Advanced Algorithms (I)

Shanghai Jiao Tong University

Chihao Zhang

March 2nd, 2020

Information

Instructor: 张驰豪 (chihao@sjtu.edu.cn)

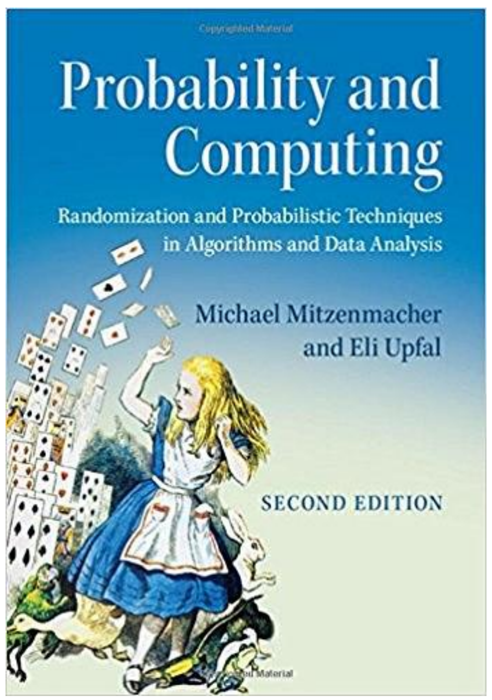
TA: 杨凤麟 (yangfl@sjtu.edu.cn)

Every Monday, 10:00 am - 11:40 am

Zoom @ 123363659

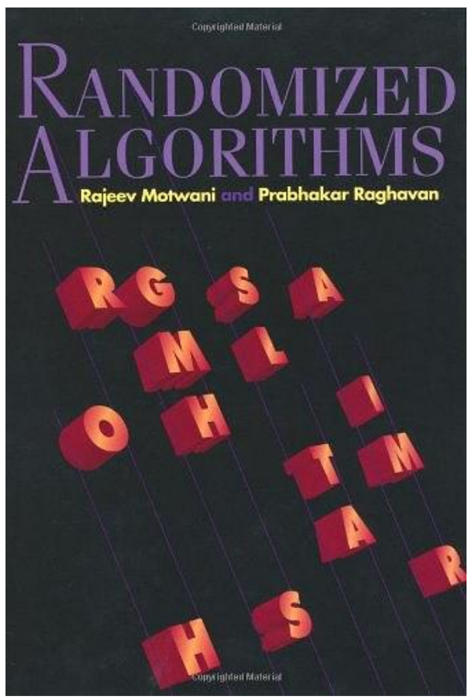
Office Hour: via Canvas or WeChat Group

References



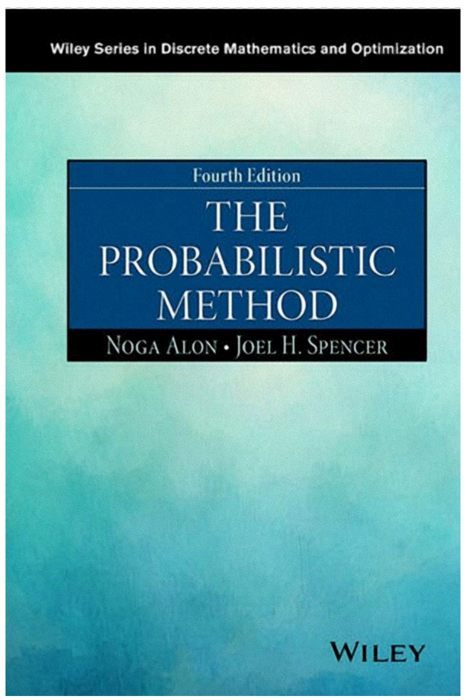
Probability and Computing

M. Mitzenmacher & E. Upfal



Randomized Algorithms

R. Motwani & P. Raghavan



The Probabilistic Method

N. Alon & J. Spencer

Polynomial Identity Testing

Polynomial Identity Testing

Given two polynomials

$$F(x) = \prod_{i=1}^{d} (x - a_i) \quad \text{and} \quad G(x) = \sum_{i=0}^{d} b_i x^i$$

Polynomial Identity Testing

Given two polynomials

$$F(x) = \prod_{i=1}^{d} (x - a_i) \quad \text{and} \quad G(x) = \sum_{i=0}^{d} b_i x^i$$

Is $F(x) \equiv G(x)$?

Polynomial Identity Testing

Given two polynomials

$$F(x) = \prod_{i=1}^{d} (x - a_i) \text{ and } G(x) = \sum_{i=0}^{d} b_i x^i$$

Is $F(x) \equiv G(x)$?

Example. $F(x) = (x-1)(x-2)(x+3); G(x) = x^3 - 7x + 6$

One can expand $F(x)$ and compare the coefficients…

One can expand $F(x)$ and compare the coefficients…

It takes $O(d^2)$ arithmetic operations.

One can expand $F(x)$ and compare the coefficients…

It takes $O(d^2)$ arithmetic operations.

Can be improved to $O(d \log d)$ using FFT.

One can expand $F(x)$ and compare the coefficients…

It takes $O(d^2)$ arithmetic operations.

Can be improved to $O(d \log d)$ using FFT.

If random coins are allowed…

One can expand $F(x)$ and compare the coefficients…

It takes $O(d^2)$ arithmetic operations.

Can be improved to $O(d \log d)$ using FFT.

If random coins are allowed…

- The problem can be solved much faster

One can expand $F(x)$ and compare the coefficients…

It takes $O(d^2)$ arithmetic operations.

Can be improved to $O(d \log d)$ using FFT.

If random coins are allowed…

- The problem can be solved much faster
- at the cost of making error.

Choosing a uniform number $s \in \{1, 2, \ldots, 100d\}$

Choosing a uniform number $s \in \{1,2,\ldots,100d\}$

Test whether $F(s) = G(s)$

Choosing a uniform number $s \in \{1, 2, \ldots, 100d\}$

Test whether $F(s) = G(s)$

- If $F(x) \equiv G(x)$, then it always holds that $F(s) = G(s)$

Choosing a uniform number $s \in \{1,2,\ldots,100d\}$

Test whether $F(s) = G(s)$

- If $F(x) \equiv G(x)$, then it always holds that $F(s) = G(s)$
- If $F(x) \not\equiv G(x)$, how likely is that $F(s) \neq G(s)$?

Choosing a uniform number $s \in \{1, 2, \ldots, 100d\}$

Test whether $F(s) = G(s)$

- If $F(x) \equiv G(x)$, then it always holds that $F(s) = G(s)$
- If $F(x) \not\equiv G(x)$, how likely is that $F(s) \neq G(s)$?

Theorem. (Fundamental Theorem of Algebra)

A polynomial of degree d has at most d roots in $\mathbb{C}$

Our algorithm outputs wrong answer only when

Our algorithm outputs wrong answer only when

- $F(x) \not\equiv G(x)$; and

Our algorithm outputs wrong answer only when

- $F(x) \not\equiv G(x)$; and
- s is a root of $F(x) - G(x)$

Our algorithm outputs wrong answer only when

- $F(x) \not\equiv G(x)$; and
- s is a root of $F(x) - G(x)$

This happens with probability at most $\dfrac{1}{100}$.

Our algorithm outputs wrong answer only when

- $F(x) \not\equiv G(x)$; and
- s is a root of $F(x) - G(x)$

This happens with probability at most $\frac{1}{100}$.

It only costs $O(d)$ operations to compute $F(s)$ and $G(s)$.

Our algorithm outputs wrong answer only when

- $F(x) \not\equiv G(x)$; and
- s is a root of $F(x) - G(x)$

This happens with probability at most $\frac{1}{100}$.

It only costs $O(d)$ operations to compute $F(s)$ and $G(s)$.

One can repeat the algorithm t times:

- error reduces to 100^{-t};
- cost increases to $O(td)$.

Multi-variable Polynomials

Multi-variable Polynomials

The idea applies to a more general setting.

Multi-variable Polynomials

The idea applies to a more general setting.

Let $F, G \in \mathbb{F}(x_1, \ldots, x_n)$ for some field $\mathbb{F}$,

Multi-variable Polynomials

The idea applies to a more general setting.

Let $F, G \in \mathbb{F}(x_1, \ldots, x_n)$ for some field $\mathbb{F}$,

$$\boxed{\text{Is } F(x_1, \ldots, x_n) \equiv G(x_1, \ldots, x_n)?}$$

Theorem. (Schwartz-Zippel Theorem)

Let $Q \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero multivariate polynomial of degree at most d. For any set $U \subseteq \mathbb{F}$, it holds that

$$\Pr_{r_1,\ldots,r_n \in_R U}[Q(r_1, \ldots, r_n) = 0] \leq \frac{d}{|U|}$$

Proof of Schwartz-Zippel

Proof of Schwartz-Zippel

Induction on n, case $n = 1$ is Fundamental Theorem of Algebra.

Proof of Schwartz-Zippel

Induction on n, case $n = 1$ is Fundamental Theorem of Algebra.

Assuming it holds for smaller n…

Proof of Schwartz-Zippel

Induction on n, case $n = 1$ is Fundamental Theorem of Algebra.

Assuming it holds for smaller n…

$$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i \cdot Q_i(x_2, \ldots, x_n)$$

Proof of Schwartz-Zippel

Induction on n, case $n = 1$ is Fundamental Theorem of Algebra.

Assuming it holds for smaller n…

$$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i \cdot Q_i(x_2, \ldots, x_n)$$

$$\Pr\left[Q = 0\right] \leq \Pr\left[Q_k = 0\right] + \Pr\left[Q = 0 \,|\, Q_k \neq 0\right] \leq \frac{d - k}{|U|} + \frac{k}{|U|}$$

Our randomized algorithm generalizes to the multi-variable setting by Schwartz-Zippel theorem.

Our randomized algorithm generalizes to the multi-variable setting by Schwartz-Zippel theorem.

If the polynomials are given in product form, one scan of F and G is sufficient to evaluate them.

Our randomized algorithm generalizes to the multi-variable setting by Schwartz-Zippel theorem.

If the polynomials are given in product form, one scan of F and G is sufficient to evaluate them.

Linear time algorithm with at most $1\,\%$ error!

Our randomized algorithm generalizes to the multi-variable setting by Schwartz-Zippel theorem.

If the polynomials are given in product form, one scan of F and G is sufficient to evaluate them.

Linear time algorithm with at most $1\,\%$ error!

It is a wide open problem in the complexity theory that whether this can be done in deterministic polynomial time.

Some Complexity Theory

Some Complexity Theory

Problems solvable in deterministic polynomial-time: **P**

Some Complexity Theory

Problems solvable in deterministic polynomial-time: **P**

Problems solvable in randomized polynomial-time: **BPP**

Some Complexity Theory

Problems solvable in deterministic polynomial-time: **P**

Problems solvable in randomized polynomial-time: **BPP**

Is **BPP** = **P**?

Min-Cut in a Graph

Min-Cut in a Graph

A *cut* in a graph $G = (V, E)$ is a set of edges $C \subseteq E$ whose removal disconnects G.

Min-Cut in a Graph

A *cut* in a graph $G = (V, E)$ is a set of edges $C \subseteq E$ whose removal disconnects G.

How to find the minimum cut?

Min-Cut in a Graph

A *cut* in a graph $G = (V, E)$ is a set of edges $C \subseteq E$ whose removal disconnects G.

How to find the minimum cut?

It can be solved using max-flow techniques

Min-Cut in a Graph

A *cut* in a graph $G = (V, E)$ is a set of edges $C \subseteq E$ whose removal disconnects G.

How to find the minimum cut?

It can be solved using max-flow techniques

With the fastest max-flow algorithm, it takes $O(n \times mn)$ time.

Karger’s Min-Cut Algorithm

Karger’s Min-Cut Algorithm

David Karger

Karger's Min-Cut Algorithm

David Karger

Using random bits, Karger found a much simpler algorithm

Karger's Min-Cut Algorithm

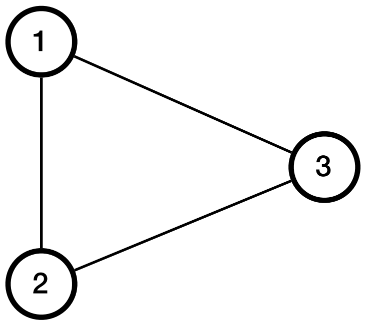
David Karger

Using random bits, Karger found a much simpler algorithm

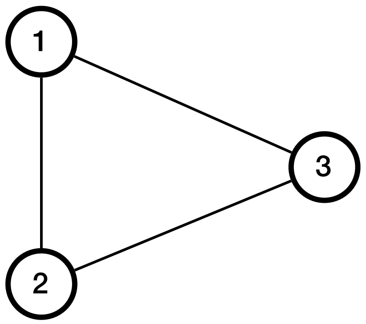
The only operation required is edge contraction

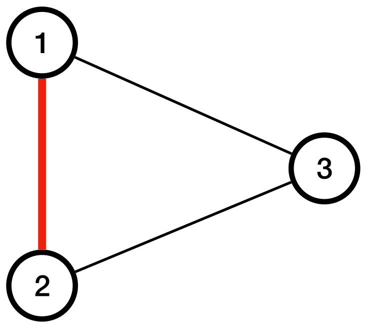
Edge Contraction

Edge Contraction

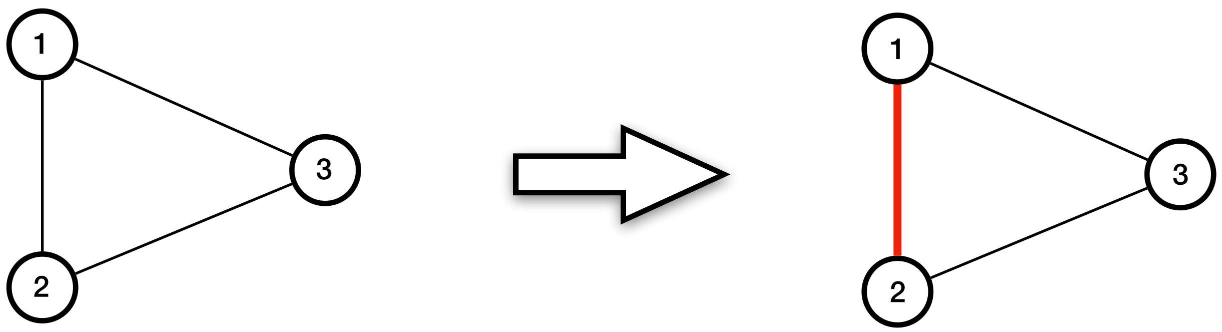


Edge Contraction

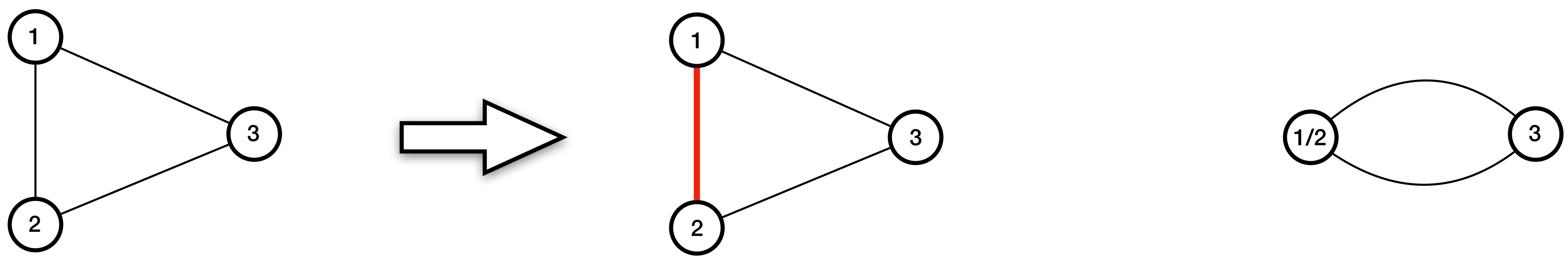




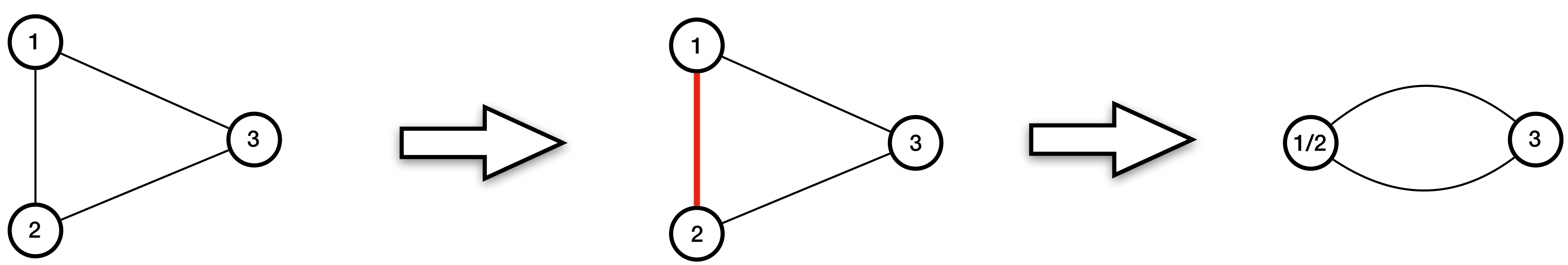
Edge Contraction



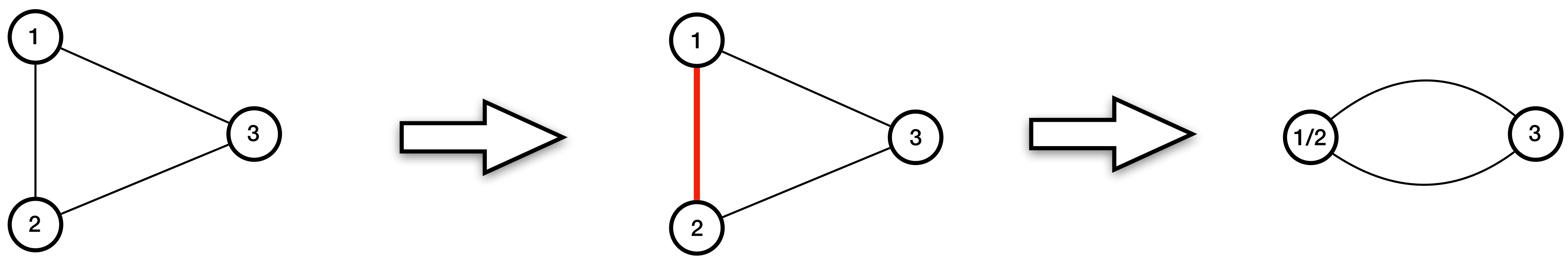
Edge Contraction



Edge Contraction

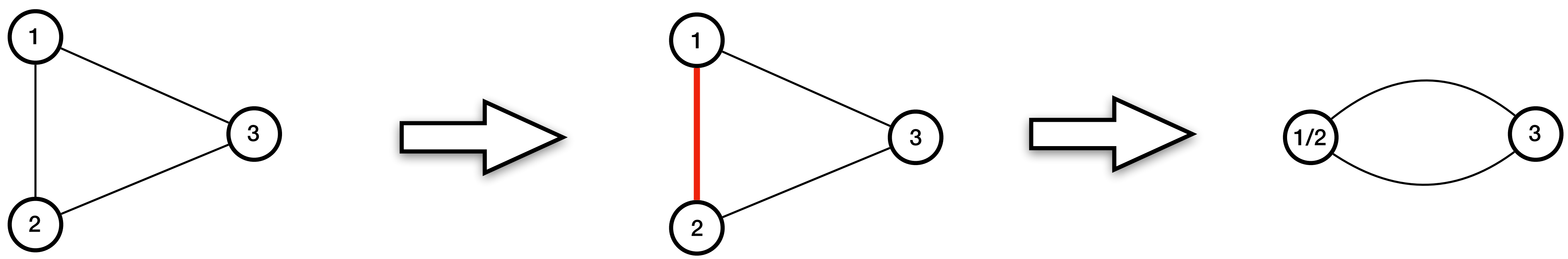


Edge Contraction



- no self-loop

Edge Contraction



- no self-loop
- parallel edges may exist

The Algorithm

The Algorithm

Karger's Min-cut Algorithm

1. Randomly choose an edge and contract it until only two vertices remains.
2. Output remaining edges.

Analysis

Analysis

The algorithm contracts $n - 2$ pair of vertices in total.

Analysis

The algorithm contracts $n-2$ pair of vertices in total.

Fix an minimum cut C, we bound the probability that it survives.

Analysis

The algorithm contracts $n - 2$ pair of vertices in total.

Fix an minimum cut C, we bound the probability that it survives.

Assume the removal of C separates $S \subseteq V$ and $\bar{S} = V \backslash S$.

Analysis

The algorithm contracts $n-2$ pair of vertices in total.

Fix an minimum cut C, we bound the probability that it survives.

Assume the removal of C separates $S \subseteq V$ and $\bar{S} = V \backslash S$.

All contractions happen within S or $\bar{S}$.

For $i = 1, \ldots, n-2$, let A_i be the event that "i-th contraction avoids C"

For $i = 1,\ldots,n-2$, let A_i be the event that "i-th contraction avoids C"

We need to bound

For $i = 1, \ldots, n-2$, let A_i be the event that "i-th contraction avoids C"

We need to bound

$$\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \prod_{i=1}^{n-2} \Pr\left[A_i \,\middle|\, \bigcap_{j=1}^{i-1} A_j\right]$$

For $i = 1,\ldots,n-2$, let A_i be the event that "i-th contraction avoids C"

We need to bound

$$\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \prod_{i=1}^{n-2} \Pr\left[A_i \,\middle|\, \bigcap_{j=1}^{i-1} A_j\right]$$

We assume $|C| = k$

In i-th contraction,

In i-th contraction,

- the graph contains $n - i + 1$ vertices;
- each vertex is of degree at least k.

In i-th contraction,

- the graph contains $n - i + 1$ vertices;
- each vertex is of degree at least k.

Therefore, conditional on that C still survives,

In i-th contraction,

- the graph contains $n - i + 1$ vertices;
- each vertex is of degree at least k.

Therefore, conditional on that C still survives,

$$\Pr\left[A_i \,\middle|\, \bigcap_{j=1}^{i-1} A_j\right] \geq 1 - \frac{2k}{k(n-i+1)} = \frac{n-i-1}{n-i+1}$$

Therefore

Therefore

$$\begin{aligned}
\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] &= \prod_{i=1}^{n-2} \Pr\left[A_i \,\middle|\, \bigcap_{j=1}^{i-1} A_j\right] \\
&\geq \prod_{i=1}^{n-2} \frac{n-i-1}{n-i+1} \\
&= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3} \\
&= \frac{2}{n(n-1)}
\end{aligned}$$

So if we repeat the algorithm $50n^2$ times, the minimum cut survives with probability at least

So if we repeat the algorithm $50n^2$ times, the minimum cut survives with probability at least

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^{50n^2} \geq 1 - e^{-100}$$

So if we repeat the algorithm $50n^2$ times, the minimum cut survives with probability at least

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^{50n^2} \geq 1 - e^{-100}$$

How about the time cost?

So if we repeat the algorithm $50n^2$ times, the minimum cut survives with probability at least

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^{50n^2} \geq 1 - e^{-100}$$

How about the time cost?

If we store the graph in a adjacency matrix, one needs $O(n)$ to contract an edge…

Karger-Stein's Trick

Karger-Stein's Trick

Recall $\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3}$

Karger-Stein's Trick

Recall $\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3}$

The more we contracts, the easier C gets hit

Karger-Stein's Trick

Recall $\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3}$

The more we contracts, the easier C gets hit

Idea: Make a copy before it becomes too bad!

Karger-Stein's Trick

Recall $\Pr\left[\bigcap_{i=1}^{n-2} A_i\right] = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3}$

The more we contracts, the easier C gets hit

Idea: Make a copy before it becomes too bad!

The success probability can be improved to $\Omega\left(\frac{1}{\log n}\right)$