Chapter 6
Randomization

Algorithm Theory
WS 2013/14

Fabian Kuhn

Randomization

Randomized Algorithm:

- An algorithm that uses (or can use) random coin flips in order to make decisions

We will see: randomization can be a powerful tool to

- Make algorithms faster
- Make algorithms simpler
- Make the analysis simpler
  - Sometimes it's also the opposite…
- Allow to solve problems (efficiently) that cannot be solved (efficiently) without randomization
  - True in some computational models (e.g., for distributed algorithms)
  - Not clear in the standard sequential model

Contention Resolution

A simple starter example (from distributed computing)

- Allows to introduce important concepts
- … and to repeat some basic probability theory

Setting:

- n processes, 1 resource
  (e.g., shared database, communication channel, …)
- There are time slots $1,2,3, \ldots$
- In each time slot, only one client can access the resource
- All clients need to regularly access the resource
- If client i tries to access the resource in slot t:
  - Successful iff no other client tries to access the resource in slot t

Algorithm

Algorithm Ideas:

- Accessing the resource deterministically seems hard
  - need to make sure that processes access the resource at different times
  - or at least: often only a single process tries to access the resource
- **Randomized solution:**
  In each time slot, each process tries with probability p.

Analysis:

- How large should p be?
- How long does it take until some process i succeeds?
- How long does it take until all processes succeed?
- What are the probabilistic guarantees?

Analysis

Events:

- $\boldsymbol{\mathcal{A}_{i,t}}$: process i tries to access the resource in time slot t
  - Complementary event: $\overline{\mathcal{A}_{i,t}}$

$$\mathbb{P}(\mathcal{A}_{i,t}) = p, \qquad \mathbb{P}(\overline{\mathcal{A}_{i,t}}) = 1 - p$$

- $\boldsymbol{\mathcal{S}_{i,t}}$: process i is successful in time slot t

$$\mathcal{S}_{i,t} = \mathcal{A}_{i,t} \cap \left(\bigcap_{j \neq i} \overline{\mathcal{A}_{j,t}} \right)$$

- **Success probability** (for process i)**:**

Fixing p

- $\mathbb{P}(\mathcal{S}_{i,t}) = p(1-p)^{n-1}$ is maximized for

$$\boldsymbol{p = \frac{1}{n}} \quad \Longrightarrow \quad \mathbb{P}(\mathcal{S}_{i,t}) = \frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}.$$

- **Asymptotics:**

$$\text{For } n \geq 2: \quad \frac{1}{4} \leq \left(1-\frac{1}{n}\right)^{n} < \frac{1}{e} < \left(1-\frac{1}{n}\right)^{n-1} \leq \frac{1}{2}$$

- **Success probability:**

$$\boldsymbol{\frac{1}{en} < \mathbb{P}(\mathcal{S}_{i,t}) \leq \frac{1}{2n}}$$

Time Until First Success

Random Variable T_i:

- $T_i = t$ if proc. i is successful in slot t for the first time

- **Distribution:**

- T_i is geometrically distributed with parameter

$$q = \mathbb{P}(\mathcal{S}_{i,t}) = \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} > \frac{1}{en}.$$

- **Expected time** until first success:

$$\mathbb{E}[T_i] = \frac{1}{q} < en$$

Time Until First Success

Failure Event $\mathcal{F}_{i,t}$: Process i does not succeed in time slots $1, \dots, t$

- The events $\mathcal{S}_{i,t}$ are independent for different t:

$$\mathbb{P}(\mathcal{F}_{i,t}) = \mathbb{P}\left(\bigcap_{r=1}^{t} \overline{\mathcal{S}_{i,r}}\right) = \prod_{r=1}^{t} \mathbb{P}\left(\overline{\mathcal{S}_{i,r}}\right) = \left(1 - \mathbb{P}(\mathcal{S}_{i,r})\right)^{t}$$

- We know that $\mathbb{P}(\mathcal{S}_{i,r}) > {}^{1}/_{en}$:

$$\mathbb{P}(\mathcal{F}_{i,t}) < \left(1 - \frac{1}{en}\right)^{t} < e^{-t/en}$$

Time Until First Success

No success by time t: $\mathbb{P}(\mathcal{F}_{i,t}) < e^{-t/en}$

$t = \lceil en \rceil$: $\mathbb{P}(\mathcal{F}_{i,t}) < 1/e$

- Generally if $t = \Theta(n)$: constant success probability

$t \geq en \cdot c \cdot \ln n$: $\mathbb{P}(\mathcal{F}_{i,t}) < 1/e^{c \cdot \ln n} = 1/n^c$

- For success probability $1 - 1/n^c$, we need $t = \Theta(n \log n)$.
- We say that i succeeds **with high probability** in $O(n \log n)$ time.

Time Until All Processes Succeed

Event $\mathcal{F}_t$: some process has not succeeded by time t

$$\mathcal{F}_t = \bigcup_{i=1}^{n} \mathcal{F}_{i,t}$$

Union Bound: For events $\mathcal{E}_1, \dots, \mathcal{E}_k$,

$$\mathbb{P}\left(\bigcup_{i}^{k} \mathcal{E}_i\right) \leq \sum_{i}^{k} \mathbb{P}(\mathcal{E}_i)$$

Probability that not all processes have succeeded by time t:

$$\mathbb{P}(\mathcal{F}_t) = \mathbb{P}\left(\bigcup_{i=1}^{n} \mathcal{F}_{i,t}\right) \leq \sum_{i=1}^{n} \mathbb{P}\left(\mathcal{F}_{i,t}\right) < n \cdot e^{-t/en}.$$

Time Until All Processes Succeed

Claim: With high probability, all processes succeed in the first $O(n \log n)$ time slots.

Proof:

- $\mathbb{P}(\mathcal{F}_t) < n \cdot e^{-t/en}$
- Set $t = \lceil en \cdot (c+1) \ln n \rceil$

Remark: $\Theta(n \log n)$ time slots are necessary for all processes to succeed with reasonable probability

Primality Testing

Problem: Given a natural number $n \geq 2$, is n a prime number?

Simple primality test:

1. **if** n is even **then**
2. **return** $(n = 2)$
3. **for** $i := 1$ **to** $\lfloor \sqrt{n}/2 \rfloor$ **do**
4. **if** $2i + 1$ divides n **then**
5. **return false**
6. **return true**

- **Running time:** $O(\sqrt{n})$

A Better Algorithm?

- How can we test primality efficiently?
- We need a little bit of basic number theory…

Square Roots of Unity: In $\mathbb{Z}_p^*$, where p is a prime, the only solutions of the equation $x^2 \equiv 1 \pmod{p}$ are $x \equiv \pm 1 \pmod{p}$

- If we find an $x \not\equiv \pm 1 \pmod{n}$ such that $x^2 \equiv 1 \pmod{n}$, we can conclude that n is not a prime.

Algorithm Idea

Claim: Let $p > 2$ be a prime number such that $p - 1 = 2^s d$ for an integer $s \geq 1$ and some odd integer $d \geq 3$. Then for all $a \in \mathbb{Z}_p^*$,

$$a^d \equiv 1 \pmod{p} \ \ \textbf{or} \ \ a^{2^r d} \equiv -1 \pmod{p} \ \text{ for some } 0 \leq r < s.$$

Proof:

- **Fermat's Little Theorem:** Given a prime number p,

$$\forall a \in \mathbb{Z}_p^*: \quad a^{p-1} \equiv 1 \pmod{p}$$

Primality Test

We have: If n is an odd prime and $n - 1 = 2^s d$ for an integer $s \geq 1$ and an odd integer $d \geq 3$. Then for all $a \in \{1, \ldots, n-1\}$,

$$a^d \equiv 1 \pmod{n} \ \textbf{ or } \ a^{2^r d} \equiv -1 \pmod{n} \ \text{ for some } 0 \leq r < s.$$

Idea: If we find an $a \in \{1, \ldots, n-1\}$ such that

$$a^d \not\equiv 1 \pmod{n} \ \textbf{ and } \ a^{2^r d} \not\equiv -1 \pmod{n} \ \text{ for all } 0 \leq r < s,$$

we can conclude that n is not a prime.

- For every odd composite $n > 2$, at least $3/4$ of all possible a satisfy the above condition
- How can we find such a *witness* a efficiently?

Miller-Rabin Primality Test

- Given a natural number $n \geq 2$, is n a prime number?

Miller-Rabin Test:

1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return true;**
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $x \coloneqq x^2 \bmod n$;
8.   **if** $x = 1$ **then return true;**
9. **return false;**

Analysis

Theorem:

- If n is prime, the Miller-Rabin test always returns **true**.
- If n is composite, the Miller-Rabin test returns **false** with probability at least $^3/_4$.

Proof:

- If n is prime, the test works for all values of a
- If n is composite, we need to pick a good witness a

Corollary: If the Miller-Rabin test is repeated k times, it fails to detect a composite number n with probability at most 4^{-k}.

Running Time

Cost of Modular Arithmetic:

- Representation of a number $x \in \mathbb{Z}_n$: $O(\log n)$ bits

- Cost of adding two numbers $x + y \bmod n$:

- Cost of multiplying two numbers $x \cdot y \bmod n$:
  - It's like multiplying degree $O(\log n)$ polynomials
    → use FFT to compute $z = x \cdot y$

Running Time

Cost of exponentiation $x^d \bmod n$:

- Can be done using $O(\log d)$ multiplications
- Base-2 representation of d: $d = \sum_{i=0}^{\lfloor \log d \rfloor} d_i 2^i$
- **Fast exponentiation:**
  1. $y \coloneqq 1;$
  2. **for** $i \coloneqq \lfloor \log d \rfloor$ **to** 0 **do**
  3. $\quad y \coloneqq y^2 \bmod n;$
  4. $\quad$ **if** $d_i = 1$ **then** $y \coloneqq y \cdot x \bmod n;$
  5. **return** $y;$
- **Example:** $d = 22 = 10110_2$

Running Time

Theorem: One iteration of the Miller-Rabin test can be implemented with running time $O(\log^2 n \cdot \log\log n \cdot \log\log\log n)$.

1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return true;**
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $\quad x \coloneqq x^2 \bmod n$;
8. $\quad$ **if** $x = 1$ **then return true;**
9. **return false;**

Deterministic Primality Test

- If a conjecture called the generalized Riemann hypothesis (GRH) is true, the Miller-Rabin test can be turned into a polynomial-time, deterministic algorithm

  → It is then sufficient to try all $a \in \{1, \dots, O(\log^2 n)\}$

- It has long not been proven whether a deterministic, polynomial-time algorithm exist

- In 2002, Agrawal, Kayal, and Saxena gave an $\tilde{O}(\log^{12} n)$-time deterministic algorithm
  - Has been improved to $\tilde{O}(\log^6 n)$

- In practice, the randomized Miller-Rabin test is still the fastest algorithm

Randomized Quicksort

Quicksort:

S	v

$S_\ell < v$	v	$S_r > v$

function Quick (S: sequence): sequence;

{returns the sorted sequence S}

begin

  if $\#S \leq 1$ then **return** S

  else { choose pivot element v in S;

    partition S into S_ℓ with elements $< v$,

    and S_r with elements $> v$

    return | Quick(S_ℓ) | v | Quick(S_r) |

end;

Randomized Quicksort Analysis

Randomized Quicksort: pick uniform random element as pivot

Running Time of sorting n elements:

- Let's just count the number of comparisons
- In the partitioning step, all $n - 1$ non-pivot elements have to be compared to the pivot

- **Number of comparisons:**

$$\boldsymbol{n - 1 \ + \ \#\text{comparisons in recursive calls}}$$

- **If rank of pivot is r:**
  recursive calls with $\boldsymbol{r - 1}$ and $\boldsymbol{n - r}$ elements

Randomized Quicksort Analysis

Random variables:

- C: total number of comparisons (for a given array of length n)
- R: rank of first pivot
- C_ℓ, C_r: number of comparisons for the 2 recursive calls

$$\mathbb{E}[C] = n - 1 + \mathbb{E}[C_\ell] + \mathbb{E}[C_r]$$

Law of Total Expectation:

$$\mathbb{E}[C] = \sum_{r=1}^{n} \mathbb{P}(R = r) \cdot \mathbb{E}[C|R = r]$$

$$= \sum_{r=1}^{n} \mathbb{P}(R = r) \cdot (n - 1 + \mathbb{E}[C_\ell|R = r] + \mathbb{E}[C_r|R = r])$$

Randomized Quicksort Analysis

We have seen that:

$$\mathbb{E}[C] = \sum_{r=1}^{n} \mathbb{P}(R = r) \cdot (n - 1 + \mathbb{E}[C_\ell | R = r] + \mathbb{E}[C_r | R = r])$$

Define:

- $\boldsymbol{T(n)}$: expected number of comparisons when sorting n elements

$$\mathbb{E}[C] = T(n)$$
$$\mathbb{E}[C_\ell | R = r] = T(r - 1)$$
$$\mathbb{E}[C_r | R = r] = T(n - r)$$

Recursion:

$$\boldsymbol{T(n) = \sum_{r=1}^{n} \frac{1}{n} \cdot \big(n - 1 + T(r - 1) + T(n - r)\big)}$$
$$\boldsymbol{T(0) = T(1) = 0}$$

Randomized Quicksort Analysis

Theorem: The expected number of comparisons when sorting n elements using randomized quicksort is $T(n) \leq 2n \ln n$.

Proof:

$$T(n) = \sum_{r=1}^{n} \frac{1}{n} \cdot \big(n - 1 + T(r-1) + T(n-r)\big), \qquad T(0) = 0$$

Randomized Quicksort Analysis

Theorem: The expected number of comparisons when sorting n elements using randomized quicksort is $T(n) \leq 2n \ln n$.

Proof:

$$T(n) \leq n - 1 + \frac{4}{n} \cdot \int_1^n x \ln x \, dx$$

$$\int x \ln x \, dx = \frac{x^2 \ln x}{2} - \frac{x^2}{4}$$

Alternative Analysis

Array to sort: [7 , 3 , 1 , 10 , 14 , 8 , 12 , 9 , 4 , 6 , 5 , 15 , 2 , 13 , 11]

Viewing quicksort run as a tree:

Comparisons

- Comparisons are only between pivot and non-pivot elements
- Every element can only be the pivot once:
  → every 2 elements can only be compared once!

- W.l.o.g., assume that the elements to sort are $1, 2, \ldots, n$
- Elements i and j are compared if and only if either i or j is a pivot before any element $h: i < h < j$ is chosen as pivot
  - i.e., iff i is an ancestor of j or j is an ancestor of i

$$\mathbb{P}(\text{comparison betw.}\, i \text{ and } j) = \frac{2}{j - i + 1}$$

Counting Comparisons

Random variable for every pair of elements (i, j):

$$\boldsymbol{X_{ij}} = \begin{cases} 1, & \text{if there is a comparison between } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$$

Number of comparisons: $\boldsymbol{X}$

$$X = \sum_{i<j} X_{ij}$$

- What is $\mathbb{E}[X]$?

Randomized Quicksort Analysis

Theorem: The expected number of comparisons when sorting n elements using randomized quicksort is $T(n) \leq 2n \ln n$.

Proof:

- **Linearity of expectation:**
  For all random variables $X_1, \dots, X_n$ and all $a_1, \dots, a_n \in \mathbb{R}$,

$$\mathbb{E}\left[\sum_i^n a_i X_i\right] = \sum_i^n a_i \mathbb{E}[X_i]\,.$$

Randomized Quicksort Analysis

Theorem: The expected number of comparisons when sorting n elements using randomized quicksort is $T(n) \leq 2n \ln n$.

Proof:

$$\mathbb{E}[X] = 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{j-i+1} = 2\sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1}\frac{1}{k}$$

Types of Randomized Algorithms

Las Vegas Algorithm:

- always a correct solution
- running time is a random variable

- **Example:** randomized quicksort, contention resolution

Monte Carlo Algorithm:

- probabilistic correctness guarantee (**m**ostly **c**orrect)
- fixed (deterministic) running time

- **Example:** primality test

Minimum Cut

Reminder: Given a graph $G = (V, E)$, a cut is a partition (A, B) of V such that $V = A \cup B$, $A \cap B = \emptyset$, $A, B \neq \emptyset$

Size of the cut $(\boldsymbol{A}, \boldsymbol{B})$: # of edges crossing the cut

- For weighted graphs, total edge weight crossing the cut

Goal: Find a cut of minimal size (i.e., of size $\lambda(G)$)

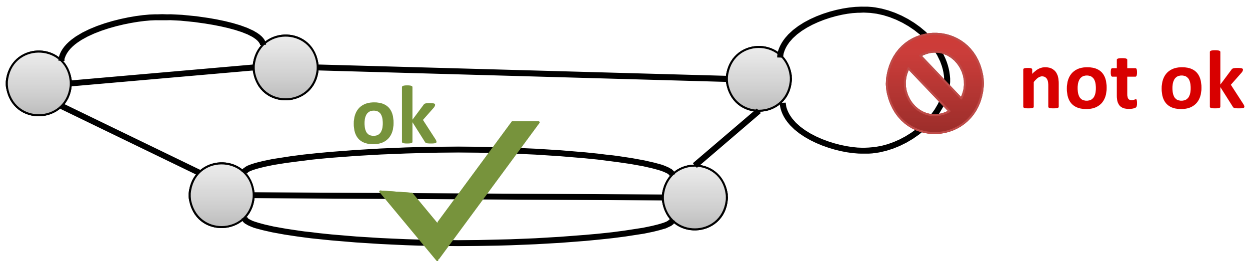
Maximum-flow based algorithm:

- Fix s, compute min s-t-cut for all $t \neq s$
- $O\big(m \cdot \lambda(G)\big) = O(mn)$ per s-t cut
- Gives an $\mathrm{O}\big(mn\lambda(G)\big) = O(mn^2)$-algorithm

Best-known deterministic algorithm: $O(mn + n^2 \log n)$

Edge Contractions

- In the following, we consider multi-graphs that can have multiple edges (but no self-loops)



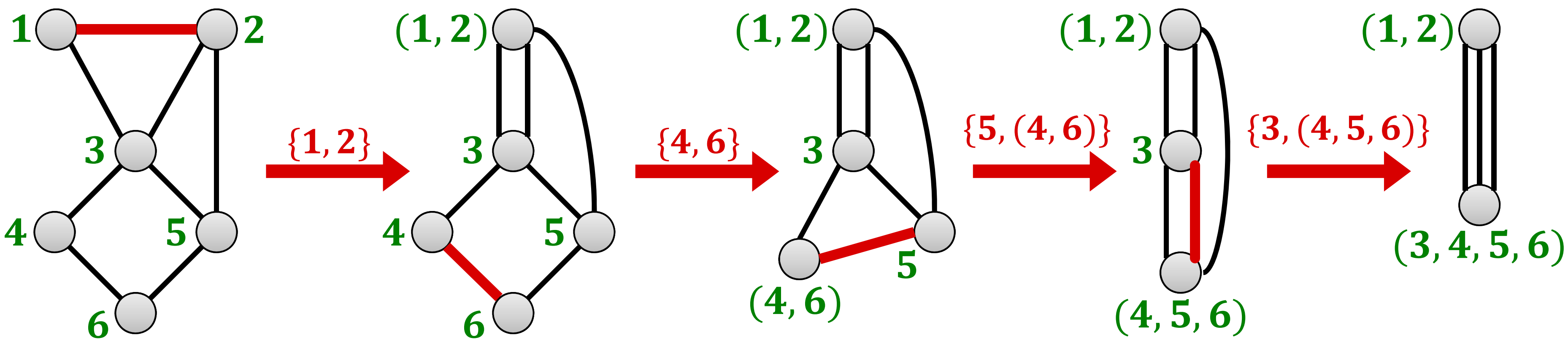
Contracting edge $\{u, v\}$:

- Replace nodes u, v by new node w
- For all edges $\{u, x\}$ and $\{v, x\}$, add an edge $\{w, x\}$
- Remove self-loops created at node w

Properties of Edge Contractions

Nodes:

- After contracting $\{u, v\}$, the new node represents u and v
- After a series of contractions, each node represents a subset of the original nodes



Cuts:

- Assume in the contracted graph, w represents nodes $S_w \subset V$
- The edges of a node w in a contracted graph are in a one-to-one correspondence with the edges crossing the cut $(S_w, V \setminus S_w)$

Randomized Contraction Algorithm

Algorithm:

while there are > 2 nodes **do**

contract a uniformly random edge

return cut induced by the last two remaining nodes

(cut defined by the original node sets represented by the last 2 nodes)

Theorem: The random contraction algorithm returns a minimum cut with probability at least $1/O(n^2)$.

- We will show this next.

Theorem: The random contraction algorithm can be implemented in time $O(n^2)$.

- There are $n - 2$ contractions, each can be done in time $O(n)$.
- You will show this in the exercises.

Contractions and Cuts

Lemma: If two original nodes $u, v \in V$ are merged into the same node of the contracted graph, there is a path connecting u and v in the original graph s.t. all edges on the path are contracted.

Proof:

- Contracting an edge $\{x, y\}$ merges the node sets represented by x and y and does not change any of the other node sets.

- The claim the follows by induction on the number of edge contractions.

Contractions and Cuts

Lemma: During the contraction algorithm, the edge connectivity (i.e., the size of the min. cut) cannot get smaller.

Proof:

- All cuts in a (partially) contracted graph correspond to cuts of the same size in the original graph G as follows:
  - For a node u of the contracted graph, let S_u be the set of original nodes that have been merged into u (the nodes that u represents)
  - Consider a cut (A, B) of the contracted graph
  - (A', B') with

$$A' := \bigcup_{u \in A} S_u, \qquad B' := \bigcup_{v \in B} S_v$$

    is a cut of G.
  - The edges crossing cut (A, B) are in one-to-one correspondence with the edges crossing cut (A', B').

Contraction and Cuts

Lemma: The contraction algorithm outputs a cut (A, B) of the input graph G if and only if it never contracts an edge crossing (A, B).

Proof:

1. If an edge crossing (A, B) is contracted, a pair of nodes $u \in A$, $v \in V$ is merged into the same node and the algorithm outputs a cut different from (A, B).

2. If no edge of (A, B) is contracted, no two nodes $u \in A$, $v \in B$ end up in the same contracted node because every path connecting u and v in G contains some edge crossing (A, B)

   In the end there are only 2 sets → output is (A, B)

Getting The Min Cut

Theorem: The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$.

To prove the theorem, we need the following claim:

Claim: If the minimum cut size of a multigraph G (no self-loops) is k, G has at least $kn/2$ edges.

Proof:

- Min cut has size $k \Longrightarrow$ all nodes have degree $\geq k$
  - A node v of degree $< k$ gives a cut $(\{v\}, V \setminus \{v\})$ of size $< k$
- Number of edges $m = {}^{1}/_{2} \cdot \sum_{v} \deg(v)$

Getting The Min Cut

Theorem: The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$.

Proof:

- Consider a fixed min cut (A, B), assume (A, B) has size k
- The algorithm outputs (A, B) iff none of the k edges crossing (A, B) gets contracted.
- Before contraction i, there are $n+1-i$ nodes
  → and thus $\geq (n+1-i)k/2$ edges
- If no edge crossing (A, B) is contracted before, the probability to contract an edge crossing (A, B) in step i is at most

$$\frac{k}{\frac{(n+1-i)k}{2}} = \frac{2}{n+1-i}.$$

Getting The Min Cut

Theorem: The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$.

Proof:

- If no edge crossing (A, B) is contracted before, the probability to contract an edge crossing (A, B) in step i is at most ${}^{2}/_{n+1-i}$.
- Event $\mathcal{E}_i$: edge contracted in step i is **not** crossing (A, B)

Getting The Min Cut

Theorem: The probability that the algorithm outputs a minimum cut is at least $2/n(n-1)$.

Proof:

- $\mathbb{P}(\mathcal{E}_{i+1} | \mathcal{E}_1 \cap \dots \cap \mathcal{E}_i) = {}^2/_{n-i}$
- No edge crossing (A, B) contracted: event $\mathcal{E} = \bigcap_{i=1}^{n-2} \mathcal{E}_i$

Randomized Min Cut Algorithm

Theorem: If the contraction algorithm is repeated $O(n^2 \log n)$ times, one of the $O(n^2 \log n)$ instances returns a min. cut w.h.p.

Proof:

- Probability to not get a minimum cut in $c \cdot \binom{n}{2} \cdot \ln n$ iterations:

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{c \cdot \binom{n}{2} \cdot \ln n} < e^{-c \ln n} = \frac{1}{n^c}$$

Corollary: The contraction algorithm allows to compute a minimum cut in $O(n^4 \log n)$ time w.h.p.

- Each instance can be implemented in $O(n^2)$ time.
  ($O(n)$ time per contraction)

Can We Do Better?

- Time $O(n^4 \log n)$ is not very spectacular, a simple max flow based implementation has time $O(n^4)$.

However, we will see that the contraction algorithm is nevertheless very interesting because:

1. The algorithm can be improved to beat every known deterministic algorithm.

1. It allows to obtain strong statements about the distribution of cuts in graphs.

Better Randomized Algorithm

Recall:

- Consider a fixed min cut (A, B), assume (A, B) has size k
- The algorithm outputs (A, B) iff none of the k edges crossing (A, B) gets contracted.
- Throughout the algorithm, the edge connectivity is at least k and therefore each node has degree $\geq k$
- Before contraction i, there are $n + 1 - i$ nodes and thus at least $(n + 1 - i)k/2$ edges
- If no edge crossing (A, B) is contracted before, the probability to contract an edge crossing (A, B) in step i is at most

$$\frac{k}{\frac{(n+1-i)k}{2}} = \frac{2}{n+1-i}.$$

Improving the Contraction Algorithm

- For a specific min cut (A, B), if (A, B) survives the first i contractions,

$$\mathbb{P}(\text{edge crossing } (A, B) \text{ in contraction } i + 1) \leq \frac{2}{n - i}.$$

- **Observation:** The probability only gets large for large i

- **Idea:** The early steps are much safer than the late steps.

  Maybe we can repeat the late steps more often than the early ones.

Safe Contraction Phase

Lemma: A given min cut (A, B) of an n-node graph G survives the first $n - \left\lceil {}^{n}/_{\sqrt{2}} + 1 \right\rceil$ contractions, with probability $> {}^{1}/_{2}$.

Proof:

- Event $\mathcal{E}_i$: cut (A, B) survives contraction i
- Probability that (A, B) survives the first $n - t$ contractions:

Better Randomized Algorithm

Let's simplify a bit:

- Pretend that $n/\sqrt{2}$ is an integer (for all n we will need it).
- Assume that a given min cut survives the first $n - {}^{n}/_{\sqrt{2}}$ contractions with probability $\geq {}^{1}/_{2}$.

$\mathbf{contract}(\boldsymbol{G}, \boldsymbol{t})$**:**

- Starting with n-node graph G, perform $n - t$ edge contractions such that the new graph has t nodes.

$\mathbf{mincut}(\boldsymbol{G})$**:**

1. $X_1 \coloneqq \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right)$;
2. $X_2 \coloneqq \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right)$;
3. **return** $\min\{X_1, X_2\}$;

Success Probability

$\mathbf{mincut}(\boldsymbol{G})$**:**

1. $X_1 \coloneqq \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right);$
2. $X_2 \coloneqq \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right);$
3. **return** $\min\{X_1, X_2\};$

$\boldsymbol{P(n)}$**:** probability that the above algorithm returns a min cut when applied to a graph with n nodes.

- Probability that X_1 is a min cut $\geq$

Recursion:

Success Probability

Theorem: The recursive randomized min cut algorithm returns a minimum cut with probability at least $1/\log_2 n$.

Proof (by induction on n):

$$P(n) = P\left(\frac{n}{\sqrt{2}}\right) - \frac{1}{4} \cdot P\left(\frac{n}{\sqrt{2}}\right)^2, \qquad P(2) = 1$$

Running Time

1. $X_1 := \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right)$;
2. $X_2 := \mathrm{mincut}\left(\mathrm{contract}\left(G, n/\sqrt{2}\right)\right)$;
3. **return** $\min\{X_1, X_2\}$;

Recursion:

- $T(n)$: time to apply algorithm to n-node graphs
- Recursive calls: $2T\left({}^{n}/_{\sqrt{2}}\right)$
- Number of contractions to get to ${}^{n}/_{\sqrt{2}}$ nodes: $O(n)$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2), \qquad T(2) = O(1)$$

Running Time

Theorem: The running time of the recursive, randomized min cut algorithm is $O(n^2 \log n)$.

Proof:

- Can be shown in the usual way, by induction on n

Remark:

- The running time is only by an $O(\log n)$-factor slower than the basic contraction algorithm.
- The success probability is exponentially better!

Number of Minimum Cuts

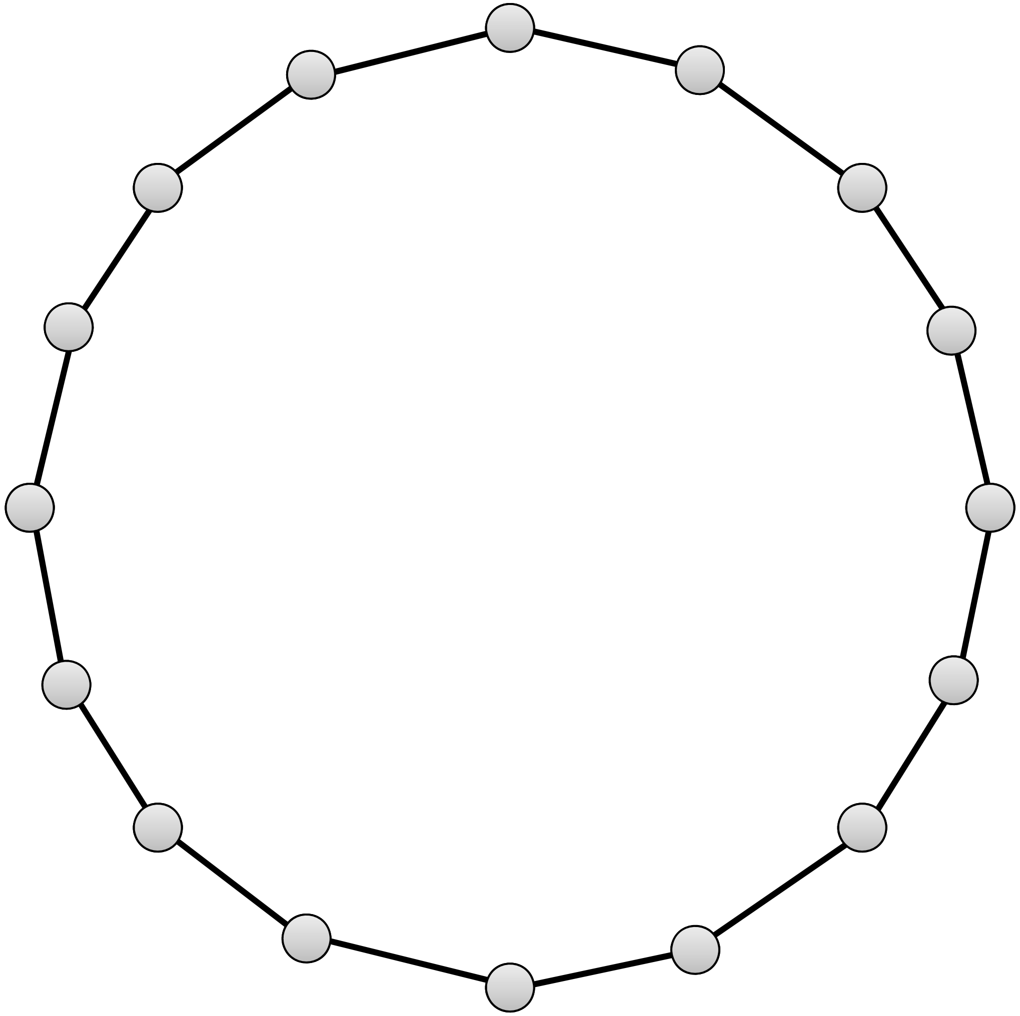
- Given a graph G, how many minimum cuts can there be?

- Or alternatively: If G has edge connectivity k, how many ways are there to remove k edges to disconnect G?

- Note that the total number of cuts is large.

Number of Minimum Cuts

Example: Ring with n nodes



- Minimum cut size: 2
- Every two edges induce a min cut
- Number of edge pairs:

$$\binom{n}{2}$$

- Are there graphs with more min cuts?

Number of Min Cuts

Theorem: The number of minimum cuts of a graph is at most $\binom{n}{2}$.

Proof:

- Assume there are s min cuts
- For $i \in \{1, \dots, s\}$, define event $\mathcal{C}_i$:

  $\mathcal{C}_i \coloneqq \{\text{basic contraction algorithm returns min cut } i\}$

- We know that for $i \in \{1, \dots, s\}$: $\mathbb{P}(\mathcal{C}_i) = 1/\binom{n}{2}$
- Events $\mathcal{C}_1, \dots, \mathcal{C}_s$ are disjoint:

$$\mathbb{P}\left(\bigcup_{i=1}^{s} \mathcal{C}_i\right) = \sum_{i=1}^{s} \mathbb{P}(\mathcal{C}_i) = \frac{s}{\binom{n}{2}}$$