Introduction to Randomization

1 Randomized Algorithms

There are two types of randomized algorithms:

1. **Las Vegas Algorithms:** Algorithms which always give the correct answer but with nondeterministic running time. Problems solvable by Las Vegas algorithms are in the complexity class ZPP, which stands for zero-probability polynomial time.
2. **Monte Carlo Algorithms:** Algorithms which always run in polynomial time but with nondeterministic output. In particular, they always get "No" instances correct and get "Yes" instances correct with probability $\geq p$. Problems solvable by Monte Carlo algorithms are in the complexity class RP, which stands for randomized polynomial time.

It is known that $ZPP \subset RP$, but it is open if $ZPP = RP$ or even $P = RP$. It is trivial that $P \subset RP$, because every deterministic algorithm is randomized with zero bits of randomness, and $RP \subset NP$ because a string of random bits forms a certifier for a deterministic algorithm.

There are two key paradigms in randomized algorithm design:

1. Circumventing pathological cases, *i.e.,* inputs specifically constructed to be bad.
2. Using random samples as a representative of a larger space.

2 Combinatorial Probability

2.1 Definitions

Definition 1. A discrete **sample space** Ω is often the set $\{0,1\}^n$ and can be thought of as a supply of coin flips.

Definition 2. An **event** $\mathcal{E} \subseteq \Omega$ is a certain bitstring representing a sequence of coin flips.

Definition 3. A **probability measure** is a function $\Pr : \Omega \to [0,1]$ with $\Pr[\mathcal{E}] = \sum_{\omega \in \mathcal{E}} \Pr[\omega]$ and $\Pr[\Omega] = 1$.

Definition 4. A **random variable** is a function $X : \Omega \to \mathbb{Z}$; it assigns an integer label to each possible sequence of coin flips.

Definition 5. The **expectation** of a random variable is a function $\mathbb{E}[X] = \sum_{j \in \mathbb{Z}} j \cdot \Pr[X = j]$. Note that $X = j$ is an event $\mathcal{E}$.

Definition 6. Consider events $\mathcal{E}_1, \ldots, \mathcal{E}_n$. They are **pairwise independent** if

$$\Pr[\mathcal{E}_i \cap \mathcal{E}_j] = \Pr[\mathcal{E}_i] \Pr[\mathcal{E}_j] \ \forall i, j \in [n].$$

They are **mutually independent** if

$$\Pr\left[\bigcap_{i \in S} \mathcal{E}_i\right] = \prod_{i \in S} \Pr[\mathcal{E}_i] \ \forall S \neq \emptyset.$$

Notice that pairwise independence does not imply mutual independence. To see this, consider three independent coin flips X_1, X_2, X_3. Let $\mathcal{E}_{12} = [X_1 = X_2]$, $\mathcal{E}_{13} = [X_1 = X_3]$, and $\mathcal{E}_{23} = [X_2 = X_3]$. Clearly the probability of each event is $1/2$. Then, we have

$$\prod_{i,j} \Pr[\mathcal{E}_{ij}] = 1/8,$$

but

$$\Pr\left[\bigcap_{i,j} \mathcal{E}_i\right] = 1/4.$$

Thus the events are not mutually independent; however, it is easy to check that they are pairwise independent.

2.2 Useful Facts

Fact 7. *The most useful inequality in this class is:*

$$1 + x \leq e^x \ \forall x \in \mathbb{R}.$$

In particular, this inequality comes in handy when x is negative and close to 0 – which is often the case when evaluating probability complements.

Fact 8. *Another useful fact allows us to analyze the intersection of events $\mathcal{E}_1, \ldots, \mathcal{E}_n$ which are not mutually independent:*

$$\Pr\left[\bigcap_{i=1}^{n} \mathcal{E}_i\right] = \prod_{i=1}^{n} \Pr\left[\mathcal{E}_i \mid \bigcap_{j=1}^{i-1} \mathcal{E}_j\right].$$

Intuitively, this is because an intersection of n events can be analyzed in sequence, and every event needs to occur. We will see an application of this fact to the Minimum Cut *problem.*

3 Minimum Cut

In the Minimum Cut problem, we are given an unweighted, undirected graph $G = (V, E)$, and we must find $S \subseteq V$, $S \neq \emptyset$ minimizing

$$\left|e(S, \bar{S})\right| = \left|\{u, v\} : u \in S, v \in \bar{S}\right|.$$

A simple polynomial time solution is to run a minimum $s - t$ cut algorithm for every pair (s, t) of nodes, then keep the best cut found. We can improve this by keeping s fixed; however, this is not very efficient as it requires $\mathcal{O}(n)$ invocations of the Ford-Fulkerson algorithm. We will analyze a self-contained algorithm which uses randomization.

Definition 9. An **edge contraction** on $e = (u, v)$ turns u and v into a single super-node which inherits all previous edges. Contractions include parallel edges but not self-loops.

Algorithm 1 Randomized Algorithm for Minimum Cut

```
while |V| > 2 do
    Let e ~ U(V).
    Perform an edge contraction on e.
end while
Return the cut given by the remaining two nodes.
```

This algorithm clearly outputs a cut after $\mathcal{O}(n)$ contractions. It is not always a minimum cut, so this is a Monte Carlo algorithm. So what is the probability of success? We fix a particular minimum cut $(S, \bar{S})$ and analyze the probability of finding it.

Define the event $\mathcal{E}_i$ as the algorithm *not* contracting an edge crossing $(S, \bar{S})$ in iteration i. Using Fact 8, we have

$$\Pr\left[\bigcap_{i=1}^{n-2} \mathcal{E}_i\right] = \prod_{i=1}^{n-2} \Pr\left[\mathcal{E}_i \mid \bigcap_{j=1}^{i-1} \mathcal{E}_j\right].$$

We will now calculate the right hand side of the equation. Let k be the number of edges crossing $(S, \bar{S})$ at the beginning of the algorithm. Due to the conditional probability, we know that in iteration i, all those k edges are still there. Furthermore, no new edges could have appeared because contractions so far could have only happened inside S or $\bar{S}$. Thus, exactly k edges still cross $(S, \bar{S})$ in iteration i.

If some other cut still survives, the number of edges across it has also not changed. The only changes to the minimum cut happen when candidate cuts are removed, which means the minimum cut can never become sparser. Thus, $(S, \bar{S})$ is still a minimum cut at iteration i.

Let n_i denote the number of nodes at iteration i, and m_i likewise for edges. To show that we are unlikely to contract across our cut, we must show that the ratio k/m_i is small – in other words, that m_i is large.

To lowerbound m_i, we can lowerbound the degrees of each node. Since $(S, \bar{S})$ is a minimum cut, k is a lower bound on the degree of any node in the graph. Thus,

$$\begin{aligned} m_i &\geq k n_i \\ &= \frac{k(n+1-i)}{2}. \end{aligned}$$

Thus we have event failure probability

$$\begin{aligned} \Pr\left[\bar{\mathcal{E}}_i \mid \bigcap_{j=1}^{i-1} \mathcal{E}_j\right] &\leq \frac{2k}{k(n+1-i)} \\ &= \frac{2}{n+1-i}. \end{aligned}$$

And success probability

$$\begin{aligned} \Pr\left[\mathcal{E}_i \mid \bigcap_{j=1}^{i-1} \mathcal{E}_j\right] &\geq 1 - \frac{2}{n+1-i} \\ &= \frac{n-1-i}{n+1-i}. \end{aligned}$$

So,\

$$\begin{aligned} \Pr\left[\bigcap_{i=1}^{n-2} \mathcal{E}_i\right] &\geq \prod_{i=1}^{n-2} \frac{n-1-i}{n+1-i} \\ &= \frac{1-2}{n(n-1)} \\ &= \frac{1}{\binom{n}{2}}, \end{aligned}$$

where the product follows from a telescoping argument. Running this algorithm with independent coin flips $\Theta(n^2 \log n)$ times improves success probability to $1 - \frac{1}{poly(n)}$ as desired.