13. RANDOMIZED ALGORITHMS

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

Randomization

Algorithmic design patterns.
- Greedy.
- 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, closest pair, Monte Carlo integration, cryptography, ...

Contention resolution in a distributed system

Contention 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 $p = 1/n$.

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

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

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

**Useful facts from calculus.** As $n$ increases from 2, the function:

- $(1 - 1/n)^n$ converges monotonically from 1/4 up to 1/e.
- $(1 - 1/n)^{n-1}$ converges monotonically from 1/2 down to 1/e.

Contention resolution: randomized protocol

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

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

$$\Pr[F[i, t]] = \Pr\left[\bigcup_{i=1}^{n} F[i, t]\right] \leq \sum_{i=1}^{n} \Pr[F[i, t]] \leq n \cdot \left(1 - \frac{1}{2n}\right)^t$$

- Choosing $t = 2 [e \cdot n] \cdot \ln n$ yields $\Pr[F[i, t]] \leq n \cdot n^{-2} = 1/n$.

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

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i]$$

13. RANDOMIZED ALGORITHMS

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Global minimum cut

Global min cut. Given a connected, undirected graph \( \overline{G} = (V, E) \), 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 \(s-v\) cut separating \(s\) from each other node \(v \in V\).

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 \(u_1\) and \(v_1\).
- 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 \(k = |F^*| = \text{size of min cut.}\)
- 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 a min-cut \(\Rightarrow |E| \geq \frac{1}{2}k n \Leftrightarrow k / |E| \leq 2 / n\).
- Thus, algorithm contracts an edge in \(F^*\) with probability \(\leq 2 / n\).
Contraction algorithm

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

**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 $n' = n - j$ supernodes.
- Suppose no edge in $F^*$ has been contracted. The min-cut in $G'$ is still $k$.
- Since value of min-cut is $k$, $|E'| \geq \frac{1}{2} k n' \iff k / |E'| \leq 2 / n'$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2 / n'$.
- Let $E_j$ = event that an edge in $F^*$ is not contracted in iteration $j$.

$$
\Pr[E_1 \cap E_2 \cap \ldots \cap E_{n-2}] = \Pr[E_1] \times \Pr[E_2 | E_1] \times \ldots \times \Pr[E_{n-2} | E_1 \cap E_2 \cap \ldots \cap E_{n-3}]
= \left(1 - \frac{2}{n}ight) \left(1 - \frac{2}{n'}\right) \ldots \left(1 - \frac{2}{n-1}\right)
= \frac{2}{n}
\geq \frac{1}{n^2}
$$

Contraction algorithm: example execution

trial 1
trial 2
trial 3
trial 4
trial 5
(finds min cut)
trial 6

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] $O(n^3 \log^3 n)$.
- 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] $O(m \log^2 n)$,

faster than best known max flow algorithm or deterministic global min cut algorithm
13. **Randomized Algorithms**

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

**Expectation**

**Expectation.** Given a discrete random variable $X$, its expectation $E[X]$ is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \cdot Pr[X = j]$$

**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?

$$E[X] = \sum_{j=0}^{\infty} j \cdot Pr[X = j] = \sum_{j=0}^{\infty} j (1-p)^{j-1} p = \frac{p}{1-p} \sum_{j=0}^{\infty} j (1-p)^j = \frac{p}{1-p} \cdot \frac{1-p}{p^2} = \frac{1}{p}$$

**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 $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = Pr[X_i = 1] = 1/n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1$.

**Benefit.** Decouples a complex calculation into simpler pieces.
Guessing cards

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

Guessing with memory. Guess a card uniformly at random from cards not yet seen.

Claim. The expected number of correct guesses is $\Theta(\log n)$.

Pf.
- Let $X_i = 1$ if $i$th prediction is correct and 0 otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - (i - 1))$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = H(n)$.

\[
\ln(n+1) < H(n) < 1 + \ln n
\]

Coupon collector

Coupon collector. Each box of cereal contains a coupon. There are $n$ different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have $\geq 1$ coupon of each type?

Claim. The expected number of steps is $\Theta(n \log n)$.

Pf.
- Phase $j$ = time between $j$ and $j + 1$ distinct coupons.
- Let $X_j = \text{number of steps you spend in phase } j$.
- Let $X = \text{number of steps in total} = X_0 + X_1 + \ldots + X_{n-1}$.

\[
E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{j=0}^{n-1} \frac{1}{i} = n H(n)
\]

Maximum 3-satisfiability

Maximum 3-satisfiability. Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

\[
C_1 = x_3 \lor \overline{x_3} \lor \overline{x_4}
C_2 = x_2 \lor x_3 \lor \overline{x_4}
C_3 = \overline{x_1} \lor x_2 \lor x_4
C_4 = x_1 \lor x_2 \lor x_3
C_5 = x_1 \lor \overline{x_2} \lor \overline{x_4}
\]

Remark. NP-hard search problem.

Simple idea. Flip a coin, and set each variable true with probability $\frac{1}{2}$, independently for each variable.
Maximum 3-satisfiability: analysis

**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 \( Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases} \)

- Let \( Z = \) number of clauses satisfied by random assignment.

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

**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. •

**Probabilistic method.** [Paul Erdös] Prove the existence of a non-obvious property 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?
A. Yes (but a random variable can almost always be below its mean).

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

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

\[
\frac{7}{8} k = E[Z] = \sum_{j=0}^{\infty} j p_j
= \sum_{j<7k/8} j p_j + \sum_{j \geq 7k/8} j p_j
\leq \left(\frac{7k}{8} - \frac{1}{8}\right) \sum_{j<7k/8} p_j + k \sum_{j \geq 7k/8} p_j
\leq \left(\frac{7}{8} k - \frac{7}{8}\right) \cdot 1 + kp
\]

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

**Johnson’s algorithm.** Repeatedly generate random truth assignments until one of them satisfies \( \geq 7k/8 \) clauses.

**Theorem.** Johnson’s algorithm is a 7/8-approximation algorithm.

**Pf.** By previous lemma, each iteration succeeds with probability \( \geq 1/(8k) \). By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most \( 8k \). •
**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-3-SAT in which each clause has at most 3 literals.


---

**Monte Carlo vs. Las Vegas algorithms**

**Monte Carlo.** Guaranteed to run in poly-time, likely to find correct answer.
**Ex:** Contraction algorithm for global min cut.

**Las Vegas.** Guaranteed to find correct answer, likely to run in poly-time.
**Ex:** Randomized quicksort, Johnson’s MAX-3-SAT algorithm.

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

---

**RP and ZPP**

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

**One-sided error.**
- If the correct answer is no, always return no.
- If the correct answer is yes, return yes with probability ≥ ½.

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

**Theorem.** P ⊆ ZPP ⊆ RP ⊆ NP.

**Fundamental open questions.** To what extent does randomization help?
Does P = ZPP? Does ZPP = RP? Does RP = NP?

---

**13. RANDOMIZED ALGORITHMS**

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
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.
- $\text{create}()$: initialize a dictionary with $S = \emptyset$.
- $\text{insert}(u)$: add element $u \in U$ to $S$.
- $\text{delete}(u)$: delete $u$ from $S$ (if $u$ is currently in $S$).
- $\text{lookup}(u)$: is $u$ 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. $h : U \rightarrow \{0, 1, \ldots, n-1\}$.

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

Collision. When $h(u) = h(v)$ but $u \neq v$.
- A collision is expected after $\Theta(n)$ random insertions.
- Separate chaining: $a[i]$ stores linked list of elements $u$ with $h(u) = i$.

Ad-hoc hash function

Ad-hoc hash function.

```java
int hash(String s, int n) {
    int hash = 0;
    for (int i = 0; i < s.length(); i++)
        hash = (31 * hash) + s[i];
    return hash % n;
}  
```

Deterministic hashing. If $|U| \geq n^2$, then for any fixed hash function $h$, there is a subset $S \subseteq U$ of $n$ elements that all hash to same slot. Thus, $\Theta(n)$ time per lookup in worst-case.

Q. But isn’t ad-hoc hash function good enough in practice?

Algorithmic complexity attacks

When can’t we live with ad-hoc hash function?
- Obvious situations: aircraft control, nuclear reactor, pace maker, ....
- Surprising situations: denial-of-service (DOS) attacks.

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

**Ideal hash function.** Maps $m$ elements uniformly at random to $n$ hash slots.
- Running time depends on length of chains.
- Average length of chain $= \alpha = m/n$.
- Choose $n = m \Rightarrow$ expect $O(1)$ per insert, lookup, or delete.

**Challenge.** Hash function $h$ that achieves $O(1)$ per operation.
**Approach.** Use randomization for the choice of $h$.

adversary knows the randomized algorithm you’re using, but doesn’t know random choice that the algorithm makes

- Universal hashing: analysis

**Proposition.** Let $H$ be a universal family of hash functions mapping a universe $U$ to the set $\{0, 1, \ldots, n - 1\}$; let $h \in H$ be chosen uniformly at random from $H$; let $S \subseteq U$ be a subset of size at most $n$; and let $u \notin S$.
Then, the expected number of items in $S$ that collide with $u$ is at most 1.

**Pf.** For any $s \in S$, define 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 $u$.

$$E_{h \in H}[X] = E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr[X_s = 1] \leq \sum_{s \in S} \frac{1}{n} = |S| \frac{1}{n} \leq 1$$

linearity of expectation \hspace{1cm} $X_s$ is a 0–1 random variable \hspace{1cm} universal

**Q.** OK, but how do we design a universal class of hash functions?

**Universal hashing (Carter–Wegman 1980s)**

A universal family of hash functions is a set of hash functions $H$ mapping a universe $U$ to the set $\{0, 1, \ldots, n - 1\}$ such that
- For any pair of elements $u \neq v$: $\Pr_{h \in H}[h(u) = h(v)] \leq 1/n$.
- Can select random $h$ efficiently.
- Can compute $h(u)$ efficiently.

**Ex.** $U = \{a, b, c, d, e, f\}$, $n = 2$.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>h(y)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h(z)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>h(t)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h(u)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>h(v)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$H = \{h_1, h_2\}$

$$\Pr_{h \in H}[h(a) = h(b)] = 1/2$$

$$\Pr_{h \in H}[h(a) = h(c)] = 1$$

$$\Pr_{h \in H}[h(a) = h(d)] = 0$$

... $H = \{h_1, h_2, h_3, h_4\}$

$$\Pr_{h \in H}[h(a) = h(b)] = 1/2$$

$$\Pr_{h \in H}[h(a) = h(c)] = 1/2$$

$$\Pr_{h \in H}[h(a) = h(d)] = 1/2$$

$$\Pr_{h \in H}[h(a) = h(e)] = 1/2$$

$$\Pr_{h \in H}[h(a) = h(f)] = 0$$

... $H = \{h_1, h_2, h_3, h_4\}$

**Designing a universal family of hash functions**

**Modulus.** We will use a prime number $p$ for the size of the hash table.

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

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

$$h_a(x) = (\sum_{i=1}^r a_i x_i) \mod p \hspace{1cm} \text{maps universe } U \text{ to set } \{0, 1, \ldots, p - 1\}$$

**Hash function family.** $H = \{h_a : a \in A\}$. 
Designing a universal family of hash functions

Theorem. \( H = \{ h_a : a \in A \} \) is a universal family of hash functions.

Pf. Let \( x = (x_1, x_2, \ldots, x_r) \) and \( y = (y_1, y_2, \ldots, y_r) \) be two distinct elements of \( U \). We need to show that \( \Pr[h_x(x) = h_y(y)] \leq 1/p \).

- Since \( x \neq y \), there exists an integer \( j \) such that \( x_j \neq y_j \).
- We have \( h_x(x) = h_y(y) \) iff
  \[
  a_j \left( \frac{y_j - x_j}{z} \right) \equiv \sum_{i \neq j} a_i(x_i - y_i) \mod p
  \]
- Can assume \( a \) was chosen uniformly at random by first selecting all coordinates \( a_i \), where \( i \neq j \), then selecting \( a_j \) at random. Thus, we can assume \( a_j \) is fixed for all coordinates \( i \neq j \).
- Since \( p \) is prime, \( a_jz = m \mod p \) has at most one solution among \( p \) possibilities.
- Thus \( \Pr[h_x(x) = h_y(y)] \leq 1/p. \)

Number theory fact

Fact. Let \( p \) be prime, and let \( z \neq 0 \mod p \). Then \( \alpha z = m \mod p \) has at most one solution \( 0 \leq \alpha < p \).

Pf.

- Suppose \( 0 \leq \alpha_1 < p \) and \( 0 \leq \alpha_2 < p \) are two different solutions.
- \( (\alpha_1 - \alpha_2)z = 0 \mod p \); hence \( (\alpha_1 - \alpha_2)z \) is divisible by \( p \).
- Since \( z \neq 0 \mod p \), we know that \( z \) is not divisible by \( p \).
- It follows that \( (\alpha_1 - \alpha_2) \) is divisible by \( p \).
- This implies \( \alpha_1 \equiv \alpha_2 \). 

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

Pf idea. Euclid’s algorithm.

Universal hashing: summary

Goal. Given a universe \( U \), maintain a subset \( S \subseteq U \) so that insert, delete, and lookup are efficient.

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

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

- Choose \( p \) prime so that \( m \leq p \leq 2m \), where \( m = |S| \).
- Fact: there exists a prime between \( m \) and \( 2m \).

Consequence.

- Space used is \( \Theta(m) \).
- Expected number of collisions per operation is \( \leq 1 \)
  \( \Rightarrow O(1) \) time per insert, delete, or lookup.

13. RANDOMIZED ALGORITHMS

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
**Chernoff Bounds (above mean)**

**Theorem.** Suppose $X_1, ..., X_n$ are independent 0-1 random variables. Let $X = X_1 + ... + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

$$Pr[X > (1 + \delta)\mu] < \left(\frac{e^{\delta}}{(1 + \delta)^{\delta}}\right)^n$$

*The sum of independent 0-1 random variables is tightly centered on the mean.*

**Pf.** We apply a number of simple transformations.

- For any $t > 0$,
  $$Pr[X > (1 + \delta)\mu] = Pr\left[e^{tX} > e^{t(1+\delta)\mu}\right] \leq e^{-t(1+\delta)\mu} E[e^{tX}]$$
  \[\text{Markov's inequality: } Pr[X > a] = E[X]/a\]

- Now
  $$E[e^{tX}] = E[e^{t(\sum_i X_i)}] = \prod_i E[e^{tX_i}]$$
  \[\text{definition of } X \text{ independence}\]

\[\text{previous slide inequality above}\]

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

**Chernoff Bounds (below mean)**

**Theorem.** Suppose $X_1, ..., X_n$ are independent 0-1 random variables. Let $X = X_1 + ... + X_n$. Then for any $\mu \leq E[X]$ and for any $0 < \delta < 1$, we have

$$Pr[X < (1 - \delta)\mu] < e^{-\delta^2\mu/2}$$

**Pf idea.** Similar.

**Remark.** Not quite symmetric since only makes sense to consider $\delta < 1$.

---

**13. RANDOMIZED ALGORITHMS**

- contention resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Load balancing

**System** in which \( m \) jobs arrive in a stream and need to be processed immediately on \( m \) identical processors. Find an assignment that balances the workload across processors.

**Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most \( \left\lfloor m/n \right\rfloor \) jobs.

**Decentralized controller.** Assign jobs to processors uniformly at random. How likely is it that some processor is assigned “too many” jobs?

Load balancing: many jobs

**Theorem.** Suppose the number of jobs \( m = 16n \ln n \). Then on average, each of the \( n \) processors handles \( \mu = 16 \ln n \) 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
  \[
  \Pr[X_i > 2\mu] < \left( \frac{e}{4} \right)^{16n \ln n} < \left( \frac{1}{e} \right)^{\ln n} = \frac{1}{n^2}
  \]
  \[
  \Pr[X_i < \frac{1}{2}\mu] < e^{-\frac{1}{2}(\frac{1}{2})^2 16n \ln n} = \frac{1}{n^2}
  \]
- Union bound \( \Rightarrow \) every processor has load between half and twice the average with probability \( \geq 1 - 2/n \).

**Load balancing**

**Analysis.**
- Let \( X_i = \) number of jobs assigned to processor \( i \).
- Let \( Y_{ij} = 1 \) if job \( j \) assigned to processor \( i \), and 0 otherwise.
- We have \( \mathbb{E}[Y_{ij}] = 1/n \).
- Thus, \( X_i = \sum_j Y_{ij} \), and \( \mu = \mathbb{E}[X_i] = 1 \).
- Applying Chernoff bounds with \( \delta = c - 1 \) yields \( \Pr[X_i > c] < \frac{e^{c-1}}{c^c} \).
- Let \( \gamma(n) \) be number \( x \) such that \( x^n = n \), and choose \( c = e^{\gamma(n)} \).
  \[
  \Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left( \frac{e}{c} \right)^c < \left( \frac{1}{\gamma(n)} \right)^{\gamma(n)} < \left( \frac{1}{\gamma(n)} \right)^{2\gamma(n)} = \frac{1}{n^2}
  \]
- Union bound \( \Rightarrow \) with probability \( \geq 1 - 1/n \) no processor receives more than \( e^{\gamma(n)} = \Theta((\log n) / \log \log n) \) jobs. 

Bonus fact: with high probability, some processor receives \( \Theta(\log n / \log \log n) \) jobs

Load balancing: many jobs

**Theorem.** Suppose the number of jobs \( m = 16n \ln n \). Then on average, each of the \( n \) processors handles \( \mu = 16 \ln n \) 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
  \[
  \Pr[X_i > 2\mu] < \left( \frac{e}{4} \right)^{16n \ln n} < \left( \frac{1}{e} \right)^{\ln n} = \frac{1}{n^2}
  \]
  \[
  \Pr[X_i < \frac{1}{2}\mu] < e^{-\frac{1}{2}(\frac{1}{2})^2 16n \ln n} = \frac{1}{n^2}
  \]
- Union bound \( \Rightarrow \) every processor has load between half and twice the average with probability \( \geq 1 - 2/n \).