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. 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.

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

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

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

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

• $(1 - 1/n)^n$ converges monotonically from $1/4$ up to $1/e$.

• $(1 - 1/n)^n$ 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 $F[i, t]$ = event that at least one of the $n$ processes fails to access database in any of the rounds 1 through $t$.

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

Choosing $t = 2 \lfloor en \rfloor \lceil c \ln n \rceil$ yields $\Pr[F[i, t]] \leq n \cdot n^2 = 1/n$. 

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 $G = (V, E)$, 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 $k = |F^*| = \text{size of min cut}.$
- 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 $\Rightarrow |E| \geq \frac{1}{2} k n.$
- 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 $k = |F^*|$ be 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 $k$, $|F'| \geq \frac{1}{2} k n'$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n'$.
- Let $E_j = \text{event that an edge in } F^* \text{ is not contracted in iteration } j$.

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

**Contraction algorithm: example execution**

<table>
<thead>
<tr>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
<th>Trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial 5 (finds min cut)</th>
<th>Trial 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Reference: Thore Husfeldt

**Global min cut: context**

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

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

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

\[ faster \text{ than best known max flow algorithm or deterministic global min cut algorithm } \]
13. RANDOMIZED ALGORITHMS

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

Expectation: two properties

**Useful property.** If $X$ is a 0/1 random variable, $E[X] = \Pr[X = 1]$.

**Pf.**

\[
E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \Pr[X = 1]
\]

not necessarily independent

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

**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.

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

linearity of expectation

Expectation

**Expectation.** Given a discrete random variables $X$, 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?

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

Wb
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 \( X = \) number of correct guesses = \( X_1 + \ldots + X_n \).
- \( E[X_i] = \text{Pr}[X_i = 1] = 1 / (n - i - 1) \).
- \( E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = H(n) \). ★

linearity of expectation

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

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 \( X_j = \) number of steps you spend in phase \( j \).
- Let \( X = \) number of steps in total = \( X_0 + X_1 + \ldots + X_{n-1} \).

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

\( \Theta(n \log n) \)

\( \text{prob of success} = (n-j)/n \)

\( \text{expected waiting time} = n / (n-j) \)

Maximum 3-satisfiability

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

- \( C_1 = x_2 \lor x_3 \lor x_4 \)
- \( C_2 = x_2 \lor x_3 \lor \overline{x_4} \)
- \( C_3 = \overline{x_1} \lor x_2 \lor x_4 \)
- \( C_4 = \overline{x_1} \lor \overline{x_2} \lor x_3 \)
- \( C_5 = x_1 \lor \overline{x_2} \lor x_4 \)

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 \( \frac{7}{8}k \).

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

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

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

The Probabilistic Method

Corollary. For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \( \frac{7}{8} \) fraction of all clauses.

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

Probabilistic method. [Paul Erdős] 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 \( \frac{7}{8} \)-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.

\[
\frac{7}{8}k = E[Z] = \sum_{j=0}^{7k/8} j p_j + \sum_{j=7k/8}^{k} j p_j 
\]

\[
\leq \left( \frac{7}{8} - \frac{1}{8} \right) \sum_{j=7k/8}^{k} p_j + k \sum_{j=7k/8}^{k} p_j 
\]

\[
\leq \left( \frac{7}{8} - \frac{1}{8} \right) \cdot 1 + kp
\]

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

Maximum 3-satisfiability: analysis

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

Theorem. Johnson’s algorithm is a \( \frac{7}{8} \)-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 3-SAT.

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

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

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 \( \geq 1/2 \).

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

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

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.**
- create(): initialize a dictionary with $S = \emptyset$.
- insert(u): add element $u \in U$ to $S$.
- delete(u): delete $u$ from $S$ (if $u$ is currently in $S$).
- 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 $H$ 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 $\Theta(\sqrt{n})$ random insertions.
- Separate chaining: $H[i]$ 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;
}
```

hash function ala Java string library

**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 search 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 reactors.
- Surprising situations: denial-of-service attacks.

Malicious adversary learns your ad hoc hash function (e.g., by reading Java API) and causes a big pile-up in a single slot that grinds performance to a halt.

**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.
Hashing performance

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

Challenge. Achieve idealized randomized guarantees, but with a hash function where you can easily find items where you put them.

Approach. Use randomization in the choice of \( h \).

\[ \operatorname{Universal hashing: analysis} \]

Proposition. Let \( H \) be a universal family 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 \( u \).

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

\( \Rightarrow \) linearity of expectation, \( X_s \) is a 0-1 random variable, universal (assumes \( u \notin S \)).

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

Universal hashing

Universal family of hash functions. [Carter-Wegman 1980s]
- For any pair of elements \( u, v \in U \), \( \Pr_{h \in H} [h(u) = h(v)] \leq 1/n \).
- Can select random \( h \) efficiently.
- Can compute \( h(u) \) efficiently.

\[ \operatorname{Ex.} U = \{ a, b, c, d, e, f \}, n = 2. \]

\[ H = \{ h_1, h_2 \} \]

\[
\begin{align*}
\Pr_{h \in H} [h(a) = h(b)] &= 1/2 \\
\Pr_{h \in H} [h(a) = h(c)] &= 0 \\
\Pr_{h \in H} [h(a) = h(d)] &= 0 \\
\vdots \\
\end{align*}
\]

\[ H' = \{ h_1, h_2, h_3, h_4 \} \]

\[
\begin{align*}
\Pr_{h \in H} [h(a) = h(b)] &= 1/2 \\
\Pr_{h \in H} [h(a) = h(c)] &= 1/2 \\
\Pr_{h \in H} [h(a) = h(d)] &= 1/2 \\
\Pr_{h \in H} [h(a) = h(e)] &= 1/2 \\
\Pr_{h \in H} [h(a) = h(f)] &= 0 \\
\vdots \\
\end{align*}
\]

Designing a universal family of hash functions

Theorem. [Chebyshev 1850] There exists a prime between \( n \) and \( 2n \).

Modulus. Choose a prime number \( p \approx n \). \( \Leftrightarrow \) no need for randomness here.

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 \( A \) 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

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

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 $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ be two distinct elements of $U$. We need to show that $Pr[h_a(x) = h_a(y)] \leq 1/n$.

- Since $x \neq y$, there exists an integer $j$ such that $x_j \neq y_j$.
- We have $h_a(x) = h_a(y)$ iff
  
  $$a_j (y_j - x_j) \equiv \sum_{i \neq j} a_i (x_i - y_i) \mod p$$

- 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, $a_j \equiv m \mod p$ has at most one solution among $p$ possibilities. \(\Rightarrow\) see lemma on next slide
- Thus $Pr[h_a(x) = h_a(y)] = 1/p \leq 1/n$.

### 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 $\alpha$ and $\beta$ are two different solutions.
- Then $(\alpha - \beta) z = 0 \mod p$; hence $(\alpha - \beta) z$ is divisible by $p$.
- Since $z \neq 0 \mod p$, we know that $z$ is not divisible by $p$; it follows that $(\alpha - \beta)$ 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)

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

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

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

- For any $t > 0$,
  $$Pr[X > (1+\delta)\mu] = Pr[ e^{tX} > e^{(1+\delta)\mu} t ] \leq e^{-t(1+\delta)\mu} \cdot E[e^{tX}]$$

- Now $E[e^{tX}] = E[e^{t \sum X_i}] = \prod_i E[e^{tX_i}]$
Chernoff Bounds (above mean)

\[ \text{Pr} \left[ \sum_i X_i > \left(1 + \delta \right) \mu \right] \leq e^{-\frac{\delta^2}{2}\mu} \]

For any \( \alpha > 0 \), \( \sum_i e^{\alpha X_i} \leq e^{\alpha \mu(1+\alpha)} \)

\[ \text{Pr} \left[ e^{\alpha X} \right] = e^{\alpha \mu} \text{ for any } \alpha > 0, \text{ s.t. } \sum_i e^{\alpha X_i} = e^{\mu(1+\alpha)} \]

\[ \sum_i e^{\alpha X_i} \leq e^{\mu(1+\alpha)} \]

\[ \text{for any } \alpha > 0, \text{ s.t. } \sum_i e^{\alpha X_i} = e^{\mu(1+\alpha)} \]

Combining everything:

\[ \text{Pr} \left[ e^{\alpha X} \right] = e^{\mu(1+\alpha)} \text{ for any } \alpha > 0, \text{ s.t. } \sum_i e^{\alpha X_i} = e^{\mu(1+\alpha)} \]

Finally, choose \( \tau = \ln(1 + \delta) \).

Chernoff Bounds (below mean)

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

\[ \text{Pr} \left[ X < (1-\delta) \mu \right] < e^{-\frac{\delta^2 \mu}{2}} \]

**Pf idea.** Similar.

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

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 \( \lfloor m/n \rfloor \) jobs.

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

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 $E[Y_{ij}] = 1/n$.
- Thus, $X_i = \sum Y_{ij}$, and $\mu = E[X_i] = 1$.
- 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 $e^x = n$, and choose $c = e^{\gamma(n)}$.
  $$Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left(\frac{e}{c}\right)^c < \left(\frac{1}{\gamma(n)}\right)^{\gamma(n)} = \frac{1}{n^2}$$
  - Union bound $\Rightarrow$ with probability $\geq 1 - 1/n$ no processor receives more than $e^{\gamma(n)} = \Theta(\log n / \log \log n)$ jobs.

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

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