ_v);
    if (X ~ Bernoulli(R_e^eff)) = 1 then
        Add edge e to the spanning tree;
        G = G/e;
    else
        G = G \ e ;
    end
    Update L_G^+ ;
end
```

Algorithm 4: Using chain rule with effective resistance

Computing L_G^+ takes $\mathcal{O}(N^3)$ so running time is still $\mathcal{O}(|E| \cdot |V|^3)$

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

How to make it faster

Main bottleneck - Update for each edge takes $\mathcal{O}(N^3)$

Harvey, Xu [HX16] idea -

- Divide and conquer edges
- Update L_G^+ lazily
- Use Woodbury identity to compute the updated inverse faster
- Modify the identity to work for contraction and deletion of edges without changing the dimensions of L_G^+

Details

A update formula for deletion and contraction

Deletion

Given a set of edges D we need to find $L^{+}_{G\setminus D} = (L_G - L_D)^{+}$
L_D is the laplacian of the subgraph induced by D

Contraction

Bit tricky because the number of vertices changes

Details

A update formula for deletion and contraction

Deletion

Given a set of edges D we need to find $L^{+}_{G \setminus D} = (L_G - L_D)^{+}$
L_D is the laplacian of the subgraph induced by D

Contraction

Bit tricky because the number of vertices changes

Details

A update formula for deletion and contraction

Deletion

Given a set of edges D we need to find $L^+_{G\setminus D} = (L_G - L_D)^+$
L_D is the laplacian of the subgraph induced by D

Contraction

Bit tricky because the number of vertices changes

Dealing with contractions

Intuition

Contraction in a electrical network corresponds to short circuiting (i.e. 0 resistance).

Earlier we defined $r_e = 1/\mathbf{w}(e)$ hence resistance decreases as $\mathbf{w}(e) \to \infty$.

Laplacian pseudoinverse for contraction

Suppose F denotes the set of edges to be contracted we are interested in $\lim_{k\to\infty} (L_G + k \cdot L_F)^+$

Sherman-Morrison-Woodbury identity

Lemma

Let $M \in \mathbb{M}_{n\times n}, U \in \mathbb{M}_{n\times k}, V \in \mathbb{M}_{n\times k}$. *Assuming all the inverses are well defined*

$$(M + UV^T)^{-1} = M^{-1} - \left(M^{-1} \cdot U \cdot (I + V^T M^{-1} U)^{-1} \cdot V^T \cdot M^{-1}\right)$$

Complexity for computing $(M + UV^T)^{-1}$ reduces to $\mathcal{O}(k \cdot n^2)$

Using results from spectral graph theory they prove these formulas are well defined

Deletion Formula

$$(L_G - L_D)^+ = L_G^+ - \left(L_G^+ \cdot (L_D L_G^+ - I)^{-1} \cdot L_D \cdot L_G^+\right)$$

Update Formula (finite k)

$$(L_G + k\ L_F)^+ = L_G^+ - \left(L_G^+\ B_F\ (\frac{I}{k} + B_F^T\ L_G^+\ B_F)^{-1}\ B_F^T\ L_G^+\right)$$

Divide and conquer

Split the vertex set into 2 equal halves $V_G = S \uplus R$.

$S = S_1 \uplus S_2 \ R = R_1 \uplus R_2$.

$$E[S] = E[S_1] \cup E[S_2] \cup E[S_1, S_2]$$
$$E[R, S] = \bigcup_{i,j \in \{1,2\}} E[R_i, S_j]$$

Harvey, Xu Algorithm

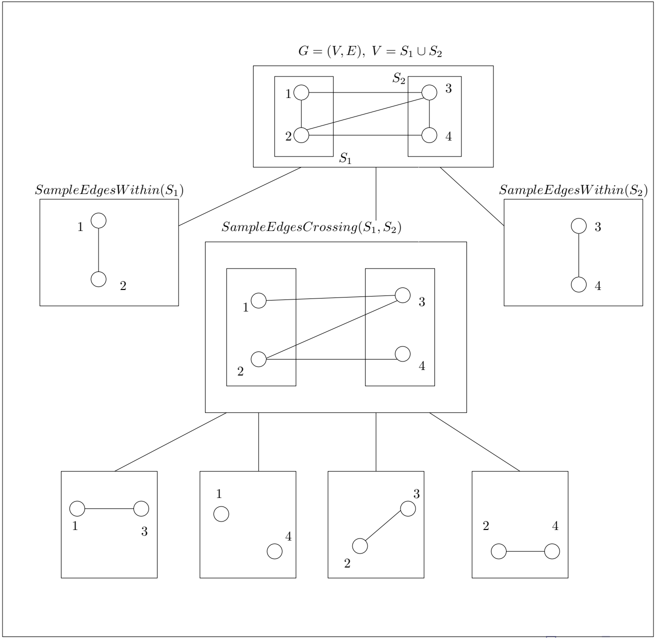
$N :=$ Current updated laplacian pseudoinverse (starts with L_G^+)

```
Function SampleEdgesWithin(S):
    if |S| > 1 then
        Divide S in half, S = S_1 ∪ S_2 for i ∈ {1,2} do
            D, F ← SampleEdgesWithin(S_i)
            N_{S,S} ← Update(S,F,D)
        end
    end
```

Algorithm 5: SampleEdgesWithin

```
Function SampleEdgesCrossing(R,S):
    if |S| == 1 then
        Try to sample S-R edge using N
    else
        for i ∈ {1,2}, j ∈ {1,2} do
            D, F ← SampleEdgesCrossing(R_i, S_j)
            N_{R∪S,R∪S} ← Update(R ∪ S, F, D)
        end
    end
```

Algorithm 6: SampleEdgesCrossing


$G = (V, E),\ V = S_1 \cup S_2$
S_2
S_1
1
2
3
4
$SampleEdgesWithin(S_1)$
$SampleEdgesCrossing(S_1, S_2)$
$SampleEdgesWithin(S_2)$

Running Time

$f(n) :=$ running time of *SampleEdgesWithin*(S)

$g(n) :=$ running time of *SampleEdgesCrossing*(S, R)

where $n = |R| = |S|$

$$g(n) = 4\ g(n/2) + \mathcal{O}(n^{\omega})$$
$$f(n) = 2\ f(n/2) + g(n) + \mathcal{O}(n^{\omega})$$

By master theorem the final running time is $\mathcal{O}(n^{\omega})$

Other algorithms

Determinant Based

- [?] uses a similar 6 way divide and conquer approach, but handles update using gaussian elimination. Runs in $\mathcal{O}(N^3)$
- [?] improves the previous algorithm to $\mathcal{O}(N^\omega)$ using a modification of LU decomposition.

Approximation Algorithms

- [?] built upon the Aldous-Broder algorithm by using the fast approximate laplacian solver of [?] to shortcut already visited regions

Initial Motivation and Future work

- Motivated by problem of efficiently sampling uniform spanning trees for dynamic graphs
- Recently [?] also used the same identity for showing that **REACHABILITY** problem is in $\mathtt{DynFO} + \mathtt{Mod}\ 2(\leq, +, \times)$. Hence we explored the possibility of using the same framework for sampling spanning trees in the dynamic setting.

Three Takeaways [2]

1. **Random Spanning Trees**
2. **Electric Networks**
3. **Sherman-Morrison-Woodbury Identity**

[2] https://math.stanford.edu/~vakil/threethings.html

Thank You

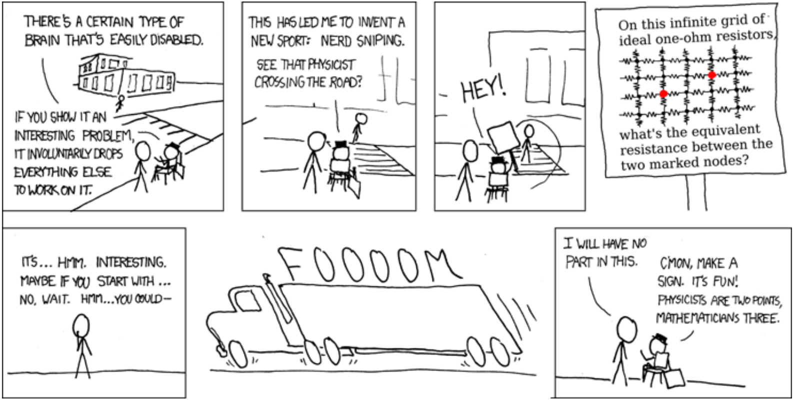


Figure: Obligatory xkcd [3]

[3] https://xkcd.com/356/

References I

Arash Asadpour, Michel X. Goemans, Aleksander Mdry, Shayan Oveis Gharan, and Amin Saberi, *An o(log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem*, Operations Research **65** (2017), no. 4, 1043–1061.

Charles J Colbourn, Bradley M Debroni, and Wendy J Myrvold, *Estimating the coefficients of the reliability polynomial*, Congressus Numerantium **62** (1988), 217–223.

Charles J Colbourn, Robert P.J Day, and Louis D Nel, *Unranking and ranking spanning trees of a graph*, Journal of Algorithms **10** (1989), no. 2, 271 – 286.

Charles J. Colbourn, Wendy J. Myrvold, and Eugene Neufeld, *Two algorithms for unranking arborescences*, Journal of Algorithms **20** (1996), no. 2, 268 – 281.

Charles J. Colbourn, *The combinatorics of network reliability*, Oxford University Press, Inc., USA, 1987.

Shlomi Dolev and Daniel Khankin, *Random spanning trees for expanders, sparsifiers, and virtual network security*, arXiv preprint arXiv:1612.02569 (2016).

Alan Frieze, Navin Goyal, Luis Rademacher, and Santosh Vempala, *Expanders via random spanning trees*, SIAM Journal on Computing **43** (2014), no. 2, 497–513.

Wai Shing Fung and Nicholas J. A. Harvey, *Graph sparsification by edge-connectivity and random spanning trees*, CoRR **abs/1005.0265** (2010).

Navin Goyal, Luis Rademacher, and Santosh Vempala, *Expanders via random spanning trees*, Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (USA), SODA 09, Society for Industrial and Applied Mathematics, 2009, p. 576585.

References II

S. O. Gharan, A. Saberi, and M. Singh, *A randomized rounding approach to the traveling salesman problem*, 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, 2011, pp. 550–559.

Nicholas JA Harvey and Keyulu Xu, *Generating random spanning trees via fast matrix multiplication*, LATIN 2016: Theoretical Informatics, Springer, 2016, pp. 522–535.

D. Kandel, Y. Matias, R. Unger, and P. Winkler, *Shuffling biological sequences*, Discrete Applied Mathematics **71** (1996), no. 1, 171 – 185.

V.G Kulkarni, *Generating random combinatorial objects*, Journal of Algorithms **11** (1990), no. 2, 185 – 207.

Louis D. Nel and Charles J. Colbourn, *Combining monte carlo estimates and bounds for network reliability*, Networks **20** (1990), no. 3, 277–298.