Randomized Algorithms
CMSC 451, Summer 2009

Randomization

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

Randomization is used for fair coin flipping 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.

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 events A and B will occur.

If Pr(A) = x and Pr(B) = y and A and B are independent, then Pr(A ∩ 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

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

\[ E[X] = \sum_{j=0}^{\infty} j \cdot P(X = j) \]

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

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

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?

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

Contraction Algorithm

Contraction algorithm. [Karger 1995]
- Pick an edge $e = (u, v)$ uniformly at random.
- Contract edge $e$ by replacing $u$ 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 $k = |F^*| = size of min cut$.
• 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 $\rightarrow |E| \leq |V|$. Then $|E| \leq Kn$.
• Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.

Amplification. To amplify the probability of success, run the
contraction algorithm many times.

Claim. If we repeat the contraction algorithm $n/2$ 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
\[
\left(1 - \frac{2}{n^2}\right)^{n/2} = \left(\frac{n}{n^2}ight)^{n/2} = \frac{1}{n^2}.
\]

Global Min Cut: Context

Remark. Overall running time is slow since we perform $\Theta(n^2 \log n)$
iters and each takes $O(n)$ time.

Improvement. \cite{Karger-Stein} $O(n \log^3 n)$.
• 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 $n / 2$ nodes remain.
• Run contraction algorithm twice on resulting graph, and return best of
two cuts.

Extensions. Naturally generalizes to handle positive weights.

Best known. \cite{Karger} $O(m \log^3 n)$, 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.
• Can solve closest pair recursively fast algorithm for these problems.

Brute force. Check all pairs of points $p$ and $q$ with $O(n^2)$ 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

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

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 \( \delta = \text{distance between closest pair} \)
- 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, \( d(p,q) < \delta \)