6.006- *Introduction to Algorithms*

Lecture 26

How Flipping Coins Helps Computation

Prof. Constantinos Daskalakis

Coin Flips in Algorithms

- Last time we gave an algorithm *SILVIO* for **primality testing**
- **Input:** Number *n* (represented by O(log n) bits)
- **Desired Behavior:** "PRIME" if *n* is prime, "COMPOSITE" o.w.
- *SILVIO* run in time poly(log *n*), i.e. polynomial in the representation of *n*.
- *SILVIO* flipped coins (namely somewhere in its execution it chose a random element in Z_n^*)
- ***SILVIO*'S Behavior:**
  - Pr[A(n)="PRIME"]=1, if *n* is prime
  - Pr[A(n)="PRIME"]≤1/2, if *n* is composite
- By repetition can boost the probability of outputting a correct answer as much as we want.
- Can *SILVIO* be derandomized? Unknown as of yet
- There *is* a primality testing algorithm that is deterministic.
- It was discovered many years later and is more complicated.
- **Moral:** Flipping coins enables simpler, and (potentially) faster computation.

Menu

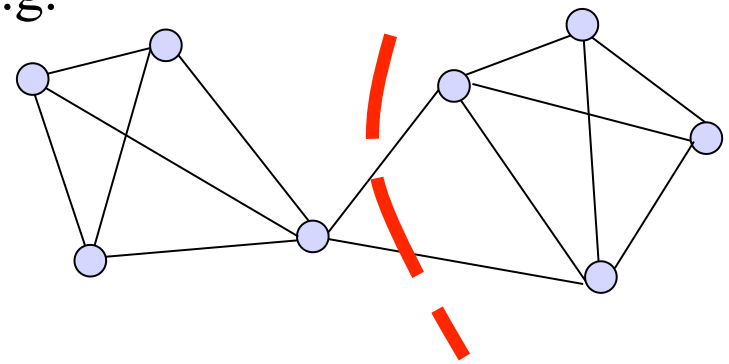
- Minimum-cut
- Random walks in graphs
  - Pagerank

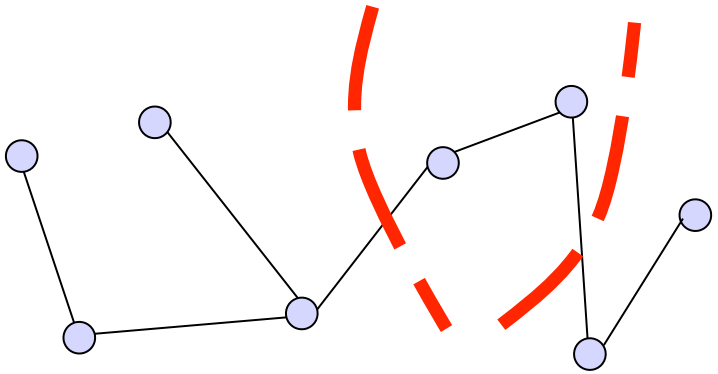
Menu

- **Minimum-cut**
- Random walks in graphs
  - Pagerank

MIN-CUT

- **Input:** Undirected connected graph G=(V,E).
- **Output:** Partition V into L and R minimizing the edges between L and R.
- i.e. find the **bottleneck** of a graph.
- E.g.



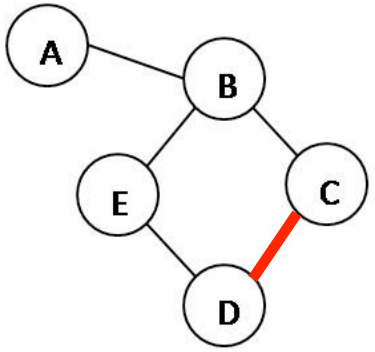


Any edge is a min-cut

- Best deterministic algorithm: $O(|V| |E| \log |V|^2/|E|)$.
- Fastest and simplest known algorithm: randomized; time $O(|V|^2 \log|V|)$
- Obtained by David Karger in 1993.
- **Intuition:** Minimum cut is (hopefully) a small set of edges.
- SO if I pick a random edge, chances are that it's not part of the minimum cut.

Karger's Algorithm

- Example execution:



- Pseudocode:

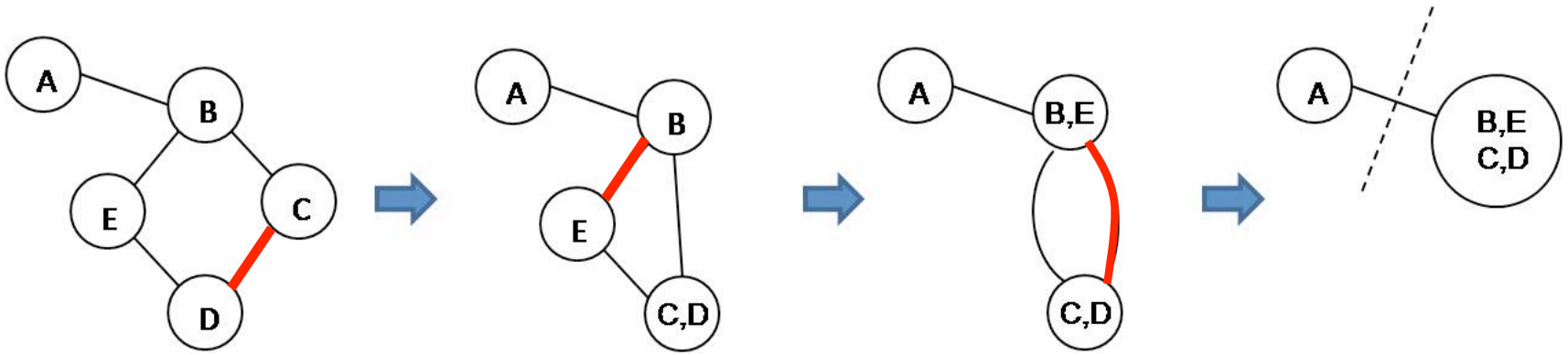
While more than two nodes remain:

- pick random edge $e = (u, v)$;
- merge u and v.
  *(called a **contraction**)*

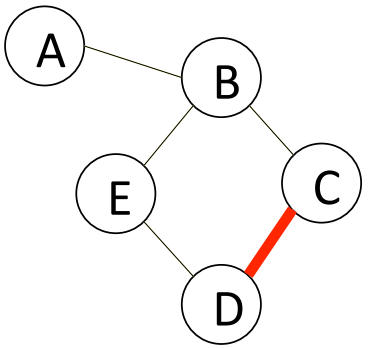
Output surviving edges.

Karger's Algorithm

- Good execution:



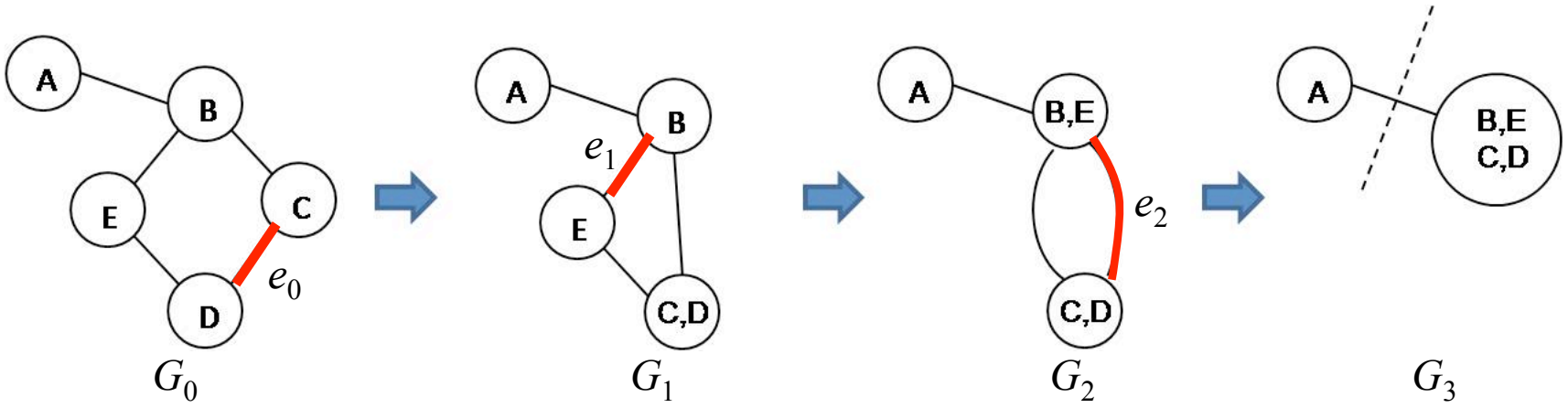
- Bad execution:



Claim: Pr[good execution] $\geq 2/n^2$ ➔ ~ n^2 repetitions suffice!

Karger's Algorithm

- Lower-bounding the probability of good execution.
- Graph may have many min-cuts (remember tree example).
- Let's fix one of them ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.

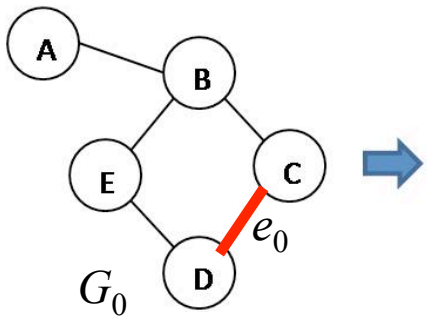


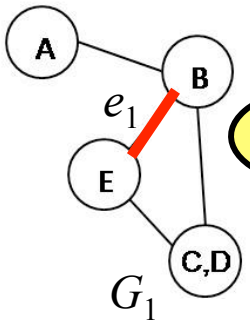
- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[none of chosen edges belongs to ***C***]

$$= \Pr[e_0 \notin C] \cdot \Pr[e_1 \notin C \mid e_0 \notin C] \cdot \ldots \cdot \Pr[e_{n-3} \notin C \mid e_0, \ldots, e_{n-4} \notin C]$$

Karger's Algorithm

- Let's fix a min-cut ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.





if the min-cut of a graph has size |*C*| then every vertex has degree ≥|*C*|

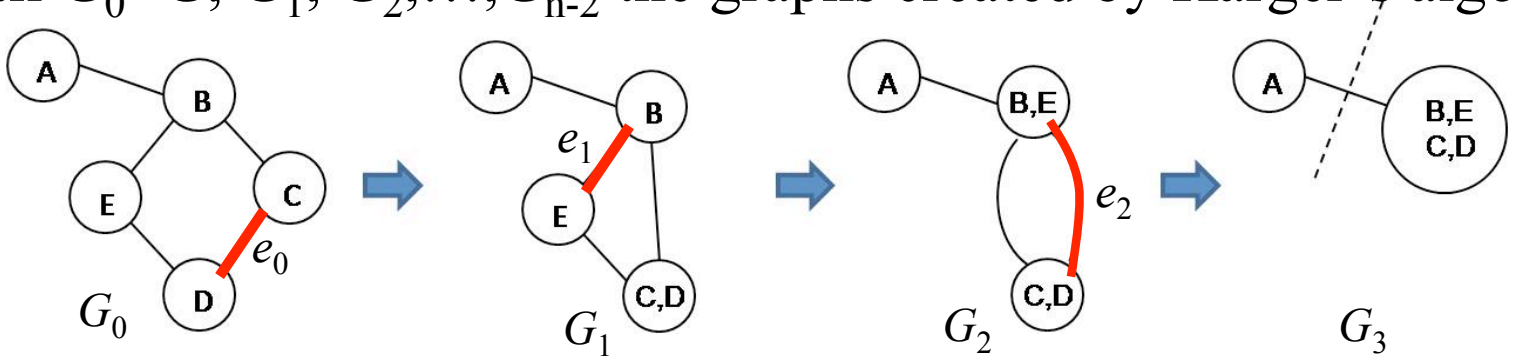
- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[none of chosen edges belongs to ***C***]

  = Pr[$e_0 \notin$ C] · Pr[$e_1 \notin$ C | $e_0 \notin$ C] ·…· Pr[$e_{n-3} \notin$ C | $e_0,\ldots,e_{n-4} \notin$ C]
- Warm-up: Pr[$e_0 \notin$ C]?

$$\Pr[e_0 \notin C] = \frac{|E| - |C|}{|E|} = 1 - \frac{|C|}{|E|} \geq 1 - \frac{|C|}{|V||C|/2} = 1 - \frac{2}{|V|}$$

Karger's Algorithm

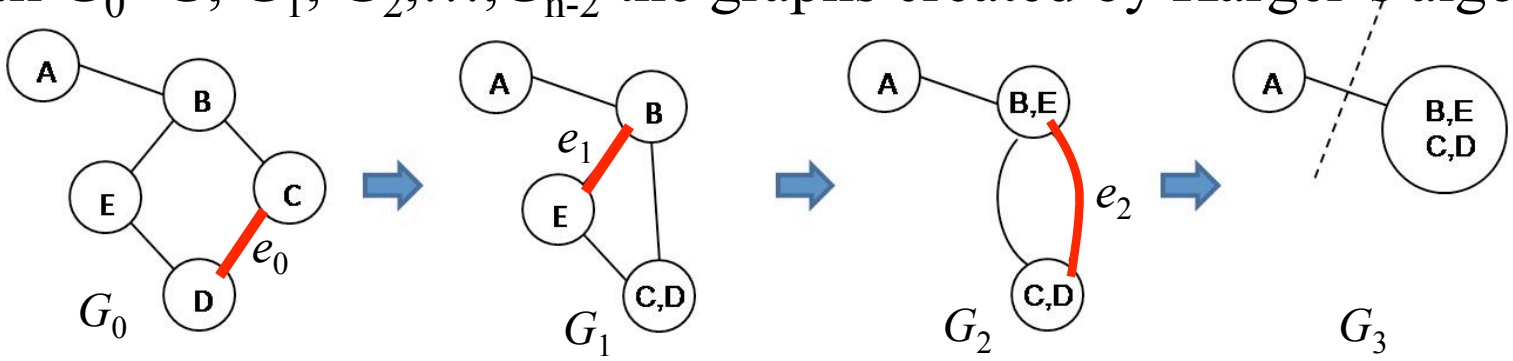
- Let's fix a min-cut ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.



- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[$e_0 \notin$ C] ·…·Pr[$e_{n-3} \notin$ C | e_0,…,$e_{n-4} \notin$ C]
- Warm-up: Pr[$e_0 \notin$ C] ≥ 1-2/|V|
- Pr[$e_i \notin$ C | e_0,…,$e_{i-1} \notin$ C] ?
- **Claim:** If e_0,…,$e_{i-1} \notin$ C, then the minimum cut of G_i has size |C|.
- **Proof:** All edges in C have survived. So min-cut at most size |C|.
- If there is a smaller cut in G_i, then that cut exists also in G_0.
- QED

Karger's Algorithm

- Let's fix a min-cut ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.

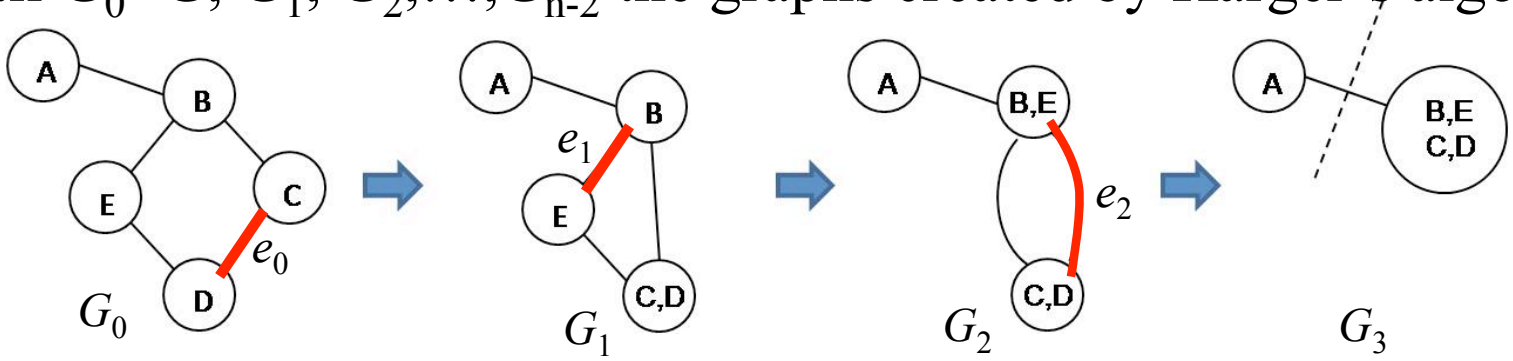


- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[$e_0 \notin$ C] ·…·Pr[$e_{n-3} \notin$ C | e_0,…,$e_{n-4} \notin$ C]
- Warm-up: Pr[$e_0 \notin$ C] ≥ 1-2/|V|
- Pr[$e_i \notin$ C | e_0,…,$e_{i-1} \notin$ C] ?
- **Claim:** If e_0,…,$e_{i-1} \notin$ C, then the minimum cut of G_i has size |C|.
- So:

$$\Pr[e_i \notin C \mid e_0 \notin C, \ldots, e_{i-1} \notin C] \geq 1 - \frac{2}{??}$$

Karger's Algorithm

- Let's fix a min-cut ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.

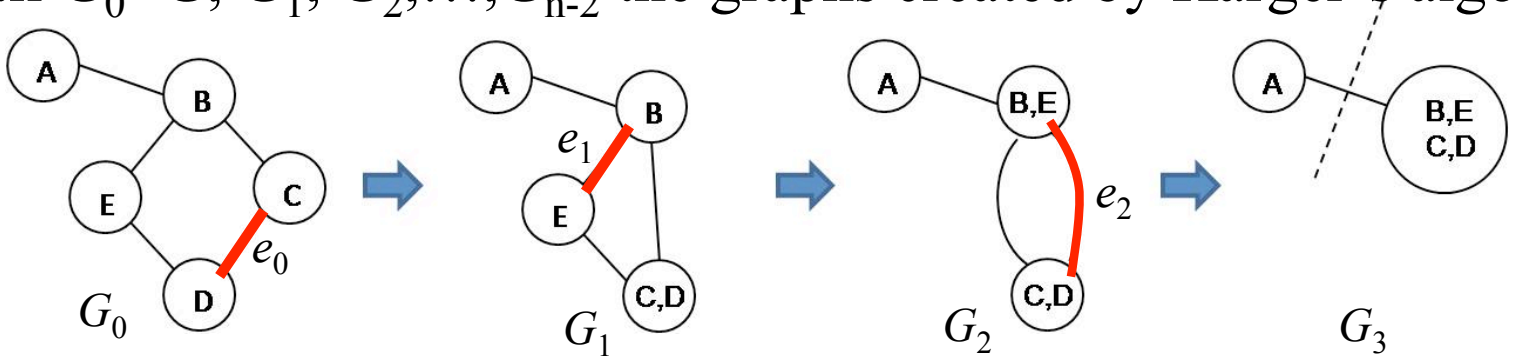


- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[$e_0 \notin$ C] ·…·Pr[$e_{n-3} \notin$ C | e_0,…,$e_{n-4} \notin$ C]
- Warm-up: Pr[$e_0 \notin$ C] ≥ 1-2/|V|
- Pr[$e_i \notin$ C | e_0,…,$e_{i-1} \notin$ C] ?
- **Claim:** If e_0,…,$e_{i-1} \notin$ C, then the minimum cut of G_i has size |C|.
- So:

$$\Pr[e_i \notin C \mid e_0 \notin C, \ldots, e_{i-1} \notin C] \geq 1 - \frac{2}{|V| - i}$$

Karger's Algorithm

- Let's fix a min-cut ***C***.
- Call G_0=G, G_1, G_2,…,G_{n-2} the graphs created by Karger's algorithm.



- Want to find probability that G_{n-2} only contains edges of ***C***.
- Pr[success] = Pr[$e_0 \notin$ C] ·…·Pr[$e_{n-3} \notin$ C | e_0,…,$e_{n-4} \notin$ C]
- So:

$$\Pr[e_i \notin C \mid e_0 \notin C, \ldots, e_{i-1} \notin C] \geq 1 - \frac{2}{|V| - i} = \frac{|V| - i - 2}{|V| - i}$$

- Hence:

$$\Pr[\text{success}] \geq \frac{|V|-2}{|V|} \cdot \frac{|V|-3}{|V|-1} \cdot \frac{|V|-4}{|V|-2} \cdot \frac{|V|-5}{|V|-3} \cdot \ldots \cdot \frac{4}{6} \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}$$

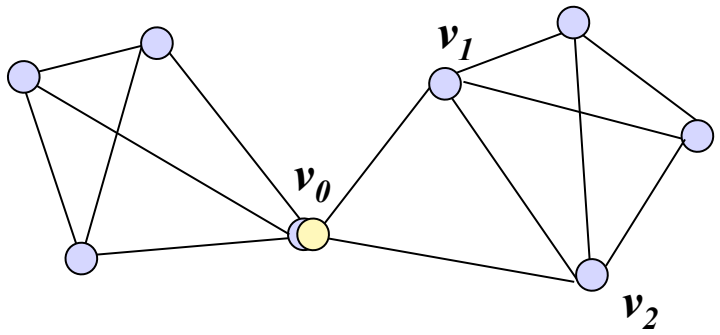
$$= \frac{2}{|V||V|-1} \geq 2/n^2$$

➔ repeat algorithm ~n^2 times and choose best cut

Menu

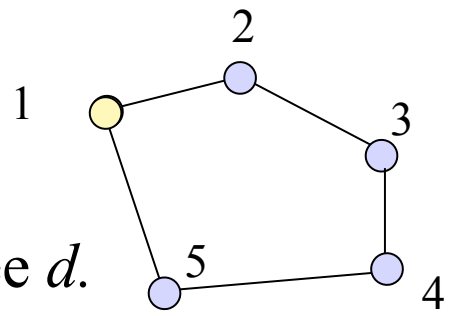
- Minimum-cut
- **Random walks in graphs**
  - Pagerank

Random Walks



- Given undirected graph $G = (V, E)$
- A squirrel stands at vertex v_0 :
- Squirrel ate fermented pumpkin so doesn't know what he's doing
- So jumps to random neighbor v_1 of v_0
- Then jumps to random neighbor v_2 of v_1
- etc
- Question: Where is squirrel after t steps?
- A: At some random location.
- OK, with what probability is squirrel at each vertex of the graph?
- Want to compute $x_t \in \mathrm{R}^n$, where
- $x_t(i)$: probability squirrel is at node i at time t.

$x_t \rightarrow x_{t+1}$?

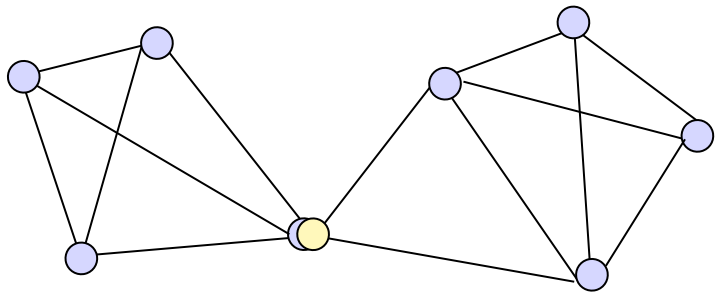


- Simplification: all nodes have same degree d.
- $x_0 = (1, 0, 0, 0, 0)$
- $x_0 \rightarrow x_1$?
- if $u_1, u_2, \ldots, u_d$ are the d neighbors of v_0, then
- $v_1 = u_i$ with probability $1/d$
- so $x_1 = (0, ½, 0, 0, ½)$
- $x_2 = (½, 0, ¼, ¼, 0)$
- …
- $A = \begin{pmatrix} 0 & ½ & 0 & 0 & ½ \\ ½ & 0 & ½ & 0 & 0 \\ 0 & ½ & 0 & ½ & 0 \\ 0 & 0 & ½ & 0 & ½ \\ ½ & 0 & 0 & ½ & 0 \end{pmatrix}$ (adjacency matrix divided by d)

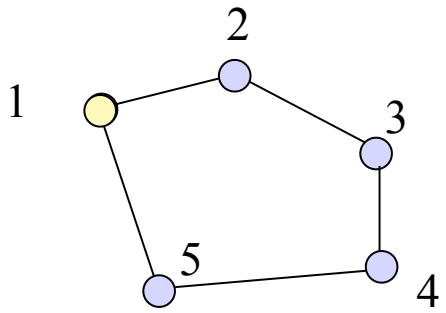
$$x_1 = x_0 A$$
$$x_2 = x_1 A = x_0 A^2$$
$$x_3 = x_2 A = x_0 A^3$$
$$\vdots$$
$$x_t = x_0 A^t$$

A_{ij} =probability of jumping to j if squirrel is at i

x_t



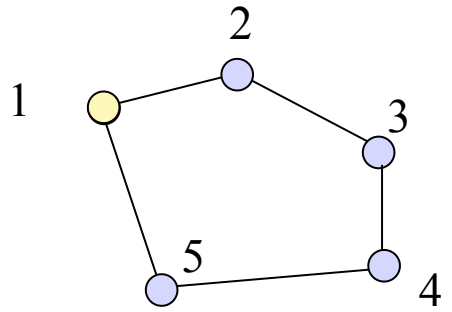
- More general undirected graphs?
- A =adjacency matrix where row i is divided by the degree d_i of i
- $x_t = x_0 A^t$
- Computing x_t ?
- Silvio will be disappointed if you don't use…
- repeated squaring!
- Compute $A \rightarrow A^2 \rightarrow A^4 \rightarrow \ldots \rightarrow A^t$ (if t is a power of 2; if not …)
- then do vector-matrix product
- How about limiting distribution x_t as $t \rightarrow \infty$?
- e.g. what is x_∞ in 5-cycle?
- $x_\infty = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5})$



Verifying $x_t \rightarrow (⅕, ⅕, ⅕, ⅕, ⅕)$

- Recall

$$A = \begin{pmatrix} 0 & ½ & 0 & 0 & ½ \\ ½ & 0 & ½ & 0 & 0 \\ 0 & ½ & 0 & ½ & 0 \\ 0 & 0 & ½ & 0 & ½ \\ ½ & 0 & 0 & ½ & 0 \end{pmatrix}$$



- x_0 = [1 0 0 0 0]
- x_1 = [0 0.5000 0 0 0.5000]
- x_2 = [0.5000 0 0.2500 0.2500 0]
- x_3 = [0 0.3750 0.1250 0.1250 0.3750]
- x_4 = [0.3750 0.0625 0.2500 0.2500 0.0625]
- x_5 = [0.0625 0.3125 0.1562 0.1562 0.3125]
- x_6 = [0.3125 0.1094 0.2344 0.2344 0.1094]
- x_7 = [0.1094 0.2734 0.1719 0.1719 0.2734]
- x_8 = [0.2734 0.1406 0.2227 0.2227 0.1406]
- x_9 = [0.1406 0.2480 0.1816 0.1816 0.2480]
- x_{10} =[0.2480 0.1611 0.2148 0.2148 0.1611]
- x_{11} =[0.1611 0.2314 0.1880 0.1880 0.2314]
- x_{12} =[0.2314 0.1746 0.2097 0.2097 0.1746]
- x_{13} =[0.1746 0.2206 0.1921 0.1921 0.2206]
- x_{14} =[0.2206 0.1833 0.2064 0.2064 0.1833]

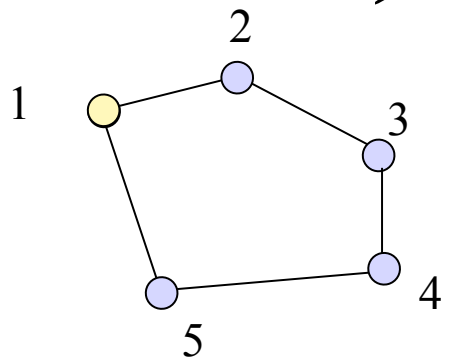
x_{15} = [0.1833 0.2135 0.1949 0.1949 0.2135]
x_{16} = [0.2135 0.1891 0.2042 0.2042 0.1891]
x_{17} = [0.1891 0.2088 0.1966 0.1966 0.2088]
x_{18} = [0.2088 0.1929 0.2027 0.2027 0.1929]
x_{19} = [0.1929 0.2058 0.1978 0.1978 0.2058]
x_{20} = [0.2058 0.1953 0.2018 0.2018 0.1953]
x_{21} = [0.1953 0.2038 0.1986 0.1986 0.2038]
x_{22} = [0.2038 0.1969 0.2012 0.2012 0.1969]
x_{23} = [0.1969 0.2025 0.1991 0.1991 0.2025]
x_{24} = [0.2025 0.1980 0.2008 0.2008 0.1980]
x_{25} = [0.1980 0.2016 0.1994 0.1994 0.2016]

$$\boxed{x_t = x_0 A^t}$$

Proving $x_t \to (⅕, ⅕, ⅕, ⅕, ⅕)$?

- Recall

$$A = \begin{pmatrix} 0 & ½ & 0 & 0 & ½ \\ ½ & 0 & ½ & 0 & 0 \\ 0 & ½ & 0 & ½ & 0 \\ 0 & 0 & ½ & 0 & ½ \\ ½ & 0 & 0 & ½ & 0 \end{pmatrix}$$

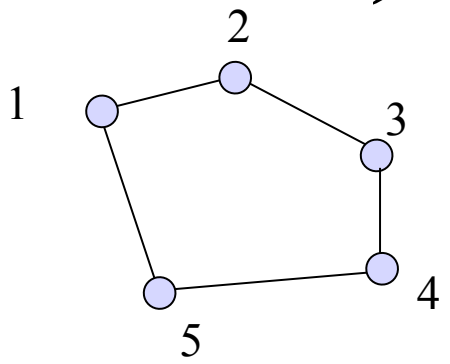


- Random idea: what are the eigenvalues of A ?
- A symmetric so 5 real eigenvalues
- $\lambda_1 = 1.0000$, $\lambda_2 = \lambda_3 = 0.3090$, $\lambda_4 = \lambda_5 = -0.8090$ (thanks Matlab)
- coincidence: $\lambda_2 = \lambda_3$ and $\lambda_4 = \lambda_5$ (5-cylce is a special graph)
- non-coincidence (holds for any undirected graph*):
  - largest eigenvalue =1
  - all others have absolute value <1
- left eigenvector corresponding to $\lambda_1 = 1.0000$?
- $e_1 = (⅕, ⅕, ⅕, ⅕, ⅕)$ is a left eigenvector for λ_1
- Wow. Why would $x_t \to e_1$ as $t \to \infty$?

Proving $x_t \rightarrow (⅕, ⅕, ⅕, ⅕, ⅕)$?

- Recall

$$A = \begin{pmatrix} 0 & ½ & 0 & 0 & ½ \\ ½ & 0 & ½ & 0 & 0 \\ 0 & ½ & 0 & ½ & 0 \\ 0 & 0 & ½ & 0 & ½ \\ ½ & 0 & 0 & ½ & 0 \end{pmatrix}$$



- $\lambda_1 = 1.0000,\ \lambda_2 = \lambda_3 = 0.3090,\ \lambda_4 = \lambda_5 = -0.8090$
- $e_1 = (⅕, ⅕, ⅕, ⅕, ⅕)$
- Proof: choose e_2, e_3, e_4, e_5 so that eigenvectors form a basis
- (guaranteed by the spectral theorem since A is symmetric)
- so $x_0 = a_1 e_1 + a_2 e_2 + a_3 e_3 + a_4 e_4 + a_5 e_5$, for some a_1, a_2, a_3, a_4, a_5
- Now $x_t = x_0 A^t =$

$$= a_1 e_1 A^t + a_2 e_2 A^t + a_3 e_3 A^t + a_4 e_4 A^t + a_5 e_5 A^t$$
$$= a_1 e_1 \lambda_1^t + a_2 e_2 \lambda_2^t + a_3 e_3 \lambda_3^t + a_4 e_4 \lambda_4^t + a_5 e_5 \lambda_5^t$$
$$\rightarrow a_1 e_1, \text{ as } t \rightarrow \infty$$

- since $e_1 = (⅕, ⅕, ⅕, ⅕, ⅕)$ is a distribution, it must be that $a_1 = 1$
- Hence $x_t \rightarrow (⅕, ⅕, ⅕, ⅕, ⅕)$, as $t \rightarrow \infty$

More General Theorem

- Given directed graph G
- Take A = adjacency matrix where row i is divided by the **out-**degree d_i of i
- (Under mild conditions*) A has eigenvalue 1 with multiplicity 1 and all other eigenvalues will have absolute value <1
- Moreover, if e_1 be the (unique) left eigenvector corresponding to eigenvalue 1,
- then e_1 will have all components positive.
- Normalize it so that it is a distribution.
- **Theorem:** A random walk on G **started anywhere** will converge to distribution e_1!
- e_1 is called the "**stationary distribution of *G***"

- (Fundamental Theorem of Markov Chains)

- Two obvious Questions:
  - why is x_∞ interesting?
  - how fast does $x_t \rightarrow x_\infty$?

Menu

- Minimum-cut
- Random walks in graphs
  - **Pagerank**

Pagerank

- No better proof that something is useful than having interesting applications ☺
- It turns out that random walks have a famous one: PageRank.
- PageRank of a webpage $p \approx^*$ Probability that a web-surfer starting from some central page (e.g. Yahoo!) and following random weblinks arrives at webpage p in infinite steps.
- How compute this probability?
- Form graph G = the hyperlink graph;
- Namely, G has a node for every webpage, and there is an edge from webpage p_1 to webpage p_2 *iff* there is a hyperlink from p_1 to p_2.
- Compute stationary distribution of G, i.e. the left eigenvector of the (normalized by out-degrees) adjacency matrix A of G, corresponding to eigenvalue 1.
- How compute stationary distribution?
- Idea 1: Crawl the web, create giant A, solve eigenvalue problem.
- Runtime $O(n^3)$ using Gaussian elimination
- too much for n = size of the web

Pagerank

- Graph G = the hyperlink graph
- Compute stationary distribution of G, i.e. the left eigenvector of the (normalized by out-degrees) adjacency matrix A of G, corresponding to eigenvalue 1.
- How compute stationary distribution?
- Different (better?) idea:
- Forget linear algebra;
- Start at some central page and do random walk for a few steps (how many?);
- Restart and repeat (how many times?);
- then take PageRank(p) ≈ empirical probability that random walk ended at p.
- If web-graph is well-connected*, hope that empirical distribution should be good approximation to stationary distribution for the right choice of "how many" above…
- or at least for the top components of the eigenvector, which are the most important for ranking the top results.
- *caveat: In reality, Pagerank corresponds to the stationary distribution of a random surfer who does the following at every step: with probability 15% jumps to a random page (called a restart), &with probability 85% jumps to a random neighbor.
- Same theory applies.

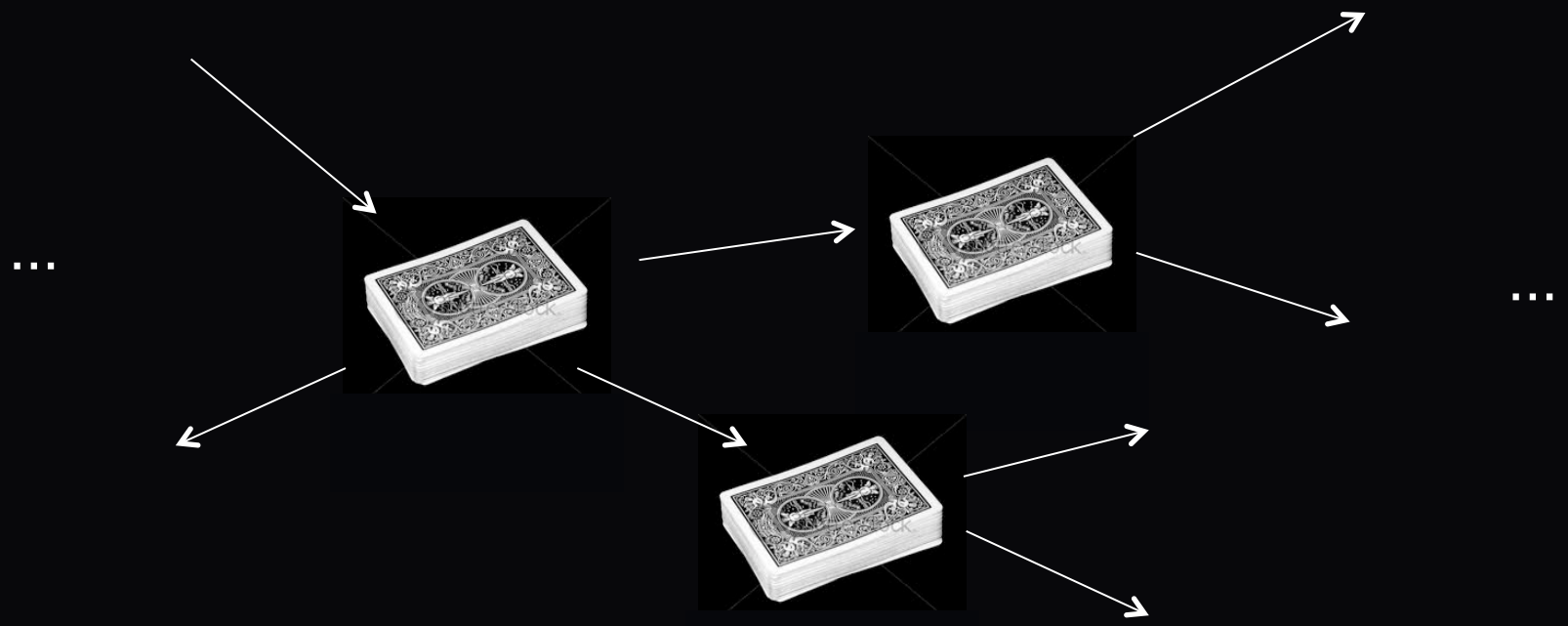
Menu

- Minimum-cut
- Random walks in graphs
  - Pagerank
  - **How fast does $x_t \rightarrow x_\infty$?**

“Mixing Time”

- Captures the speed at which $x_t \rightarrow x_\infty$
- Speed depends on connectivity of G.
- Sometimes G is given to us and we can’t change it.
- But sometimes we design G .
- e.g. in card shuffling
- type of shuffle defines connectivity of the graph between deck configurations…

Card Shuffling Graph



" $\longrightarrow$ " : reachable via a particular move defined by shuffle

while performing the shuffle we jump from node to node of this graph

stationary distribution of a correct shuffle?

probability 1/52! on each permutation

Effect of Shuffle to Mixing Time

Different shuffles have different mixing times. Examples:

- Top-in-at-Random: *take top card and stick it to random location*

Number of repetitions to be close to uniform permutation?

~300 repetitions

- Riffle Shuffle:

Number of repetitions? ~10

So different shuffles have different graphs with different mixing times.

Summary

Randomness is useful

As are the other techniques we saw in this class

When facing an algorithmic problem:

- understand it
- try brute force first
- then try to improve it using:
    - a cool data structure such as an AVL tree/heap/hash table
    - a cool algorithmic technique such as Divide and Conquer, or DP
    - map it to a graph problem and use off the shelf algorithm such as BFS/DFS/Dijkstra/Bellman-Form/Topological Sort
    - or modify these algorithms
- If everything else fails, maybe NP-hard? Try to reduce an NP-hard problem to your problem.
- Look at a catalog of NP-hard problems, find a similar problem to your problem and try to reduce that problem to your problem.
- Great hanging out every Tuesday and Thursday
- Evaluate class: http://web.mit.edu/subjectevaluation/evaluate.html