南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

09 Randomized Algorithms

CS216 Algorithm Design and Analysis (H)

Instructor: Shan Chen

chens3@sustech.edu.cn

Randomization

- **Algorithm design patterns:**
  - Greedy
  - Divide and conquer
  - Dynamic programming
  - Duality (e.g., network flow)
  - Reductions
  - Randomization

in practice, access to a pseudorandom number generator

- **Randomization.** Allow fair coin flip in unit time.
- **Why randomize?** Can lead to simplest, fastest, or only known algorithm for a particular problem.
  - E.g., symmetry-breaking protocols, graph algorithms, quicksort, hashing, load balancing, closest pair, Monte Carlo integration, cryptography, etc.

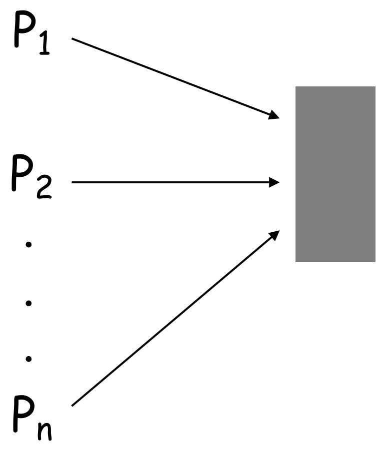
1. Content Resolution

Contention Resolution in 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

- **Randomized protocol.** Each process requests access to the database at any round t with probability $p = 1/n$.

- **Lemma 1.** Let $S[i, t]$ = event that process i succeeds in accessing the database at round t. Then $1/(2n) \geq \Pr[S(i, t)] \geq 1/(e \cdot n)$.

- **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 $1/e$.

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

  process i requests access, none of remaining processes requests access

  Setting $p = 1/n$, we have $\Pr[S(i, t)] = 1/n \, \underbrace{(1 - 1/n)^{n-1}}_{\text{between } 1/e \text{ and } 1/2}$. ▪

  value that maximizes Pr[S(i,t)]

Contention Resolution: Randomized Protocol

- **Randomized protocol.** Each process requests access to the database at any round t with probability $p = 1/n$.

- **Lemma 2.** The probability that process i fails to access the database in $e \cdot n$ rounds is at most $1/e$. After $e \cdot n\ (c \ln n)$ rounds, the probability $\leq n^{-c}$.
- **Pf.** Let $F[i, t]$ = event that process i fails to access database between rounds $1 \sim t$. By independence and **Lemma 1**, $\Pr[F(i, t)] \leq (1 - 1/(en))^t$.

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

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

Contention Resolution: Randomized Protocol

- **Theorem.** The probability that all processes succeed within $2en \ln n$ rounds is $\geq 1 - 1/n$.
- **Pf.** Let $F[t]$ = event that at least one of the n processes fails to access database in any rounds $1 \sim t$.

$$\Pr\big[F[t]\big] = \Pr\left[\bigcup_{i=1}^{n} F[i,t]\right] \leq \sum_{i=1}^{n} \Pr[F[i,t]]$$

union bound

Lemma 2

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

Union bound. Given events $E_1, \ldots, E_n$, $\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i]$

2. Global Min 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, circuit design, TSP solvers, etc.

- **Network flow solution:**
  - Replace every edge (u, v) with two antiparallel edges (u, v) and (v, u).
  - Pick any vertex $s \in V$: for every other node $v \in V$, compute min s-v cut.

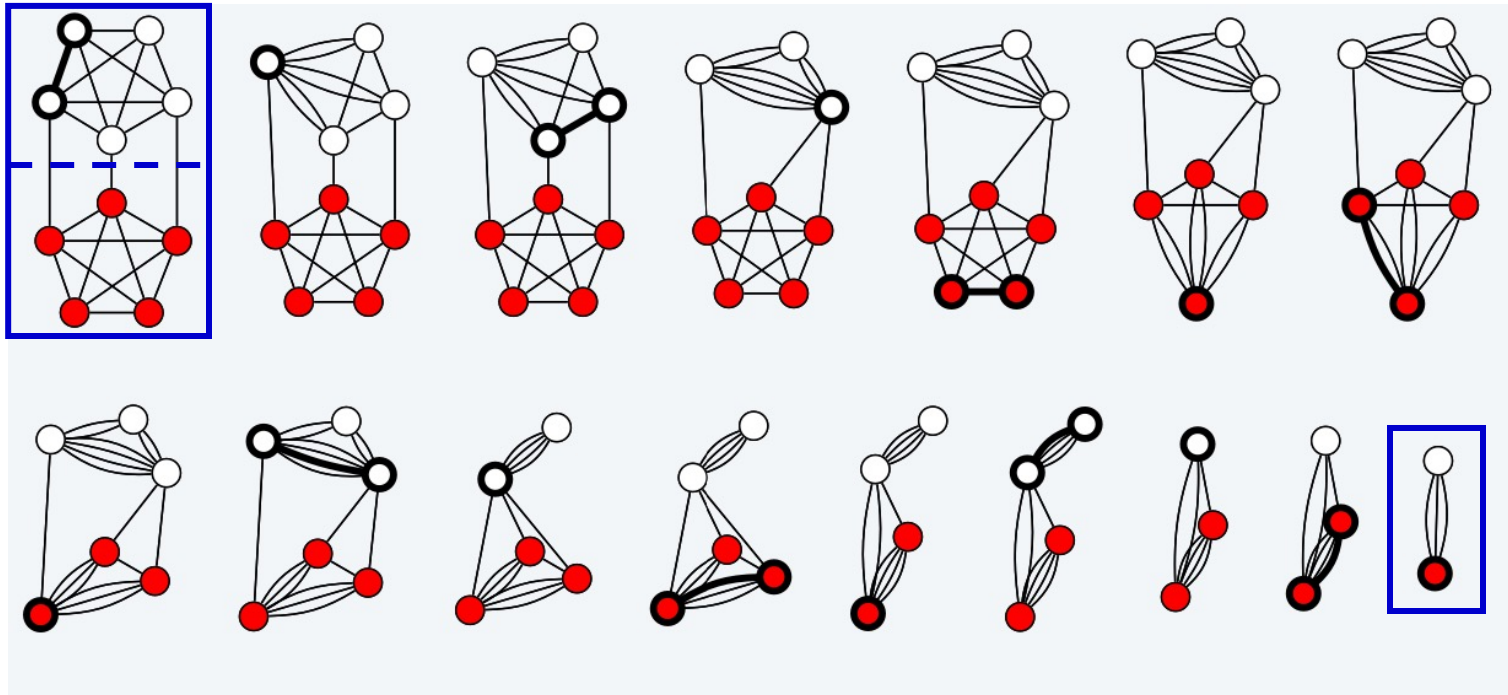
- **False intuition.** Global min-cut is harder than min s-t cut.

Global Min 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 supernode w
    - ✓preserve edges, updating endpoints of u and v to w
    - ✓keep parallel edges, but delete self-loops

Global Min Cut: Contraction Algorithm

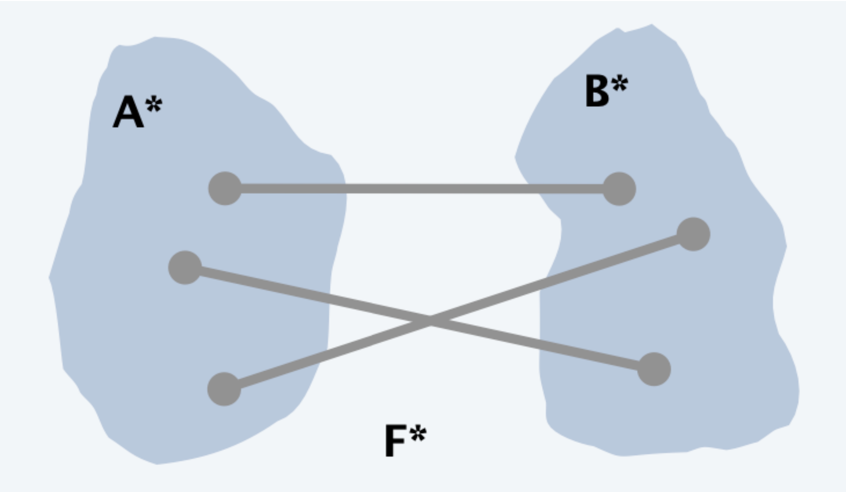
- **Contraction algorithm:** [Karger 1995]
  - Pick an edge $e = (u, v)$ uniformly at random. Contract edge e.
  - Repeat until graph has just two supernodes v_1 and v_2.
  - Return the cut $(S(v_1), S(v_2))$ (where $S(v_i)$ denote all nodes contracted to v_i).



Reference: Thore Husfeldt

Contraction Algorithm: Analysis

- **Theorem.** 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 in this min cut and let $k = |F^*|$ = 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. Therefore, we have $2|E| \geq kn \Leftrightarrow k / |E| \leq 2/n$.
  - Thus, the algorithm contracts an edge in F^* with probability $\leq 2/n$.



Contraction Algorithm: Analysis

- **Theorem.** 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 in this min cut and let $k = |F^*|$ = size of min cut.
  - ➢ Let $G' = (V', E')$ be graph after j iterations, then G' has $n' = n - j$ (super)nodes.
  - ➢ If no edge in F^* has been contracted, the min-cut in G' is still k. Then, as before, $k / |E'| \leq 2/n'$. Thus, algorithm contracts an edge in F^* with probability $\leq 2/n'$.
  - ➢ Let E_j = event that no edge in F^* is contracted in iteration j.

$$
\begin{aligned}
\Pr[E_1 \cap E_2 \cdots \cap E_{n-2}] &= \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \cdots \times \Pr[E_{n-2} \mid E_1 \cap E_2 \cdots \cap E_{n-3}] \\
&\geq \left(1 - \tfrac{2}{n}\right)\left(1 - \tfrac{2}{n-1}\right) \cdots \left(1 - \tfrac{2}{4}\right)\left(1 - \tfrac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right) \cdots \left(\tfrac{2}{4}\right)\left(\tfrac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&\geq \frac{2}{n^2}
\end{aligned}
$$

Contraction Algorithm: Amplification

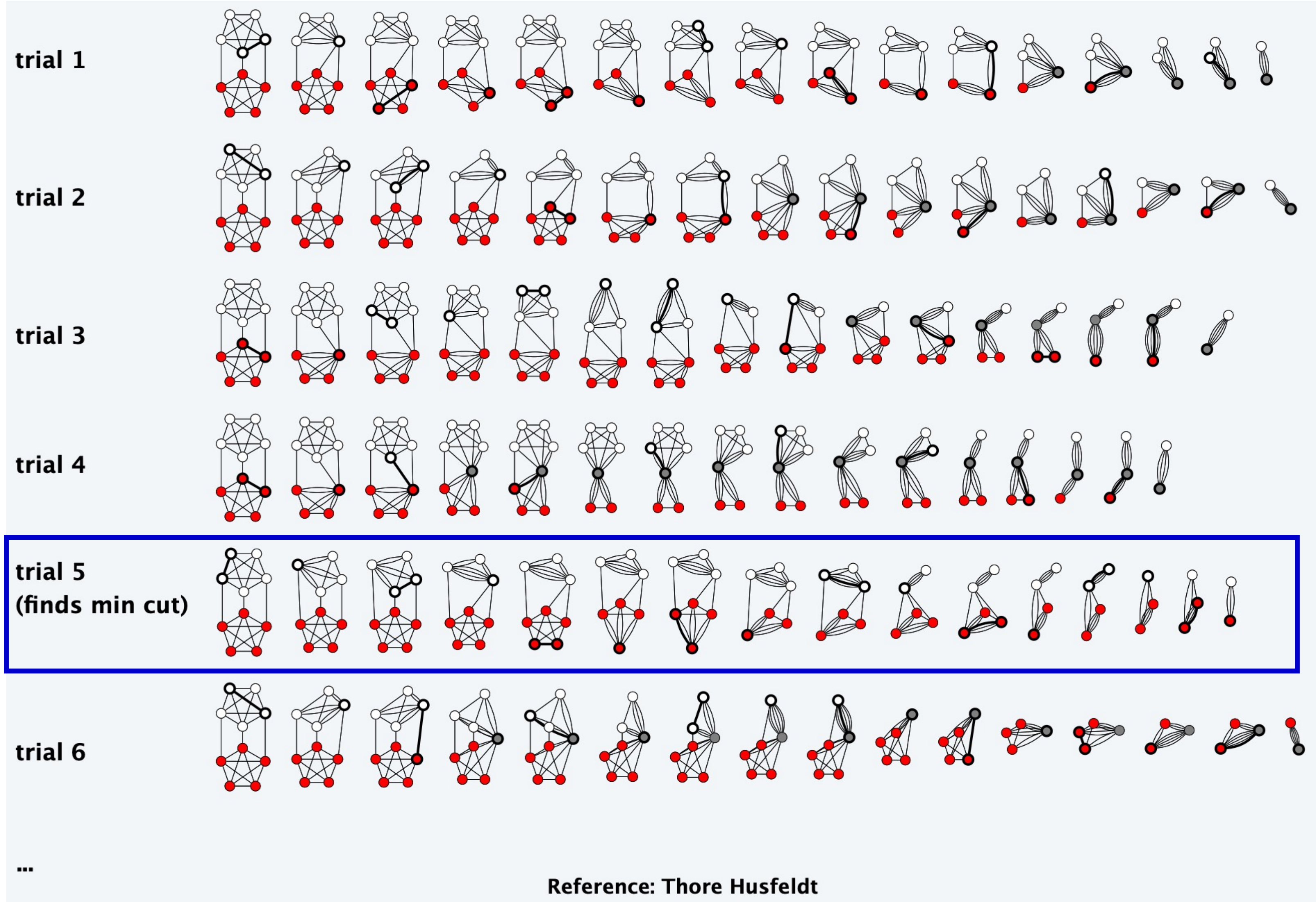
- **Amplification.** To amplify the probability of success, run the contraction algorithm many times with independent randomness.

- **Claim.** If we repeat the contraction algorithm $n^2 \ln n$ times, then the probability of failing to find the global min cut is $\leq 1/n^2$.
- **Pf.** By independence, the probability of failure is at most

$$\left(1-\frac{2}{n^2}\right)^{n^2 \ln n} = \left[\left(1-\frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right]^{2\ln n} \underset{\uparrow}{\leq} \left(e^{-1}\right)^{2\ln n} = \frac{1}{n^2}$$

$(1 - 1/x)^x \leq 1/e$

Contraction Algorithm: Demo



More on Global Minimum Cut

- **Remark.** Overall running time $\Theta(n^2 m \log n)$ 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.
- **Extensions.** Naturally generalizes to handle positive weights.
- **Best known.** [Karger 2000] $O(m \log^3 n)$. ← *faster than best known max flow algorithm or deterministic global min cut algorithm*

3. Load Balancing

Load Balancing

- **Load balancing.** System in which m jobs arrive in a stream and need to be processed immediately on n identical processors. Find an assignment that balances the workload across processors.

- **Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most $\lceil m/n \rceil$ jobs.

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

Chernoff Bounds

- **Setting:**
  - $X_1, \dots, X_n$: independent random variables on $\{0, 1\}$
  - $X = X_1 + \dots + X_n$
  - $\mathbf{E}(X) = \mathbf{E}(X_1) + \dots + \mathbf{E}(X_n)$
- **Theorem.** (above mean) For any $\delta > 0$ and $\mu \geq \mathbf{E}(X)$, we have

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

typically choose μ = E(X)

- **Theorem.** (below mean) For any $\delta > 0$ and $\mu \leq \mathbf{E}(X)$, , we have

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

- **Takeaway.** Provide exponentially decreasing bounds on the probabilities of large deviations from the expected value.

Load Balancing: # Jobs = # Processors

- **Analysis:** (number of jobs m = number of processors n)
  - Let X_i = number of jobs assigned to processor i.
  - Let $Y_{ij} = 1$ if job j is assigned to processor i, and $Y_{ij} = 0$ otherwise.
  - Thus, $X_i = \sum_j Y_{ij}$. We have $\mathbf{E}[Y_{ij}] = 1/n$ and $\mathbf{E}[X_i] = 1$.
  - Chernoff bounds with $\mu = \mathbf{E}[X_i] = 1$ and $\delta = c - 1 > 0 \Rightarrow \Pr[X_i > c] < e^{c-1} / c^c$.
  - Let $\gamma(n)$ be number x such that $x^x = n$, and choose $c = e\,\gamma(n)$.

$$\Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left(\frac{e}{c}\right)^c = \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)} \le \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)} = \frac{1}{n^2}$$

  - Union bound $\Rightarrow$ with probability $\le n \cdot 1/n^2 = 1/n$ exists some processor receives more than c jobs $\Rightarrow$ with probability $\ge 1 - 1/n$ no processor receives more than $c = e\,\gamma(n) = \Theta(\log n / \log\log n)$ jobs.

    ✓Do $\ln$ and $\ln\ln$ on both sides of $\gamma(n)^{\gamma(n)} = n \Rightarrow \gamma(n)/2 \le \log n / \log\log n \le \gamma(n)$.

Load Balancing: # Jobs > # Processors

- **Theorem.** Suppose the number of jobs $m = 16\ n \ln n$. Then on average, each of the n processors handles $16 \ln n$ jobs. With high probability, every processor will have between half and twice the average load.
- **Pf.** (number of jobs m > number of processors n)
  - Let X_i = number of jobs assigned to processor i.
  - Applying Chernoff bounds with $\delta = 1$ and $\mu = \mathbf{E}(X_i) = 16 \ln n$ yields

$$\Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16 \ln n} < \left(\frac{1}{e}\right)^{2 \ln n} = \frac{1}{n^2}$$

$$\Pr\left[X_i < \frac{1}{2}\mu\right] < e^{-\frac{1}{2}\left(\frac{1}{2}\right)^2 16 \ln n} = \frac{1}{n^2}$$

  - Union bound ⇒ every processor has load between half and twice the average with probability $\geq 1 - 2/n$. ▪

4. MAX 3-SAT

Maximum 3-Satisfiability

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

$$\begin{aligned}
C_1 &= x_2 \vee \overline{x_3} \vee \overline{x_4} \\
C_2 &= x_2 \vee x_3 \vee \overline{x_4} \\
C_3 &= \overline{x_1} \vee x_2 \vee x_4 \\
C_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\
C_5 &= x_1 \vee \overline{x_2} \vee \overline{x_4}
\end{aligned}$$

- **Remark.** **NP**-hard optimization problem.
- **Simple idea.** Flip a coin, and set each variable true with probability ½, independently for each variable.

Maximum 3-Satisfiability: Analysis

- **Theorem.** Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$.

- **Pf.** Consider random variables $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$

  Let $Z = \sum_j Z_j$ be number of clauses satisfied by random assignment.

$$
\begin{aligned}
E[Z] &= \sum_{j=1}^{k} E[Z_j] \\
&= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \\
&= \tfrac{7}{8} k
\end{aligned}
$$

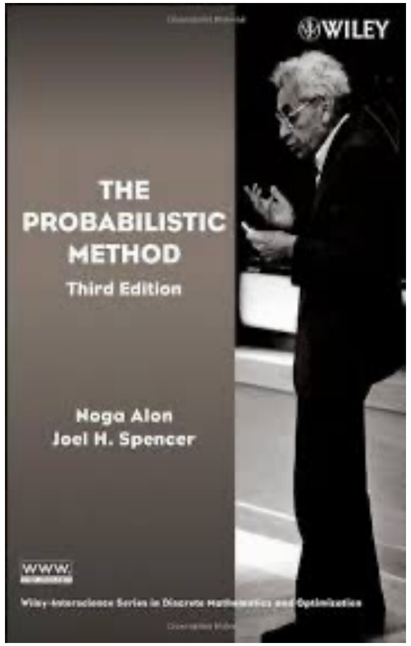
linearity of expectation

disjunction of 3 literals

each literal corresponds to a different 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.** [Paul Erdös] Prove the existence of a non-obvious property by showing that a random construction produces it with positive probability!



Maximum 3-Satisfiability: Further 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.

$$\tfrac{7}{8}k = E[Z] = \sum_{j \geq 0} j\,p_j = \sum_{j < 7k/8} j\,p_j + \sum_{j \geq 7k/8} j\,p_j$$

$$\leq \left(\tfrac{7k}{8} - \tfrac{1}{8}\right) \sum_{j < 7k/8} p_j + k \sum_{j \geq 7k/8} p_j \leq \left(\tfrac{7}{8}k - \tfrac{1}{8}\right) \cdot 1 + k\,p$$

j is integer

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

Maximum 3-Satisfiability: Analysis

- **Johnson's algorithm.** Repeatedly generate random truth assignments until one of them satisfies ≥ $7k/8$ clauses.

- **Theorem.** Johnson's algorithm is a $7k/8$-approximation algorithm.
- **Pf.** (direct proof)
  - **Lemma** ⇒ each iteration succeeds with probability $p \geq 1/(8k)$
  - The expected number of trials to find the satisfying assignment is $\sum_{j=1}^{\infty} j \Pr[j \text{ trials}] = \sum_{j=1}^{\infty} j(1-p)^{j-1}p = \frac{1}{(1-(1-p))^2}p = \frac{1}{p} \leq 8k$ ▪

calculus fact

- **Takeaway.** **NP**-hard problems may have good approximation algorithms.

Maximum Satisfiability

- **Extensions:**
  - MAX-SAT: Allow one, two, or more literals per clause.
  - Weighted MAX-SAT: 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 deterministic *7/8*-approximation algorithm for version of MAX 3-SAT in which each clause has ≤ 3 literals.
- **Theorem.** [Håstad 1997] Unless **P** = **NP**, no ρ-approximation algorithm for MAX 3-SAT (and hence MAX SAT) for any $\rho > 7/8$.

very unlikely to improve over
simple randomized algorithm for MAX 3-SAT

Randomized Algorithms: Closing Remarks

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

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

stop algorithm after a certain point

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

e.g., don't know when to stop

Randomized Algorithms: Closing Remarks

- **RP.** (Randomized Poly-Time) [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 *≥ 1/2*.

can decrease probability of false negative to 2^{-100} by 100 independent repetitions

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

running time can be unbounded, but fast on average

- **Theorem.** **P ⊆ ZPP ⊆ RP ⊆ NP**

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