COMPSCI 630: Randomized Algorithms	January 10, 2018
Lecture 1	
Lecturer: Rong Ge	*Scribe: Xiang Wang*

1 Overview

In this first lecture, we will talk about some general ideas of randomized algorithm and why we need randomness in algorithms. After that, we will give a concrete example of randomized algorithm, which is Karger's algorithm [Kar93] for Min Cut problem. We will also see Karger-Stein algorithm [KS96], which achieves significantly better success probability compared to Karger's algorithm.

2 Introduction

2.1 What are Randomized Algorithms?

What are randomized algorithms? Informally, they are algorithms that make random choices during execution. We give a more formal definition of randomized algorithms as follows:

Definition 1. *Randomized algorithm A is an algorithm that takes an input $x \in \{0,1\}^n$ and a random string $r \in \{0,1\}^m$, such that*

$$\forall x, \Pr_{random\ r}[A(x,r)\ outputs\ correctly] \geq \frac{2}{3}$$

Note algorithm A is actually a deterministic algorithm given the realization of random string r. We call A a randomized algorithm because given an input x, the output can be different depending on different values of r. We also require for any input, the probability of success should be at least $2/3$. Here we have two remarks for this definition.

Remark 1. Randomized algorithm should work for every input x. The algorithm is randomized doesn't mean that for some inputs it works but for some other inputs it just doesn't work. Randomized algorithm should work for every input in the sense that it should succeed with probability of more that half for every input.

Remark 2. The number $2/3$ in the definition is pretty arbitrary. It can be any number larger than half.

2.2 Average Case Analysis

After introducing the definition of randomized algorithms, let's look at a very similar concept, average case analysis.

Definition 2. *Given a distribution D on input $x \in \{0,1\}^n$, A is an average case algorithm if*

$$\Pr_{x\sim D}[A(x)\ outputs\ correctly] \geq \frac{2}{3}$$

In the average case analysis, the input is random, which comes from distribution D. Here, the algorithm A is a deterministic algorithm. So, what's the difference between average case algorithms and randomized algorithms? As we emphasized above, randomized algorithms should work for every input. But for average case algorithms, input x comes from distribution D. So we have nothing to say for input outside the scope of distribution D. Even within distribution D, the average case algorithm is allowed to fail on some inputs. Another difference is that in randomized algorithms, the randomness comes from the random string, which we can generate ourselves. Thus we can repeat the algorithm to amplify the probability of success. But average case algorithms output deterministic result given an input. So we cannot amplify the probability of success by repeating the average case algorithm. Throughout this course, we will mostly focus on randomized algorithms.

2.3 Why is Randomness Useful?

Why do we need randomness in algorithms? Why don't we just use deterministic algorithms? Randomness has several good properties in algorithm design.

1. First, we can use randomness to fight adversaries.

   - The first kind of adversary is worst case input. A good example is quick sort algorithm. If you use a deterministic quick sort algorithm, the worst case running time is $\Omega(n^2)$. But randomized quick sort algorithm achieves expected running time of $O(n \log n)$.
   - For online algorithms, there are another kind of adversaries. Online problems are problems where inputs are given step by step, and the algorithm needs to make the decision based on the current inputs without knowing the future. Here the adversary can change the future inputs based on what choices the algorithm makes currently. In online algorithms, randomness makes adversaries unable to predict the algorithm's future decisions. And there are online problems where the randomized algorithm is provably better than the best deterministic algorithm.
   - The third case is about cryptography. There are lots of tools in cryptography requiring randomness. But we will not talk about that in this course.

   Generally, randomness is helpful for fighting against adversaries because the adversary may know our algorithm but cannot know the randomness in the execution of algorithm.

2. The second nice property of randomness is the idea of sampling.

   - The first thing we can do using sampling is estimation. For example given a whole data set, we want to know the mean of the data set. We don't need to go through the whole date set and instead we can just sample a bunch of entries and compute their mean, which is usually a good approximation of the real mean. Using sampling in estimation can give us sublinear time algorithms because we don't need to read the whole input.
   - Sampling can also help us in compression. Usually a small data set sampled from a large data set can still preserve many properties of the large data set. So we can just keep what we have sampled using small storage and preserve the properties of large data set in the meantime. This is the basic idea of compression. In this course, we are going to talk about two examples of compression, dimension reduction and graph sparsification.
   - We will also talk about two tools for generating samples, Markov chains and random walks.

3. Another benefit of randomized algorithms is that usually they are much simpler than corresponding deterministic algorithms. Also, the analysis of randomized algorithms is usually simpler.

3 Karger's Min Cut Algorithm

Now, let's look at a concrete example of randomized algorithm, Karger's Min Cut Algorithm [Kar93]. In the Min Cut problem, we want to find the cut with least number of edges in an undirected graph. A cut of graph $G = (V,E)$ is a partition $(S,\bar{S})$ of the vertex set V. The number of edges of a cut is the number of edges with end vertices in S and $\bar{S}$ separately.

A trivial deterministic algorithm for Min Cut problem is to run the $s-t$ Cut algorithm for all pairs of vertices and then pick the cut with least number of edges. This algorithm turns out to be very slow because we need to run the $s-t$ Cut algorithm n^2 times and the $s-t$ Cut algorithm is kind of complex.

Karger's algorithm is pretty simple. In each iteration, we just uniformly choose a random edge (u,v) and merge vertices u and v, until there are only two vertices a and b left. Then we let all vertices merged into a as S and let all vertices merged into b as $\bar{S}$, and output this cut $(S,\bar{S})$. Note, in the contraction step, if there are edges between the two vertices, we just delete these self-loops; but if there are vertices connecting to both contracted vertices, we need to keep these parallel edges. It's important to keep parallel edges in Karger's algorithm because parallel edges increase the probability of their end vertices being contracted.

Now, let's analyze the probability of Karger's algorithm outputting a min cut. Let $(S,\bar{S})$ be a min cut of the graph. The algorithm outputs $(S,\bar{S})$ if all $n-2$ contractions are within S or within $\bar{S}$. Let E_i be event that i-th step is good.

$$\Pr[E_i] = 1 - \frac{|\text{min cut}|}{\text{total number of edges in graph}}$$

As we know, the degree of every vertex should be larger than the size of min cut. So,

$$\text{total number of edges} = \frac{\sum_{v\in V}\deg(v)}{2} \geq \frac{|\text{min cut}| \times (\text{total number of vertices})}{2}$$

Thus, we have

$$\Pr[E_i] \geq 1 - \frac{2|\text{min cut}|}{|\text{min cut}| \times (\text{total number of vertices})} = 1 - \frac{2}{n-i+1}$$

Thus, the probability of success is

$$\begin{aligned}
\Pr[\text{Success}] &\geq \Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}] && (1)\\
&= \Pr[E_1]\cdot\Pr[E_2|E_1]\cdots\Pr[E_{n-2}|E_1\cap E_2\cap\cdots E_{n-3}] && (2)\\
&\geq (1-\frac{2}{n})(1-\frac{2}{n-1})\cdots(1-\frac{2}{3}) && (3)\\
&= \frac{2}{n(n-1)} && (4)
\end{aligned}$$

If we run this algorithm $n^2/2$ times, we can reduce the probability of failure to $1/e$. Karger's algorithm is faster than the trivial deterministic algorithm and also much easier to implement.

4 Karger-Stein Algorithm

Karger-Stein algorithm [KS96] is an extension of Karger's algorithm. The basic idea is to perform the contraction procedure until the number of vertices reaches a threshold. The motivation is that with the number of vertices decreases, the probability that an edge from the min cut is contracted increases. So it's good to switch to a deterministic algorithm when the number of vertices gets small.

In Algorithm 1, Min-Cut(G) is an algorithm for solving MIN CUT problem on small graphs; Contract(G,t) will randomly contract edges in G until t vertices left. We can use induction to show that the probability of Karger-Stein's Algorithm outputting a min cut is lager than $1/\log n$. Thus, we can repeat this algorithm $O(\log n)$ times to increase the success probability to a constant larger than $2/3$. For the running time of each run, we can solve the recursion $T(n) = 2T(n/\sqrt{2}) + O(n^2)$ to get $T(n) = O(n^2 \log n)$. Thus, the running time in total is $O(n^2 \log^2 n)$. Remember, for Karger's algorithm, we need to run $O(n^2)$ times to increase the success probability to a constant larger than $2/3$, which costs total running time $O(n^4)$. So, we can see Karger-Stein's algorithm is much faster.

Algorithm 1 Fast-Min-Cut(G)

```
if |V| ≤ 8 then
    return Min-Cut(G)
else
    Let t = ⌈|V|/√2⌉
    Let G1=Contract(G,t); G2=Contract(G,t);
    return min{Fast-Min-Cut(G1),Fast-Min-Cut(G2)}
end if
```

5 Summary

In this lecture, we have introduced the definition of randomized algorithms and compared it with the concept of average case algorithms. We also have talked about the importance of randomness in algorithm design, including fighting adversaries, sampling and giving simpler algorithms. Finally, we have looked at Karger's Min Cut algorithm and its extension, Karger-Stein Algorithm.

References

[Kar93] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30, 1993.

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.