Randomized Algorithms

input is still worst-case. But algorithm uses randomness.

1) significantly simpler algorithms
2) faster (improved polynomial)
3) some applications are impossible without it
   e.g. cryptography (no security without randomness)

Main con:

output of the randomized algorithm is a random variable.
It will be incorrect with some probability.

But: can drive this error probability down to tinier than 2^{-100} easily.

We will first look at 2 examples, then talk a little more about randomized algorithms in general

MAX-CUT

Input: $G(V,E)$ $|V|=n$, $|E|=m$

Goal: Find $S \subseteq V$ s.t. $E(S,\bar{S}) = \{\{u,v\} \mid u \in S, v \in \bar{S}\}$ has largest possible size

Ideas? NP-hard!

Proposition: There is a randomized algorithm that outputs a cut S s.t.

$$\mathbb{E}[|cut(S,\bar{S})|] = \frac{m}{2}$$

↓ over the random choices of the algo (NOT over the input)

Tip: in applications, we don't usually explicitly specify the sample space of an random "experiment". But it's a good idea to ask yourself what it is.

Algorithm: (Imagine a "rand" function that outputs a uniform bit)

1) For every vertex $v \in V$, choose a uniformly random & indep bit $b_v \in \{0,1\}$. (I'll often use the phrase "toss a fair coin" equivalently)

random variable

2) Output $S = \{v \mid b_v = 1\}$.

(In english, one would describe this as "return a random cut")

This is a randomized algo.

Input graph is arbitrary.

Let S be the cut output by the algorithm.

Then S takes values in $2^{[V]}$.

what's the sample space?

Let $X = |S|$. Then X is a random var.

Analysis

Lemma: $\mathbb{E}[X] = \frac{m}{2}$.

Proof: Important trick: try to write your r.v. as a sum of natural indicator r.v.s.

$$X_{\{i,j\}} = \begin{cases} 1 & \text{if edge } \{i,j\} \in \text{cut}(S,\bar{S}) \\ 0 & \text{o/w.} \end{cases}$$

Then $X = \sum_{\{i,j\} \in E} X_{\{i,j\}}$.

By linearity of expectation,

$$\mathbb{E}[X] = \sum_{\{i,j\} \in E} \mathbb{E}[X_{\{i,j\}}].$$

$$\mathbb{E}[X_{\{i,j\}}] = \Pr[i \in S, j \notin S] + \Pr[i \notin S, j \in S]$$
$$= 1/4 + 1/4 = 1/2$$

□

This says that on average the algorithm outputs a cut with $\frac{1}{2}$ of all possible edges

On average is not that great.

Idea: Independent Repetition Breeds Resilience

Algo:
1) Run Algo k times independently.
2) Output the largest cut.

Lemma: $\Pr\left[X \leq \frac{m}{2}(1-\varepsilon)\right]$

$= \Pr\left[\bar{X} > \frac{m}{2}(1+\varepsilon)\right] \leq \frac{1}{1+\varepsilon}$

Chance that all k cuts $\leq \frac{m}{2}(1-\varepsilon)$:

thus at most $\frac{1}{(1+\varepsilon)^k} \sim e^{-\varepsilon k} < e^{-100}$.

Choose $k = \frac{1}{\varepsilon} 100$.

In a later class, I'll show you how to get a "derandomization".

MIN CUT

Input: $G(V, E)$

Goal: Find $S \neq \emptyset$ or V (non-trivial) s.t. $|E(S, \bar{S})|$ is minimized.

Can you think of an algorithm to solve this?

Reduce to n^2 · MaxFlows. So: $n^2 \cdot m^2 \cdot n = m^2 \cdot n^3$

And, more than that, fairly complicated.

Today: A beautiful, simple randomized algorithm

Def (Contracting an edge).

Input: $G(V, E)$, edge $\{i, j\}$.

Output: G', V' = Vertex set: $(V \setminus \{i, j\}) \cup \underbrace{s_{ij}}_{\text{Supernode}}$

E' = re-route all edges incident on i or j to s_{ij}.

Keep parallel edges!

After a contraction,

vertices : -1

edges : If no parallel edges, -1
o/w # edges parallel to $\{i, j\}$ the contracted edge.

Algorithm $G(V, E)$

Repeat until there are 2 super-vertices left

1) Choose a uniformly random edge.

2) Contract it.

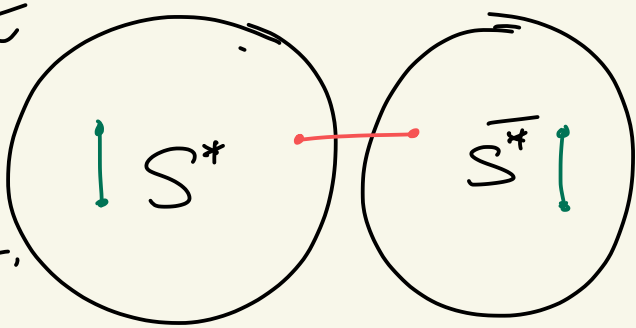
Output the associated cut.

Each supervertex stands for a subset of vertices. So at the end, the 2 super vertices must stand for $S, \bar{S}$ for some $S \subseteq V$.

So, the algorithm computes a cut.

Question: what's the probability that the algo outputs a min cut?

- If we never contract an edge in a min-cut, the output must be that min-cut.



Intuition: min-cut has small # edges.

So the chance we contract it must be low.

Fix a min cut $(S^*, \overline{S^*})$.

Q: What's the chance that the 1st edge chosen is in the min-cut?

Lemma: Let c be the size of any min-cut. Then, G has $m \geq \frac{n \cdot c}{2}$ edges.

Pf: The degree of every vertex $\geq c$ (as otherwise $\{v\} \mid V \setminus \{v\}$ is a cut with size $< c$)

So: $d_1, d_2, \ldots, d_n \geq c$

$\#\text{edges} = \sum_i d_i/2 \geq n \cdot c/2$.

Lemma: Chance that the 1st edge is in the min-cut $\leq 2/n$. Conditioned on 1st edge not being in the min-cut, chance that 2nd edge is not in the min-cut is $\leq 2/n-1 \ldots$

Lemma: $\Pr[\text{min-cut } S^* \text{ is output}] \geq \frac{2}{n \cdot (n-1)}$

Pf: $\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\ldots\left(1-\frac{2}{3}\right)$

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

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

So., with probability $\geq \frac{2}{n(n-1)} \geq \frac{2}{n^2}$,

we output $S^* \rightarrow$ a min-cut.

That sounds low...

Idea: Repetition builds resilience

Alg:

Run Basic Karger for k times

Output the smallest of the k cuts returned.

Analysis:

$\Pr[$ iteration i fails to output a min-cut $] \leq \left(1 - \frac{2}{n^2}\right)$

$\Pr[$ all k iterations fail $] \leq \left(1 - 2/n^2\right)^k$

$\leq e^{-\frac{2k}{n^2}}$

$\leq e^{-100}$ if

$k \geq \frac{n^2}{2} \cdot 100.$

So, $\sim n^2$ iterations suffice.

Runtime: each iteration of Karger takes $\sim O(n^2)$ time.

So: $O(n^4)$... SAD

[Karger-Stein 1996]