13.2 Global Minimum Cut

Global Minimum 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 $v \in V$.
- Need to compute $O(n)$ minimum $s$-$v$ cuts in total.

False Intuition: Global min-cut is harder than min $s$-$t$ cut.

Contraction Algorithm

Claim: The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Proof:
- Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$, let $k = |F^*| = \text{size of min cut}$.
- In first step, algorithm contracts an edge in $F^*$ with probability $k/|E|$. 
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be a min-cut. $\Rightarrow |E| \geq kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.

Contraction Algorithm

Claim: The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Proof:
- Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$, let $k = |F^*| = \text{size of min cut}$.
- In first step, algorithm contracts an edge in $F^*$ with probability $k/|E|$. 
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be a min-cut. $\Rightarrow |E| \geq kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.

Contraction Algorithm

Claim: The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Proof:
- Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$, let $k = |F^*| = \text{size of min cut}$.
- In first step, algorithm contracts an edge in $F^*$ with probability $k/|E|$. 
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be a min-cut. $\Rightarrow |E| \geq kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.

Contraction Algorithm

Claim: The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Proof:
- Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$, let $k = |F^*| = \text{size of min cut}$.
- In first step, algorithm contracts an edge in $F^*$ with probability $k/|E|$. 
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be a min-cut. $\Rightarrow |E| \geq kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.
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

\[
1 - \left(1 - \frac{1}{2^n}ight)^{n^2 \ln n} = \frac{1}{n^2}
\]

Global Min Cut: Context

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

**Extensions.** Naturally generalizes to handle positive weights.

**Best known.** [Karger 2000]: \( O(m \log^3 n) \).

13.3 Linearity of Expectation

**Expectation: Two Properties**

**Useful property.** If \( X \) is a \( \{0,1\} \) random variable, 

\[
E[X] = P[X = 1]
\]

**Pf.**

\[
E[X] = \sum_{i=1}^{\infty} i \cdot P[X = i] = \sum_{i=1}^{\infty} i \cdot (1-p)^{i-1} p = \frac{1}{p} - \frac{1}{p^2} = \frac{1}{1-p}
\]

Fact: \( \sum_{i=0}^{\infty} (1-p)^i = \frac{1}{p} \)

**Linearity of expectation.** Given two random variables \( X \) and \( Y \) defined over the same probability space, \( E[X + Y] = E[X] + E[Y] \).

Decouples a complex calculation into simpler pieces.

Expectation: Given a discrete random variable \( X \), its expectation \( E[X] \) is defined by:

\[
E[X] = \sum_{i=1}^{\infty} i \cdot P[X = i]
\]

**Waiting for a first success.** Coin is heads with probability \( p \) and tails with probability \( 1-p \). How many independent flips \( X \) until first heads?

\[
E[X] = \sum_{i=1}^{\infty} i \cdot P[X = i] = \sum_{i=0}^{\infty} (1-p)^i \cdot 1 = \frac{1}{1-p} = \frac{1}{p}
\]

**Fact:** \( \sum_{i=0}^{\infty} (1-p)^i = \frac{1}{p} \)

Guessing Cards

**Game.** Shuffle a deck of \( n \) cards; turn them over one at a time; try to guess each card.

**Memoryless guessing.** No psychic abilities; can’t even remember what’s been turned over already. Guess a card from full deck uniformly at random.

**Claim.** The expected number of correct guesses is 1.

**Pf.** (surprisingly effortless using linearity of expectation)

- Let \( X_i = 1 \) if \( i \)th prediction is correct and 0 otherwise.
- Let \( X = \sum_{i=1}^{n} X_i \) be number of correct guesses.
- \( E[X] = P[X = 1] = 1/n \)
- \( E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1 \)

**Linearity of expectation**
13.4 MAX 3-SAT

Claim. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is \(7k/8\).

**Pf.** Consider random variable \(Z_j\) defined as:

\[Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}\]

Let \(Z = \text{weight of clauses satisfied by assignment } Z_j\).

By linearity of expectation:

\[E[Z] = \sum_{j=1}^{k} E[Z_j] = \sum_{j=1}^{k} \text{Pr}[\text{clause } C_j \text{ is satisfied}] = \frac{7}{8} k\]

Maximum 3-Satisfiability: Analysis

**Claim.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a 7/8 fraction of all clauses.

**Pf.** Consider random variable \(Z_j\) defined as:

\[Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}\]

Let \(Z = \text{weight of clauses satisfied by assignment } Z_j\).

By linearity of expectation:

\[E[Z] = \sum_{j=1}^{k} E[Z_j] = \sum_{j=1}^{k} \text{Pr}[\text{clause } C_j \text{ is satisfied}] = \frac{7}{8} k\]

The Probabilistic Method

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a 7/8 fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time.

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \(\frac{1}{2}\), independently for each variable.

The Probabilistic Method

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a 7/8 fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time.

**Probabilistic method.** We showed the existence of a non-obvious property of 3-SAT by showing that a random construction produces it with positive probability!
Maximum 3-Satisfiability: Analysis

Q. Can we turn this idea into a 7/8-approximation algorithm? In general, a random variable can almost always be below its mean.

Lemma. The probability that a random assignment satisfies \(7k/8\) clauses is at least 1/(8k).

Proof. Let \(p_j\) be probability that exactly \(j\) clauses are satisfied; let \(p\) be probability that \(\geq 7k/8\) clauses are satisfied.

Rearranging terms yields \(p \geq 1 / (8k)\).

Maximum Satisfiability

Extensions
- Allow one, two, or more literals per clause.
- Find max weighted set of satisfied clauses.

Theorem. [Asano-Williamson 2000] There exists a 0.784-approximation algorithm for MAX-SAT.

Theorem. [Karloff-Zwick 1997, Zwick+computer 2002] There exists a 7/8-approximation algorithm for version of MAX-3SAT where each clause has at most 3 literals.

Theorem. [Ivaldi 1997] Unless P = NP, no \(\rho\)-approximation algorithm for MAX-3SAT (and hence MAX-SAT) for any \(\rho > 7/8\).

Monte Carlo vs. Las Vegas Algorithms

Monte Carlo algorithm. Guaranteed to run in poly-time, likely to find correct answer.

Las Vegas algorithm. Guaranteed to find correct answer, likely to run in poly-time.

Remark. Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

13.5 Randomized Divide-and-Conquer

RP and ZPP

RP. [Monte Carlo] Decision problems solvable with one-sided error in poly-time.

ZPP. [Las Vegas] Decision problems solvable in expected poly-time.

Theorem. P \(\subseteq\) ZPP \(\subseteq\) RP \(\subseteq\) NP.

Quicksort

Sorting. Given a set of n distinct elements S, rearrange them in ascending order.

```plaintext
RandomizedQuicksort(S) {
    if |S| = 0 return
    choose a splitter aᵢ uniformly at random
    foreach (a ∈ S) {
        if (a < aᵢ) put a in S-
        else if (a > aᵢ) put a in S+
    }
    RandomizedQuicksort(S-)
    output aᵢ
    RandomizedQuicksort(S+)
}
```

**Remark.** Can implement in-place.

**Running time.**

- **[Best case.]** Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons.
- **[Worst case.]** Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons.

**Randomize.** Protect against worst case by choosing splitter at random.

**Intuition.** If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $O(n \log n)$ comparisons.

**Notation.** Label elements so that $x_1 < x_2 < \ldots < x_n$.

13.6 Universal Hashing
Dictionary Data Type

Dictionary. Given a universe $U$ of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in $S$ is efficient.

Dictionary interface.
1. Create(): Initialize a dictionary with $S = \emptyset$.
2. Insert(u): Add element $u \in U$ to $S$.
3. Delete(u): Delete $u$ from $S$, if $u$ is currently in $S$.
4. Lookup(u): Determine whether $u$ is in $S$.

Challenge. Universe $U$ can be extremely large so defining an array of size $|U|$ is infeasible.

Applications. File systems, databases, Google, compilers, checksums, P2P networks, associative arrays, cryptography, web caching, etc.

Hashing

Hash function. $h : U \to \{0, 1, \ldots, n-1\}$.

Hashing. Create an array $H$ of size $n$. When processing element $u$, access array element $H[h(u)]$.

Collision. When $h(u) = h(v)$ but $u \neq v$.
- A collision is expected after $\Theta(n)$ random insertions. This phenomenon is known as the "birthday paradox."
- Separate chaining: $H\[i\]$ stores linked list of elements $u$ with $h(u) = i$.

Algorithmic Complexity Attacks

When can’t we live with ad hoc hash function?
- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

Real world exploits. [Crosby-Wallach 2003]
- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.
Universal Hashing

Universal hashing property. Let \( H \) be a universal class of hash functions; let \( h \in H \) be chosen uniformly at random from \( H \) and let \( u \in U \). For any subset \( S \subseteq U \) of size at most \( n \), the expected number of items in \( S \) that collide with \( u \) is at most 1.

Pf. For any element \( s \in S \), define indicator random variable \( X_s = 1 \) if \( h(s) = h(u) \) and 0 otherwise. Let \( X \) be a random variable counting the total number of collisions with \( u \).

\[
E_{h \in H}[X] = \sum_{s \in S} P(X_s = 1) = \sum_{s \in S} \Pr[h(s) = h(u)] = |S| \leq 1
\]

(definition of expectation \( X_s \) is a 0-1 random variable)

Designing a Universal Family of Hash Functions

Theorem. There exists a prime between \( n \) and \( 2n \).

Modulus. Choose a prime number \( p \approx n \).

Integer encoding. Identify each element \( u \in U \) with a base-\( p \) integer of \( r \) digits: \( x = (x_1, x_2, \ldots, x_r) \).

Hash function. Let \( A = \{ \text{all } r \text{-digit, base-} p \text{ integers} \} \). For each \( a = (a_1, a_2, \ldots, a_r) \) where \( 0 \leq a_i < p \), define

\[
h_a(x) = \sum_{i=1}^{r} a_i x_i \mod p
\]

Hash function family. \( H = \{ h_a : a \in A \} \).

13.9 Chernoff Bounds

Chernoff Bounds (above mean)

Theorem. Suppose \( X_1, \ldots, X_n \) are independent 0-1 random variables. Let \( X = X_1 + \cdots + X_n \). Then for any \( a \geq 0 \) and \( \delta > 0 \), we have

\[
\Pr[X > (1 + \delta)n] < e^{-\frac{\delta^2 n}{2}}
\]

Fact. Let \( p \) be prime, and let \( m \equiv 0 \mod p \). Then \( m \equiv \sqrt{-1} \mod p \) has at most one solution \( 0 < \alpha < p \).

Pf. Suppose \( \alpha \) and \( \beta \) are two different solutions. Then \( (\alpha - \beta) \equiv 0 \mod p \), hence \( (\alpha - \beta) \) is divisible by \( p \). Since \( p > 0 \mod p \), we know that \( \alpha \) is divisible by \( p \). This implies \( \alpha = \beta \).

Bonus fact. Can replace "at most one" with "exactly one" in above fact.

Pf idea. Euclid’s algorithm.
Chernoff Bounds (above mean)

**Pf. (cont)**

- Let $\mathbb{P}_i = \mathbb{P}[X_i = 1]$. Then,
  $\mathbb{E}[e^{tX_i}] = \mathbb{E}[e^t] \cdot \mathbb{E}[e^{tX_i}] = 1 + \mathbb{P}_i e^{t(1 - \mathbb{P}_i)} \leq e^{e^{t(1 - \mathbb{P}_i)}}$

- Combining everything:
  $\mathbb{P}(X > (1 + \delta)\mathbb{E}[X]) \leq e^{-t(1 + \delta)\mathbb{E}[X]} \leq e^{-t(1 + \delta)\mathbb{E}[X] e^{e^{t(1 - \mathbb{P}_i)}}}$

- Finally, choose $t = \ln(1 + \delta)$.

**13.10 Load Balancing**

**Analysis**

- Let $X_i$ be the number of jobs assigned to processor $i$.
- Let $Y_{ij} = 1$ if job $j$ assigned to processor $i$, and $0$ otherwise.
- We have $E[Y_{ij}] = 1/n$.
- Thus, $X_i = \sum_j Y_{ij}$ and $\mathbb{E}[X_i] = 1$.
- Applying Chernoff bounds with $\delta = 1 - \epsilon$ yields $\mathbb{P}(X_i > c) < e^{-\epsilon}$.

- Let $(\alpha)$ be number $x$ such that $x^x = n$, and choose $c = e^{\alpha(n)}$.

**Load Balancing: Many Jobs**

**Theorem.** Suppose the number of jobs $m = 16n \ln n$. Then on average, each of the $n$ processors handles $\mu = 4 \ln n$ jobs. With high probability, every processor will have between half and twice the average load.

**Pf.**

- Let $X_i$, $Y_{ij}$ be as before.
- Applying Chernoff bounds with $\delta = 1$ yields
  $\mathbb{P}(X_i > 2\mu) < \left(\frac{1}{2}\right)^{2^\alpha(\ln\alpha)} < \left(\frac{1}{2}\right)^{\ln\alpha} = \frac{1}{2^\alpha}$

- Union bound: with probability $\geq 1 - 1/n$, no processor receives more than $e^{\alpha(n)} = 4^{\ln n} = n^\ln 4$ jobs.
Extra Slides