So far: Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.
  - Solve problem to optimality.
  - Solve problem in poly-time.
  - Solve arbitrary instances of the problem.

$\rho$-approximation algorithm.
  - Guaranteed to run in poly-time.
  - Guaranteed to solve arbitrary instance of the problem
  - Guaranteed to find solution within ratio $\rho$ of true optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!
We’ve seen

Approximation algorithms
1. Load balancing
2. Center selection
3. Set cover
4. Vertex cover
5. Disjoint paths problem
6. Vertex cover with linear programming
7. Load balancing with linear programming
8. Arbitrarily good approximations
Today we start Randomization

Algorithmic design patterns.
- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.
Why randomize

• Underlying model more powerful
  • Efficient deterministic algorithms that always yield the correct answer
  • Efficient randomized algorithms that yield the correct answer with high probability
  • Randomized algorithms that always yield the correct answer and are expected to run efficiently with high probability
• Conceptually much simpler algorithms
• Very little internal memory or state
• Distributed systems
  • Reduce communication or synchronization
Today

Randomized algorithms

- Probabilities
- Contention resolution algorithms
- Finding the global minimum cut
Finite Probability Spaces 1/2

Sample space $\Omega$
- Possible outcomes of process under consideration

Probability (mass) $p(i)$
- Associated to every point $i \in \Omega$, non-negative
- $\sum_{i \in \Omega} p(i) = 1$

Event $E$
- Any subset of $\Omega$
- Defined by the set of outcomes that constitute it
- Probability of the event

$$
\Pr[E] = \sum_{i \in E} p(i)
$$
Finite Probability Spaces 2/2

When $p(i) = p(j), \forall i, j \in \Omega$

$$\Pr[E] = \frac{|E|}{|\Omega|}$$

Complementary event

$$\overline{E} = \Omega \setminus E$$

$$\Pr[\overline{E}] = 1 - \Pr[E]$$
Fair and biased coins

Points in sample space and their probability -> complete description of the system
- We are interested in computing the probabilities of the events
  - Subsets in sample space

Fair coin flipping
- $\Omega=$\{heads, tails\}, $p(\text{heads})=p(\text{tails})=1/2$

Biased coin flipping
- $p(\text{heads})=2/3$, $p(\text{tails})=1/3$
- Heads twice as likely as tails
Process naming 1/2

- Processes $p_1, \ldots, p_n$
- Each chooses name
  - At random from the space of all $k$-bit strings
  - Concurrently
- Names $\in \{0,1,2,\ldots,2^k-1\}$
  - Numerical value of name in binary
- $\Omega = \text{all } n\text{-tuples of integers, each integer between } 0 \text{ and } 2^k-1$
  - $|\Omega| = (2^k)^n = 2^{kn}$, each with probability $2^{-kn}$
Process naming 2/2

- Event $E$: processes $p_1$ and $p_2$ choose the same name
- $E \subseteq \Omega$, contains all n-tuples with first two coordinates identical
- $|E| = 2^{k(n-1)}$
- Hence

$$\Pr[E] = \sum_{i \in E} p(i) = 2^{k(n-1)} \cdot 2^{-kn} = 2^{-k}$$
Conditional probability

Conditional probability. E, F events; need to know the likelihood of E when also F happens

\[ \Pr[E | F] = \frac{\Pr[E \cap F]}{\Pr[F]}, \quad \Pr[F] > 0 \]

- Usage
  - Events \( F_1, ..., F_k \) partition \( \Omega \), \( \Pr[F_j] > 0 \), \( \sum_{j=1}^{k} \Pr[F_j] = 1 \)
    - Each point in the sample space is in exactly one \( F_i \)
    - We know \( \Pr[F_j] \) and \( \Pr[E | F_j] \), \( \forall j = 1, ..., k \)
  - Then we get

\[
\sum_{j=1}^{k} \Pr[E | F_j] \cdot \Pr[F_j] = \sum_{j=1}^{k} \frac{\Pr[E \cap F_j]}{\Pr[F_j]} \cdot \Pr[F_j] = \sum_{j=1}^{k} \Pr[E \cap F_j] = \Pr[E]
\]
Independent events

Intuition. 2 events independent if information about outcome of one does not affect our estimate of the likelihood of the other

Concretely: \( \Pr[E|F] = \Pr[E] \) and \( \Pr[F|E] = \Pr[F] \)

Obs. \( \Pr[E|F] = \Pr[E] \Rightarrow \Pr[F|E] = \Pr[F] \)

Definition. Events \( E \) and \( F \) are independent if 
\[
\Pr[E \cap F] = \Pr[E] \times \Pr[F].
\]
Events \( F_1, \ldots, F_k \) are independent if
\[
\Pr[\bigcap_{i=1}^{k} F_i] = \prod_{i=1}^{k} \Pr[F_i]
\]
Checking independence

Example. Three fair coins, $E_i$ the event that coin $i$ comes up "heads" => $\Pr[E_i]=1/2$, $i=1,2,3$.

$A$ - event that coins 1 and 2 have the same value
$B$ - event that coins 2 and 3 have the same value
$C$ - event that coins 1 and 3 have different values

$$\Omega=\{(h_1,h_2,h_3), (h_1,h_2,t_3), (h_1,t_2,h_3), (h_1,t_2,t_3), (t_1,h_2,h_3), (t_1,h_2,t_3), (t_1,t_2,h_3), (t_1,t_2,t_3)\}$$ =>

$\Pr[A]=\Pr[B]=\Pr[C]=1/2$ and $\Pr[A \cap B]= \Pr[A \cap C]=\Pr[C \cap B]=1/4$ but $\Pr[A \cap B \cap C]=0$
The union bound

- Events $F_1, \ldots, F_k$, we need the probability that ANY of them happens:

$$
\Pr[\bigcup_{i=1}^{n} E_i]
$$

- If $F_i \cap F_j = \emptyset$, $\forall i, j = 1, \ldots, k$ then

$$
\Pr[\bigcup_{i=1}^{k} F_i] = \sum_{i=1}^{k} \Pr[F_i]
$$

- In general

$$
\Pr[\bigcup_{i=1}^{k} F_i] \leq \sum_{i=1}^{k} \Pr[F_i]
$$
Union bound and randomized algorithms

Assume randomized algorithm produces correct result with high probability

- Bad events $F_1, ..., F_k$: if none happens, then correct answer
- $F$ - the event that the algorithm fails $\Rightarrow \Pr[F] \leq \Pr[\bigcup_{i=1}^{k} F_i]$

$\Rightarrow \Pr[F] \leq \Pr[\bigcup_{i=1}^{k} F_i] \leq \sum_{i=1}^{k} \Pr[F_i]$

- If each bad event has small probability $\Rightarrow$ their sum small
- Decompose complicated event (algorithm failure) into numerous small events of computable probability
Process naming again

- **n=1000, k=32**
  - 1000 processes, each chooses a 32-bit name
- **Two of them choose the same name: how unlikely that is?**
- **F** - event that two processes choose same name =>
  \[
  F = \bigcup_{i<j} E_{ij}
  \]
  where \( E_{ij} \) - event that \( p_i \) and \( p_j \) choose same name

- There are \( \binom{1000}{2} \) events \( E_{ij} \) =>
  \[
  \Pr[F] \leq \sum_{i,j} \Pr[E_{ij}] = \binom{1000}{2} \cdot 2^{-32}
  \]

- \( \binom{1000}{2} \approx 0.5 \times 10^6 \), \( 2^{32} \approx 4 \times 10^9 \) => \( \Pr[F] \approx 0.000125 \)
Contention resolution
Contention Resolution in a Distributed System

Contestation resolution. Given n processes $P_1, \ldots, P_n$, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can't communicate.

Challenge. Need symmetry-breaking paradigm.
Contention Resolution: Randomized Protocol

**Protocol.** Each process requests access to the database at time \( t \) with probability \( p = \frac{1}{n} \).

**Claim.** Let \( S[i, t] = \) event that process \( i \) succeeds in accessing the database at time \( t \). Then \( \frac{1}{(e \cdot n)} \leq \Pr[S(i, t)] \leq \frac{1}{2n} \).

**Pf.** By independence, \( \Pr[S(i, t)] = p \cdot (1-p)^{n-1} \).

- Setting \( p = \frac{1}{n} \), we have \( \Pr[S(i, t)] = \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1} \).

**Useful facts from calculus.** As \( n \) increases from 2, the function:
  - \((1 - \frac{1}{n})^n\) converges monotonically from \(1/4\) up to \(1/e\)
  - \((1 - \frac{1}{n})^{n-1}\) converges monotonically from \(1/2\) down to \(1/e\).
Contention Resolution: Randomized Protocol

**Claim.** The probability that process i fails to access the database in n rounds is at most 1/e. After $e \cdot n(c \ln n)$ rounds, the probability is at most $n^{-c}$.

**Pf.** Let $F[i, t] = \text{event that process } i \text{ fails to access database in rounds 1 through } t$. By independence and previous claim, we have

$$Pr[F(i, t)] \leq \left(1 - \frac{1}{en}\right)^t.$$

- **Choose** $t = \lceil e \cdot n \rceil$:
  $$Pr[F(i, t)] \leq \left(1 - \frac{1}{en}\right)^{en} \leq \left(1 - \frac{1}{en}\right)^{en} \leq \frac{1}{e}$$

- **Choose** $t = \lceil e \cdot n \rceil \lceil c \ln n \rceil$:
  $$Pr[F(i, t)] \leq \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$$
Contention Resolution: Randomized Protocol

**Claim.** The probability that all processes succeed within $2e \cdot n \ln n$ rounds is at least $1 - 1/n$.

**Pf.** Let $F[t]$ = event that at least one of the $n$ processes fails to access database in any of the rounds 1 through $t$.

$$\Pr[F[t]] = \Pr\left[\bigcap_{i=1}^{n} F[i, t]\right] \leq \sum_{i=1}^{n} \Pr[F[i, t]] \leq n \left(1 - \frac{1}{en}\right)^t$$

Choosing $t = 2 \lceil en \rceil \lceil \ln n \rceil$ yields $\Pr[F[t]] \leq n \cdot n^{-2} = 1/n$. 

Union bound. Given events $E_1, \ldots, E_n$, 

$$\Pr\left[\bigcap_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i]$$
Global minimum cut
Global Minimum Cut

Global min cut. Given a connected, undirected graph $G = (V, E)$ find a cut $(A, B)$ of minimum cardinality.

Applications. Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

Network flow solution.
- Replace every edge $(u, v)$ with two antiparallel edges $(u, v)$ and $(v, u)$.
- Pick some vertex $s$ and compute min $s$-$v$ cut separating $s$ from each other vertex $v \in V$.

False intuition. Global min-cut is harder than min $s$-$t$ cut.
Contraction Algorithm

**Contraction algorithm.** [Karger 1995]

- Pick an edge $e = (u, v)$ uniformly at random.
- **Contract** edge $e$.
  - replace $u$ and $v$ by single new super-node $w$
  - preserve edges, updating endpoints of $u$ and $v$ to $w$
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes $v_1$ and $v_2$.
- Return the cut (all nodes that were contracted to form $v_1$).
Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Pf. Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$. Let $k = |F^*| = $ size of min cut.

- In first step, algorithm contracts an edge in $F^*$ probability $k / |E|$.
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be min-cut. $\Rightarrow |E| \geq \frac{1}{2}kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$. 

![Diagram of contraction algorithm](image)
Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

\textbf{Pf.} Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$. Let $k = |F^*| = \text{size of min cut}$.

- Let $G'$ be graph after $j$ iterations. There are $n' = n-j$ (super)nodes.
- Suppose no edge in $F^*$ has been contracted. The min-cut in $G'$ is still $k$.
- Since value of min-cut is $k$, $|E'| \geq \frac{1}{2}kn'$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n'$.
- Let $E_j = \text{event that an edge in } F^* \text{ is not contracted in iteration } j$.

$$
\Pr[E_1 \cap E_2 \square \cap E_{n-2}] = \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \square \times \Pr[E_{n-2} \mid E_1 \cap E_2 \square \cap E_{n-3}]
\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})\square (1 - \frac{2}{4})(1 - \frac{2}{3})
= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\square \left(\frac{2}{4}\right)\left(\frac{1}{3}\right)
= \frac{2}{n(n-1)}
\geq \frac{2}{n^2}
$$
Contraction Algorithm

Amplification. To amplify the probability of success, run the contraction algorithm many times.

Claim. If we repeat the contraction algorithm \(n^2 \ln n\) times with independent random choices, the probability of failing to find the global min-cut is at most \(1/n^2\).

Pf. By independence, the probability of failure is at most

\[
(1 - 1/x)^x \leq 1/e
\]

\[
\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} = \left(\left(1 - \frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right)^{2\ln n} \leq \left(e^{-1}\right)^{2\ln n} = \frac{1}{n^2}
\]
Claim. An undirected graph $G=(V,E)$ with $n$ nodes has at most $n(n-1)/2$ global min-cuts.

Proof. Let $C_1, \ldots, C_r$ be all the min-cuts in $G$.

- $E_i$ = event that $C_i$ returned by our algorithm
- $E = \bigcup_{i=1}^r E_i$ is the event that the algorithm returns any global min-cut
- Previous slide $\Rightarrow \Pr[E] \geq 2/n(n-1)$ and also $\Pr[E_i] \geq 2/n(n-1)$
- $E_i$ and $E_j$ are independent (only one cut returned by any given run of the algorithm) $\Rightarrow$

$$1 \geq \Pr[E] = \Pr\left[\bigcup_{i=1}^r E_i\right] = \sum_{i=1}^r \Pr[E_i] \geq \frac{2r}{n(n-1)} \implies r \leq \frac{n(n-1)}{2}$$
Global Min Cut: Context

Remark. Overall running time is slow since we perform $\Theta(n^2 \log n)$ iterations and each takes $\Omega(m)$ time.

Improvement. [Karger-Stein 1996] $O(n^2 \log^3 n)$.

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm until $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.