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.

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

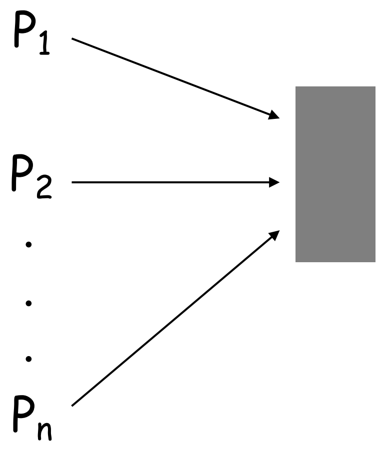
1. Contention Resolution

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

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

Contention Resolution: Randomized Protocol

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

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

union bound

Lemma 2 for c = 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. Median and Selection

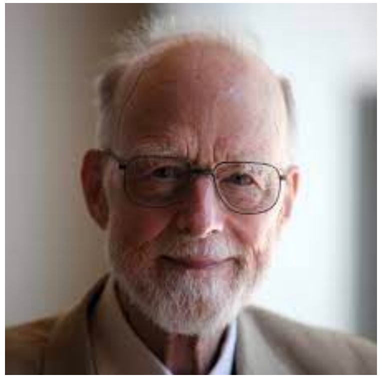
Median and Selection

- **Median and Selection.** Given n elements from a totally ordered universe, find the median element or in general the k-th smallest element.
  - minimum or maximum ($k = 1$ or $k = n$): $O(n)$ compares
  - median: $k = \lfloor (n + 1) / 2 \rfloor$
    - ✓ $O(n \log n)$ compares by sorting
    - ✓ $O(n \log k)$ compares with a binary heap
- **Applications.** Order statistics, find the "top k", bottleneck paths, etc.

- **Q.** Can we do it with $O(n)$ compares?
- **A.** Yes! Selection is easier than sorting.

Recall: Randomized Quicksort

- **Randomized Quicksort:**
  - Pick a random pivot element p.
  - 3-way partition the array into L, M, and R.
    - ✓ L: elements $< p$, M: elements $= p$, R: elements $> p$.
  - Recursively sort both L and R.



Tony Hoare (1959)

A	L	G	O	R	I	T	H	M	S	pick random pivot	$O(1)$
A	L	G	I	H	M	O	R	T	S	3-way partition	$O(n)$
A	G	H	I	L	M	O	R	S	T	sort	$T(\|L\|) + T(\|R\|)$
A	G	H	I	L	M	O	R	S	T	total	$O(n \log n)$ on average

Median and Selection: Divide and Conquer

- **Divide and Conquer:**
  - Pick a random pivot element *p*.
  - *3*-way partition the array into *L, M,* and *R*.
    - ✓*L*: elements < *p*, *M*: elements = *p*, *R*: elements > *p*.
  - Recursively select in one subarray: the one containing the *k*-th smallest element.

A	L	G	O	R	I	T	H	M	S

pick random pivot $O(1)$

A	L	G	I	H	M		O		R	T	S

3-way partition $O(n)$

select $T(|L|)$ or $O(1)$ or $T(|R|)$

Randomized Quickselect

- **Randomized Quickselect.** Divide and Select.

```
Quick-Select(A, k) { // 1 ≤ k ≤ |A|
   Pick pivot p uniformly at random from A
   Partition the list into two three parts L, M and R

   if (k ≤ |L|)
      return Quick-Select(L, k)
   else if (k > |L| + |M|)
      return Quick-Select(R, k - |L| - |M|)
   else
      return p
}
```

- **Q.** What is the expected time complexity of randomized quickselect?
  - ➢ Time complexity is measured by the number of compares.

Randomized Quickselect: Time Complexity

一根木头切一刀，大的部分平均是3/4

- **Intuition.** Split a length-n array uniformly ⇒ expected larger size ~$3n/4$.
  - $T(n) \le T(3n/4) + n \Rightarrow T(n) \le 4n$

    not rigorous: cannot assume E[T(i)] ≤ T(E[i])

- **Def.** Let $T(n, k)$ be the expected number of compares to select the k-th smallest element in an array of length n. Let $T(n) = \max_k T(n, k)$.

- **Claim.** $T(n) \le 4n$

- **Pf.** (by strong induction on n)

T(i) ≤ T(n - i) since T(n) is monotonely non-decreasing

比n/2小的都当成另一半更大的

$$
\begin{aligned}
T(n) &\le n + 1/n\,[\,2T(n/2) + \ldots + 2T(n-3) + 2T(n-2) + 2T(n-1)\,] \\
&\le n + 1/n\,[\,8(n/2) + \ldots + 8(n-3) + 8(n-2) + 8(n-1)\,] \\
&\le n + 1/n\,(3n^2) \\
&= 4n
\end{aligned}
$$

inductive hypothesis

Median and Selection: Closing Remarks

- We learned that randomized Quickselect runs in $O(n)$ time on average.

- [Blum-Floyd-Pratt-Rivest-Tarjan 1973] There exists a compare-based deterministic selection algorithm whose worst-case running time is $O(n)$.
  - This algorithm is also known as median-of-medians selection.
  - Optimized version requires $\leq 5.4305n$ compares.

- **Remark.** In practice, we use randomized selection algorithms since deterministic algorithms have too large constants.
  - However, deterministic algorithms can be used as a fallback for pivot selection.

对位置的选择中

3. Global Min Cut

Global Minimum Cut

没有源点和汇点

- **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.

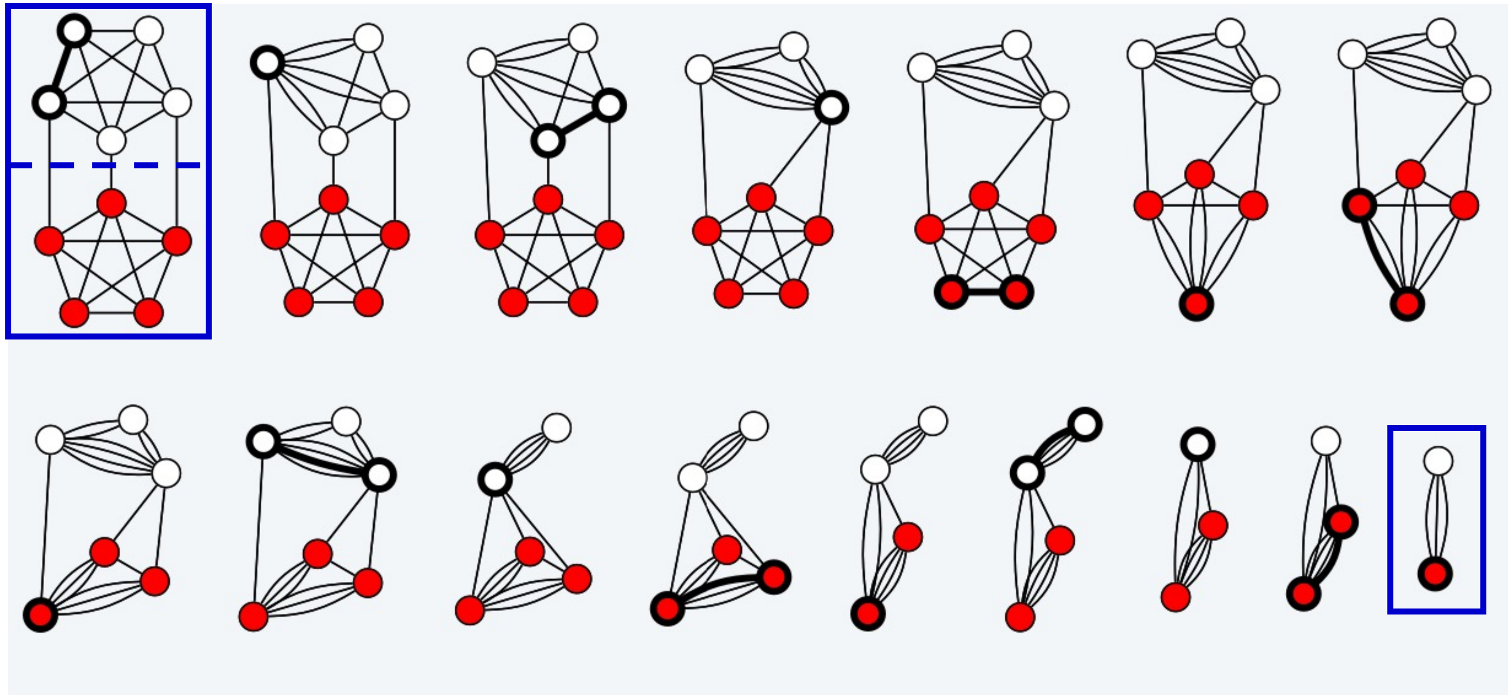
随机算法反而使得这个更简单

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).

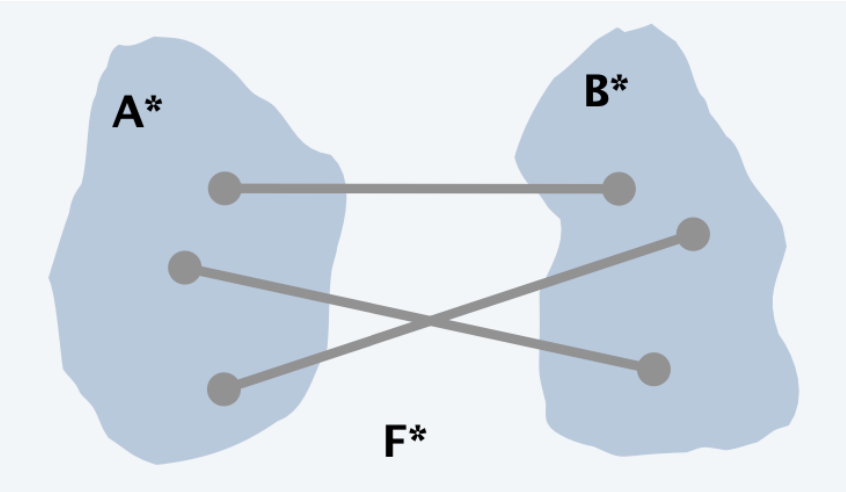


Reference: Thore Husfeldt

每个点对应一个点集

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.

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

Contraction Algorithm: Amplification

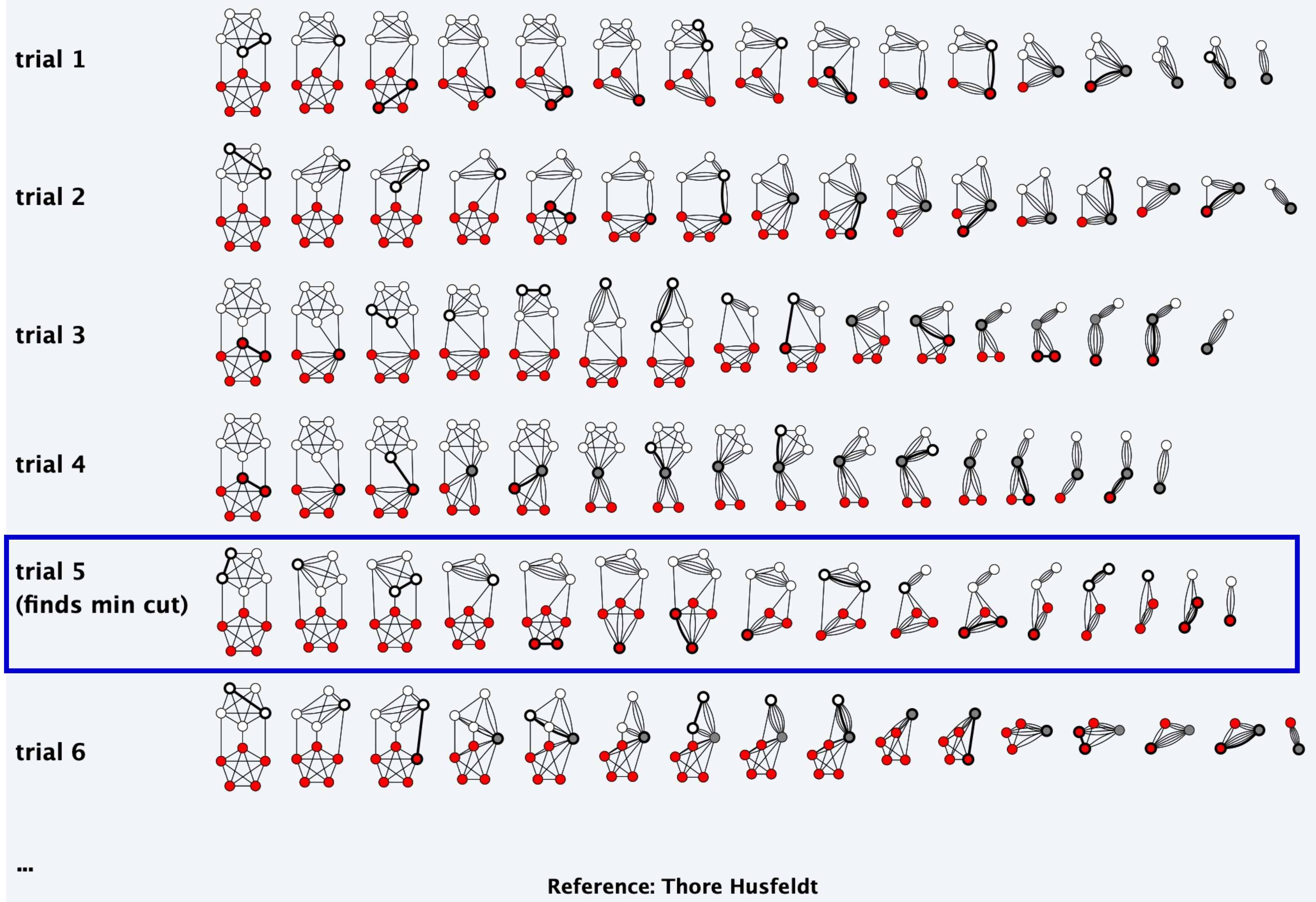
- **Amplification.** To amplify the probability of success, run the contraction algorithm many times with independent randomness.

- **Claim.** If we repeat the contraction algorithm $n^2 \ln n$ times, then the probability of failing to find the global min cut is $\leq 1/n^2$. 很大的概率能找到最优解
- **Pf.** By independence, the probability of failure is at most

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

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

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: (cumulative) 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 and deterministic global min cut algorithm*

Announcement

- **Lab 13 will be released today and the deadline is Jun 3.**

4. 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:**
  - $X_1, \ldots, X_n$: independent random variables on $\{0, 1\}$
  - $X = X_1 + \ldots + X_n$
  - $\mathbf{E}(X) = \mathbf{E}(X_1) + \ldots + \mathbf{E}(X_n)$
- **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.** Chernoff bounds 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 $c = e\,\gamma(n)$.

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

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

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

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 $\geq 1 - 2/n$. ▪

5. MAX 3-SAT

Maximum 3-Satisfiability

随机算法，尽量满足更多的3SAT

- **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.

随机赋值

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

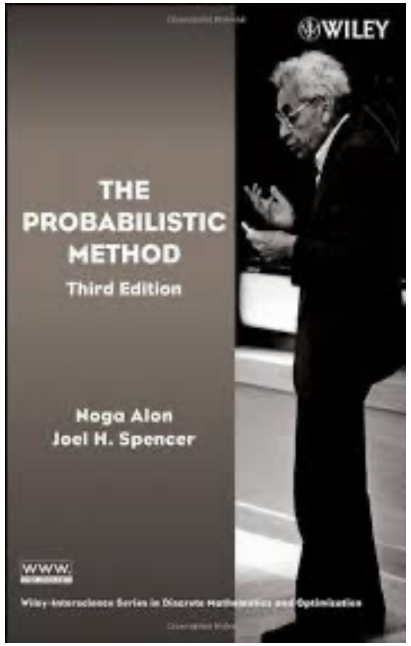
disjunction of 3 literals
each literal corresponds to a different variable

The Probabilistic Method

存在一个赋值，一定使得至少7/8能满足

- **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: Further 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.

平均意义上是多项式时间

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

总是能超过7k/8

stop algorithm after a certain point

限定时间就可以

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

无法知道什么时候是最优值

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.

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

- **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.

running time can be unbounded, but fast on average

- **Fact.** **P ⊆ ZPP ⊆ RP ⊆ NP**

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