Randomized Algorithms

CMSC 451, Summer 2009

Randomization

Algorithmic design patterns.
- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

Randomization. Allow fair coin flip in unit time.

Randomize input: Useful for testing and analysis of a deterministic algorithm.

Randomize choices in the algorithm: The algorithm itself relies on randomization to operate correctly.
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, Monte Carlo integration, cryptography.
Probability Basics

**Pr[A]:** The probability that an event A will occur.

**Pr[A∩B]:** The probability that both event A and event B will occur.

If \( \text{Pr}[A] = x \) and \( \text{Pr}[B] = y \) and A and B are independent, then \( \text{Pr}[A \cap B] = xy \).

**Random variable:** An unknown value that can change every time it is inspected.

**Discrete random variable:** A random variable that has a limited number of possible values (at most a countably infinite number).

Expectation

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

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

**Example:** The expected value of a roll of a six-sided die is:

\[
E[\text{Roll of 6-sided die}] = 1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 = 3.5
\]

**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?

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

\( j \) - 1 tails, 1 head
13.2 Global Minimum 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, network design, circuit design, TSP solvers.

Different from network flow problem since

- on an undirected graph
- no source and sink are specified
- all capacities are 1

Reminder: How do we find the min cut in a flow network?
Recall: Augmenting Path Algorithm

**Augment**($f, c, P$) {
    $b \leftarrow$ bottleneck($P$)
    **foreach** $e \in P$ {
        **if** ($e \in E$) $f(e) \leftarrow f(e) + b$
        **else** $f(e^R) \leftarrow f(e^R) - b$
    }
    return $f$
}

**Ford-Fulkerson**($G, s, t, c$) {
    **foreach** $e \in E$ $f(e) \leftarrow 0$
    $G_f \leftarrow$ residual graph
    **while** (there exists augmenting path $P$) {
        $f \leftarrow$ Augment($f, c, P$)
        update $G_f$
    }
    return $f$
}

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 \( v_1 \)).

**Practice: 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 \( v_1 \)).
**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 $k = |F^*| = \text{size of min cut}$.  

- 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 min-cut.  
  $\Rightarrow |E| \geq 1/2 kn$.  
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$. 

---

**Contraction Algorithm**

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

**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 $k$, $|E'| \geq 1/2 kn'$.  
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n'$. 

- Let $E_j$ be event that an edge in $F^*$ is not contracted in iteration $j$.  

\[
\Pr[E_1 \cap E_2 \cap \ldots \cap E_{n-3}] = \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \ldots \times \Pr[E_{n-2} \mid E_1 \cap E_2 \cap \ldots \cap E_{n-3}] \\
\approx \left(1 - \frac{2}{n} \right) \left(1 - \frac{2}{n-1} \right) \ldots \left(1 - \frac{2}{n-3} \right) \\
= \left(\frac{n-2}{n} \right) \left(\frac{n-3}{n-1} \right) \ldots \left(\frac{3}{5} \right) \left(\frac{2}{3} \right) \\
= \frac{2}{n^2} \\
\approx \frac{2}{n^2}
\]
**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

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

\[
(1 - 1/x)^x \leq e^{-1}
\]

**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] \( O(n^2 \log^3 n) \).
- 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.
Recall: Closest Pair of Points

**Closest pair.** Given n points in the plane, find a pair with smallest Euclidean distance between them.

**Fundamental geometric primitive.**
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.
  \fast closest pair inspired fast algorithms for these problems

**Brute force.** Check all pairs of points p and q with Θ(n^2) comparisons.

**1-D version.** O(n log n) easy if points are on a line.

**Assumption.** No two points have same x coordinate.

\ to make presentation cleaner
Recall: Closest Pair Algorithm

\begin{verbatim}
Closest-Pair(p_1, ..., p_n) {
    Compute separation line L such that half the points are on one side and half on the other side.
    \delta_1 = Closest-Pair(left half)
    \delta_2 = Closest-Pair(right half)
    \delta = min(\delta_1, \delta_2)
    Delete all points further than \delta from separation line L
    Sort remaining points by y-coordinate.
    Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \delta, update \delta.
    return \delta.
}
\end{verbatim}

O(n log n)
2T(n / 2)
O(n)
O(n log n)
O(n)

Closest Pair of Points Randomized Algorithm

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Consider points in the unit square.
- No loss of generality - scale the points in linear time.

Basic algorithm outline:
- Consider the points in some random order p_1, p_2, ..., p_n
- Maintain the current value of \delta = distance between closest pair
- When considering a new point, look in the vicinity to see if any previously considered points are < \delta away. Update \delta if necessary.

How do we look for points in the vicinity?
Closest Pair of Points Randomized Algorithm

Basic algorithm outline:
• Consider the points in some random order \( p_1, p_2, \ldots, p_n \)
• Maintain the current value of \( \delta = \) distance between closest pair
• When considering a new point, look in the vicinity to see if any previously considered points are \( < \delta \) away. Update \( \delta \) if necessary.

Consider the algorithm in stages:
• The closest pair of points remains constant
• Each stage either verifies that \( \delta \) is the minimum distance or finds a pair of points with separating distance less than \( \delta \). The newly found distance becomes the \( \delta \) value for the next stage.

How many stages are there? It depends on the random ordering.
• Minimum: 1. \( p_1 \) and \( p_2 \) are the closest pair.
• Maximum: \( n-1 \). Adding a new point always decreases \( \delta \).
• Will show: expected running time is within a constant factor of the minimum case!

How do we look for points in the vicinity?

Main points:
If \( p \) is involved in the closest pair, then the other point lies in a close subsquare.

\[ \delta / 2 \]

This defines a 5x5 grid around the point.

If \( p \) and \( q \) are in the same subsquare, \( d(p,q) < \delta \)