CS5314
Randomized Algorithms

Lecture 3: Events and Probability
(verifying matrix multiplication, randomized min-cut)

Objectives

- A simple randomized algorithm to check if we multiply two matrices correctly
- A simple randomized algorithm for finding min-cut of a graph
- Introduce concept:
  - Law of Total Probability
  - Principle of Deferred Decision

Verifying Matrix Multiplication

- Suppose our friend, John, tells us that he has just multiplied two $n \times n$ matrices A and B, and obtained a resultant $n \times n$ matrix C
- He wants us to double-check for him whether $AB = C$
- How can we help John?

Verifying Matrix Multiplication

- One way to do so is to multiply two matrices A and B again, and compare the result with C
- A simple way to multiply A and B would take $O(n^3)$ operations
  - Even if we apply the best-known matrix algorithm [Coppersmith-Winograd 1990], we still need $O(n^{2.376})$ operations
- Can we do the checking faster?

A Randomized Algorithm
(verifying matrix multiplication)

- Pick an n-dimension vector $r = (r_1, r_2, \ldots, r_n)$, uniformly at random, from $\{0,1\}^n$
  - Precisely, r is a $n \times 1$ matrix
- Compute ABr and Cr
- If $ABr = Cr$, we conclude $AB \equiv C$.
- Otherwise, we conclude $AB \not\equiv C$.

A Randomized Algorithm
(Performance Analysis)

Questions

- What is the runtime of the algorithm?

  Ans. $O(n^2)$ time. It is because:

  - To compute ABr, we compute Br first to get a vector r', and then compute Ar'. Each step thus takes $O(n^2)$ time.
  - Computing Cr and the final comparison also takes $O(n^2)$ time.

A Randomized Algorithm
(Performance Analysis)

Questions

- When will the algorithm make an error?

  Ans. … when $AB \neq C$, and the r we choose satisfies $ABr = Cr$

- When $AB \neq C$, can we bound $\Pr(ABr = Cr)$?

  To bound it, let $D = AB - C$. Then,

$$\Pr(ABr = Cr) = \Pr(Dr = 0)$$

A Randomized Algorithm
(Performance Analysis)

- Next, since $D \neq 0$, there must be some entry in D, say d_{xy}, is not zero

$$\begin{pmatrix} d_{11} & d_{12} & \cdots & & \cdots & d_{1n} \\ \vdots & \ddots & & & & \vdots \\ d_{x1} & d_{x2} & \cdots & d_{xy} & \cdots & d_{xn} \\ \vdots & & & \ddots & & \vdots \\ d_{n1} & d_{n2} & \cdots & & & d_{nn} \end{pmatrix}$$

non-zero

A Randomized Algorithm
(Performance Analysis)

- Then, when $Dr=0$,

  ➔ $\sum_{j=1,2,\ldots,n} d_{xj}\, r_j = 0$

  ➔ Equivalently, $r_y = -\sum_{j\neq y} d_{xj} r_j \;/\; d_{xy}$

- Thus, [why?]

  $\Pr(Dr = 0) \leq \Pr(\, r_y = -\sum_{j\neq y} d_{xj} r_j \;/\; d_{xy})$

A useful lemma

Lemma: To obtain $r = (r_1, r_2, \ldots, r_n)$,

choosing r uniformly at random from $\{0,1\}^n$

is equivalent to

choosing each r_i independently and uniformly at random from $\{0,1\}$

Proof: If each r_i is chosen independently and uniformly at random, then each of the 2^n possible vectors r is chosen with probability $1/2^n$

Law of Total Probability

Theorem: Let $E_1, E_2, \ldots, E_n$ be mutually disjoint events in the sample space Ω, and let $\bigcup_{i=1,2,\ldots,n} E_i = \Omega$.

(I.e., $E_1, E_2, \ldots, E_n$ forms a partition of Ω)

Then, for any event B,

$$\Pr(B) = \sum_{i=1,2,\ldots,n} \Pr(B \cap E_i)$$

$$= \sum_{i=1,2,\ldots,n} \Pr(B \mid E_i) \Pr(E_i)$$

Back to the Analysis…

$\Pr(ABr = Cr) = \Pr(Dr = 0)$

$\leq \Pr(\, r_y = -\sum_{j \neq y} d_{xj} r_j / d_{xy} \,)$

$= \sum_S \Pr(\, (r_y = -\sum_{j \neq y} d_{xj}\, r_j /\, d_{xy}) \cap S),$

where S denotes a particular choice for $(r_1, r_2, \ldots, r_n)$ with r_y missing ➔ the summation is over all 2^{n-1} choices

$= \sum_S \Pr(\, (r_y = -\sum_{j \neq y} d_{xj}\, r_j /\, d_{xy}) \mid S)\, \Pr(S)$

$\leq \sum_S (1/2) \Pr(S)$ [why??]

$= 1/2.$

Back to the Analysis…

Conclusion:

- When $AB \neq C$, $\Pr(ABr = Cr) \leq 1/2$

➔ algorithm is wrong with prob $\leq 1/2$

➔ correct in at least 50% of time

- By repeat running k times, probability that the result is correct $\geq 1 - 1/2^k$

Back to the Analysis... (Remark)

- In previous analysis, when we compute

$$\Pr(\, r_y = -\textstyle\sum_{j \neq y} d_{xj} r_j / d_{xy} \,),$$

  we did not consider each choice of **r**, and sum up by

$$\textstyle\sum_{\mathbf{r} \in \{0,1\}^n} \Pr((r_y = -\sum_{j \neq y} d_{xj}\, r_j / d_{xy}) \mid \mathbf{r}) \Pr(\mathbf{r})$$

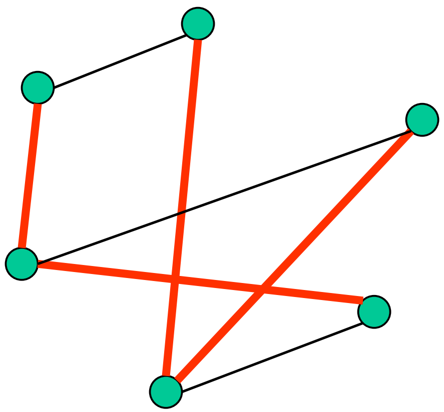
- Instead, we fix only some part of **r** at first, and fix some part (r_y) later
- Known as: Principle of Deferred Decision

Min-Cut Problem

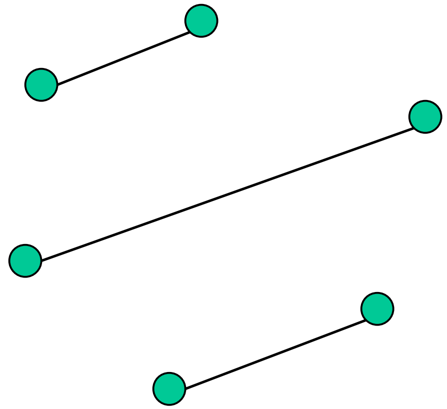
Let G be an undirected graph.

- A cut in G is a set of edges such that by removing them, G is broken into more than one connected components

befre removing a cut



after removing a cut



Min-Cut Problem

- Min-Cut Problem: To find a cut for G whose size (number of edges) is minimum
- Useful in studying network reliability
- Let n = number of vertices in G
- m = number of edges in G
- Best deterministic (i.e., not randomized) algorithm for Min-Cut runs in:

$$O(\, nm + n^2 \log n \,) \text{ time}$$

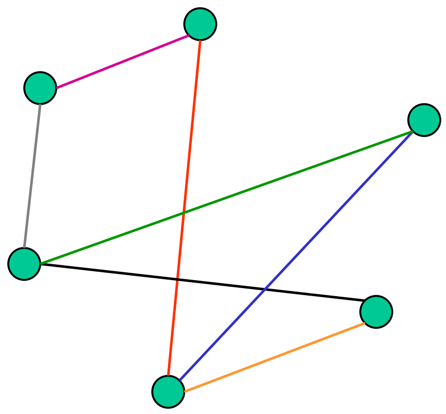
Randomized Min-Cut

- Set G' to be G
- While G' has more than 2 vertices
  1. Pick an edge e from G', uniformly at random, among all edges in G'
  2. Contract e (and remove self-loops) to obtain a new graph
  3. Set G' to be this new graph
- Output the set of edges in G'

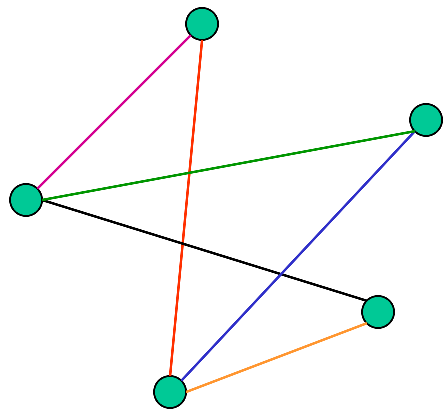
// Boundary Case: Return { } if input G is not connected

Example Run

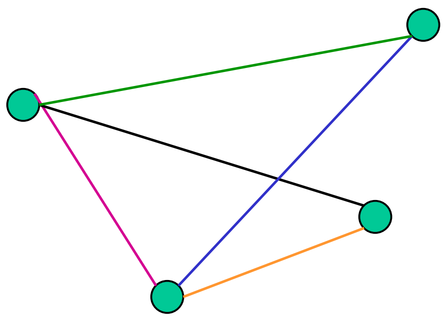
Step 1. The original *G*



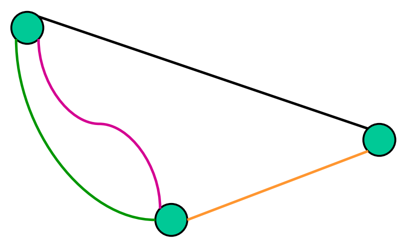
Step 2. Contracting gray edge



Step 3. Contracting red edge

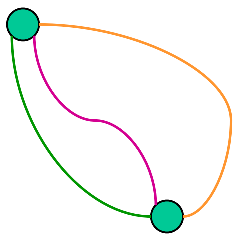


Step 4. Contracting blue edge



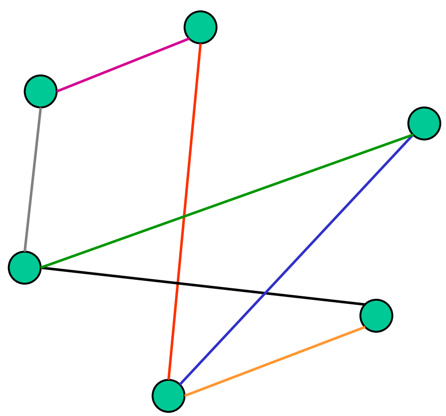
Example Run

Step 5. Contracting black edge

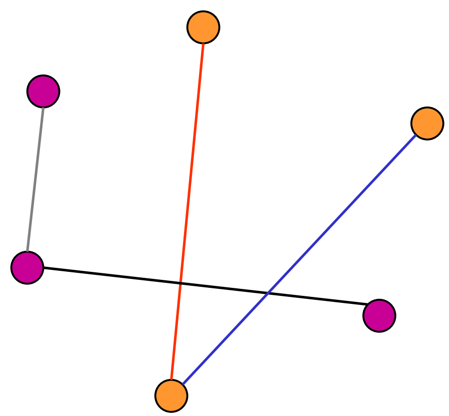


➔ The remaining edges form a cut in G

The original G



G - remaining edges ➔ more than one component



Randomized Min-Cut (Facts)

1. Each edge contraction removes 1 vertex
2. Edges in final output is a cut of G
3. Not every final output is a min-cut
4. Suppose C is one of the min-cut of G. If every edge of C is not contracted, then the final output contains only edges in C [why??]

   → By 2 and 4, final output must be C

Randomized Min-Cut
(Performance Analysis)

Suppose C a min-cut of G. Let k be its size

Then,

Pr(the algorithm is correct)

$\geq$ Pr(C is output in the end)

= Pr(all edges of C are not contracted)

Question: what is the above probability?

Randomized Min-Cut
(Performance Analysis)

Let E_i be the event that the edge contracted at the i^{th} step is not from *C*

Then, Pr(all edges of *C* are not contracted)

$$= \Pr\left(\bigcap_{i=1,2,\ldots,n-2} E_i \right)$$ [why n-2?]

$$= \Pr(E_1) \times \Pr(E_2 \mid E_1) \times \Pr(E_3 \mid E_1 \cap E_2) \times$$

$$\ldots \times \Pr\left(E_{n-2} \mid \bigcap_{i=1,2,\ldots,n-3} E_i\right)$$

Randomized Min-Cut
(Performance Analysis)

Key Observation:

At the beginning of each step, the degree of any node is at least k [why??]

➔ How many edges in G' before i^{th} step?

[I.e., when $i-1$ vertices are already removed]

Randomized Min-Cut
(Performance Analysis)

Thus,

$$\Pr(E_1) \geq 1 - \frac{k}{nk/2} = 1 - \frac{2}{n}$$

$$\Pr(E_2 \mid E_1) \geq 1 - \frac{k}{(n-1)k/2} = 1 - \frac{2}{n-1}$$

$$\Pr(E_3 \mid E_1 \cap E_2) \geq 1 - \frac{k}{(n-2)k/2} = 1 - \frac{2}{n-2}$$

$$\vdots$$

$$\Pr(E_{n-2} \mid \cap_{i=1,2,\ldots,n-3} E_i) \geq 1 - \frac{k}{3k/2} = 1/3$$

Randomized Min-Cut
(Performance Analysis)

Then,

Pr(all edges of *C* are not contracted)

= Pr($\bigcap_{i=1,2,\ldots,n-2} E_i$)

$\geq$ (1- 2/n)(1- 2/(n-1))(1- 2/(n-2)) … (1/3)

$$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{n-5}{n-3} \cdot \ldots \cdot \frac{4}{6} \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}$$

$$= \frac{2}{n(n-1)}$$

Randomized Min-Cut
(Performance Analysis)

Conclusion:

- Algorithm is correct with prob $\geq \dfrac{2}{n(n-1)}$

- Repeat running t times, and then choose the cut with the smallest size will improve correctness probability

- What will be the modified probability?

Randomized Min-Cut
(Repeated Runs)

- After t runs, the algorithm is wrong with prob at most $\left(1 - \frac{2}{n(n-1)}\right)^t$
- Using the fact $1 - x \leq e^{-x}$ [for any x],

$$\left(1 - \frac{2}{n(n-1)}\right)^t \leq e^{-\frac{2t}{n(n-1)}}$$

- Setting $t = n(n-1) \log_e n$, the algorithm is wrong after t runs with prob $\leq 1/n^2$

Randomized Min-Cut
(Repeated Runs)

Conclusion:

- After $n(n-1)\log_e n$ runs, our algorithm returns min-cut with high probability
  (This means: At least $1 - 1/n^c$ for some constant c)
- With careful implementation, each run takes $O(n^2)$ time

➔ For $n(n-1)\log_e n$ runs $= O(n^4 \log n)$ time

- Not so good… Deterministic algorithm only needs $O(\ nm + n^2 \log n\ )$ time !!!

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

Observation: We will wrongly contract an edge of C more easily at later steps

- What if we just run randomized min-cut until G' contains $n/\sqrt{2}$ vertices?

  ➔ prob that no edge of C is contracted $\geq \dfrac{(n/\sqrt{2})(n/\sqrt{2}-1)}{n(n-1)}$, which is very close to 1/2 [for simplicity, we assume this $\geq$1/2]

Randomized Min-Cut
(Karger and Stein's Speed Up [1993])

- We use

$$\text{Contract}(X)$$

to denote the graph after running randomized min-cut on X until it contains $|X|/\sqrt{2}$ vertices

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

The modified algorithm is as follows:

```
NewCut(G) {
    if (|G| == 2)  return edges of G;
    G1 = Contract(G),  G2 = Contract(G);
    Y1 = NewCut(G1),  Y2 = NewCut(G2);
    return min { Y1, Y2 }
}
// Remark:  G1 and G2 are normally not the same
//          since output of Contract() is random
```

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

Questions:

- What is the runtime of Karger and Stein's algorithm?

  Ans. $T(n) = O(n^2) + 2T(n/\sqrt{2}\,)$

  ➔ By Master Theorem, $T(n) = O(n^2 \log n)$

- When will NewCut(G) return min-cut C?

  Ans. ...when either (i) G_1 contains C and NewCut(G_1) returns C, or (ii) G_2 contains C and NewCut(G_2) returns C

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

- We now express the probability that NewCut(G) returns C, in terms of n
- Let $f(x)$ denote the minimum probability that C is returned if NewCut( ) is run on any graph of x vertices (produced by some edge contractions from G) with C = one of its min-cut

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

Thus,

Pr(G_1 contains C and NewCut(G_1) returns C)

$=$ Pr(NewCut(G_1) returns C | G_1 contains C)

$\times$ Pr(G_1 contains C)

$\geq$ $f(n/\sqrt{2})$ (1/2)

So, [why?]

Pr(NewCut(G) not return C) $\leq \left(1-\tfrac{1}{2}f(n/\sqrt{2})\right)^2$

Randomized Min-Cut
(Karger and Stein's Speed Up [1993])

In other words, [why?]

$$f(n) \geq 1 - \left(1-\tfrac{1}{2}f(n/\sqrt{2}\,)\right)^2$$

In general, we have

$$f(x) \geq 1 - \left(1-\tfrac{1}{2}f(x/\sqrt{2}\,)\right)^2 \quad \text{for } x \geq 3$$

$$f(2) = 1$$

Solving the recurrence, we get [trust me]

$$f(n) \geq 1/\log n$$

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

- Thus, new algorithm is correct with prob at least $1/\log n$

  ➔ Much better than just $2/n(n-1)$
- Repeat for $2(\log n)(\log_e n)$ times, prob of not returning C is at most

  $(1 - 1/\log n)^{2(\log n)(\log_e n)} \leq e^{-2\log_e n} = 1/n^2$

Randomized Min-Cut

(Karger and Stein's Speed Up [1993])

Conclusion:

- New algorithm returns min-cut after $2(\log n)(\log_e n)$ runs, with high probability
- For $2(\log n)(\log_e n)$ runs = $O(n^2 \log^3 n)$ time
  - Better than deterministic algorithm [which needs $O(\ nm + n^2 \log n\ )$ time] ... when the graph is dense (i.e., when $m = \Theta(n^2)$)

A very useful inequality

Lemma: $1 + x \leq e^x$ (for any x)

1st Proof:

Study by cases ($x > 0$, $x < -1$, others)

Use the fact: $e^x = 1 + x + x^2/2! + x^3/3! + \ldots$

2nd Proof:

Compare the curve: $y = 1+x$ and $y = e^x$

Corollary: $1 - x \leq e^{-x}$ (for any x)

An interesting application: The Birthday Pairing Problem

Let's say we have a class of N people, born in the year of 1989.

With no ideas about their birthdays, we assume each person chooses his birthday uniformly at random from the 365 days

What is the probability that we can find two persons born on the same day?

An interesting application: The Birthday Pairing Problem

- If N = 50, will the previous probability be greater than 0.5?

  Ans. Greater than 0.965

- If we want the previous probability to be greater than 0.5, how large should N be?

  Ans. N = 23 will be sufficient…