MA 202 Project

Probabilistic Applications in Randomized Algorithms

Supervisor: Prof. Bireswar Das

Anushk Bhana
21110031

Ayush Modi
21110039

Kaushal Kothiya
211100107

Ameya Tajne
21110220

Vedant Kumbhar
21110234

March 5, 2023

Contents

1 Preface

Everyone in the computer theory community is familiar with the idea of randomization. It is not an exaggeration to state that randomization is one of the most popular techniques for algorithm creation right now. The field of randomized algorithms has expanded significantly during the previous decade. At this time, randomized algorithms progressed from being a tool in computational number theory to finding broad use in a wide range of methods. Its expansion has been fueled by two advantages of randomization: simplicity and speed. A randomized algorithm is great many times, either the simplest or the quickest algorithm accessible for various purposes.

We know very little about the input distribution in many cases. Even if we have some information on the distribution, we may not be able to simulate it computationally. As a result, we frequently use randomness and probabilistic analysis as tools for designing and analyzing algorithms by making some of the algorithm's behaviour random.

This paper discusses some important algorithms deliberately and how probability plays a crucial role in them. The algorithms which are discussed in this paper are Min-Cut, Median, and the Miller-Rabin Primality Testing algorithms.

2 Motivation

Randomized algorithms in today's date have a wide range of applications, including computer science, encryption, machine learning, and scientific simulations. We were highly interested in these fields and their applications and wanted to dwell further on these topics, and thus we chose to learn about randomized algorithms that will help us grasp how these approaches might be used to address real-world issues.

With its integration with probability theory, computer science, and algorithm design, randomized algorithms are an interesting topic of study. The passion for these subjects naturally gravitated us toward exploring randomized algorithms in more depth. Probability plays a very integral role in the theory of Randomized Algorithms. Learning about the various applications of probability in Randomized algorithms would enable us to understand how the concepts of probability are applied to solve various problems with the help of these fast randomized algorithms.

Thus, writing a journal on the topic of Probabilistic Applications in Randomized Algorithms would help us reflect on what we have learned about these topics and how it relates to our experiences and understanding. This study will assist us in discovering new concepts, locating fascinating examples, and identifying gaps in our comprehension of the content.

3 Min-cut Algorithm

In a graph, a **cut-set** is a group of edges that, if removed, would split the graph into two or more disconnected graphs. Finding a minimum cardinality cutset in a graph G = (V, E) with n vertices is known as the "minimum cut" issue. Many situations, such as the research of network reliability, might lead to minimum cut issues. The min-cut is the least number of edges that can break before certain machines cannot interact, where nodes are equivalent to the machines in the network and edges are connections between machines[2].

This algorithm mainly works on the idea of random contraction, an edge is selected at random and the edge between two vertices [u,v] is contracted, i.e converted into a single edge. The process is followed until we have two sets of vertices left S and V-S. The new graph can have multiple edges between two vertices but no self-loops are allowed[2].

3.1 Pseudo Code for min-cut algorithm

Input: Graph G = (V, E) with n vertices and m edges
while $n > 2$ do
Contract a random edge e(u, v): merge the vertices and remove self-loops
end while
return Pre-image of the two remaining vertices[3].

3.2 Theorem : The algorithm outputs a correct min-cut set with probability at least $2/n(n-1)$

A min-cut set of G should have a size of k. The graph might have a number of minimum-sized cut-sets. We calculate the probability of only finding one such set C. The remaining is partitioned into S and V-S such that there are no edges connecting the vertices in S to those in V-S if we remove such a set C from the entire set. This is so because in the graph, C is a cut-set. Let's say that after going through the algorithm cycle, two edges only in S or two vertices in V-S are contracted; while edges in C are left unaltered. After n iterations, the algorithm produces a graph with two vertices connected by edges in C. The connections between the vertices in S or V-S that were eliminated during execution were edges. We may therefore draw the conclusion that the algorithm returns C as the minimum cut-set if it never selects an edge of C during any of its n iterations[2].

Let E_i be the event in which the edge contracted in the iteration i is not in C. Let $F_1 = \bigcap_{j=0}^{i} E_j$ then the event that no edge of C was contracted in the first i iterations. We need to compute $Pr(F_{n-2})$.

We start by computing $Pr(E_1) = Pr(F_1)$. Every of the graph's vertices must

have degree k or more because the smallest cut-set contains k edges. The graph must have at least nk/2 edges if each vertex is connected to at least k edges. Out of the collection of all edges, the first contracted edge is selected uniformly at random. Given that C contains k edges and that the graph has at least nk/2 edges, the likelihood that we will not select an edge from C in the first iteration is given by[2]

$$Pr(E_1) = Pr(F_1) \geq 1 - 2k/nk = 1 - 2/n$$

$$Pr(E_2|F_1) \geq 1 - (2k/k(n-1)) = 1 - 2/n - 1$$

Similarly,

$$Pr(E_i|F_{i-1}) = 1 - 2k/k(k-i+1) = 1 - 2/1 - i + n$$

Now computing,

$$Pr(F_{n-2}) = Pr(E_{n-2} \cup F_{n-3}) = Pr(E_{n-2}|F_{n-3}) * Pr(F_{n-3})$$

$$\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right)$$

After simplification,

$$= \frac{2}{n(n-1)}$$

Since the algorithm has a one-sided error, we can reduce the error probability by repeating algorithm. Let us consider we run this min-cut algorithm n(n -1). After n iterations it outputs the minimum size cut-set found amongst all the cycles. The probability that the output is not a min-cut set is bounded by[2],

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

3.3 Analysis of Algorithm

This algorithm randomly selects an edge between any two vertices [u,v], and keeps contracting it until there any two sets of vertices S and V-S which have the minimum number of edges between them. Repetition can help getting better outputs for the algorithm and increases its efficiency drastically.

4 Median Algorithm

Let S be a set of n elements drawn from the universe of ordered elements. The median of elements in S is m where at least $\lfloor n/2 \rfloor$ elements in S which are less than or equal to m and at least $\lfloor n/2 \rfloor + 1$ elements are greater than or equal to m[2].

We can find the median of any set S containing n elements by sorting them in $O(nlog(n))$ time. Many complex algorithms can find the median in $O(n)$ time(for example, Median of median algorithm). But with the help of simpler randomized algorithms, we can find the median in linear time with a smaller constant factor[2].

4.1 Useful Inequalities for randomized median find algorithm

4.1.1 Markov's Inequality

Let $X \geq 0$ be a non-negative random variable (discrete or continuous), and let $k > 0$. Then:

$$\mathrm{P(X} \geq k) \leq \frac{E[x]}{k}$$

Equivalently (plugging in $kE[X]$ for k above):

$$\mathrm{P(X} \geq \mathrm{kE\ [X]}) \leq \frac{1}{k}$$

4.1.2 Chebyshev's Inequality

Let X be any random variable with expected value $\mu = E[X]$ and finite variance $Var(X)$[2]. Then, for any real number $\alpha > 0$:

$$\mathrm{P}(|X - \mu| \geq \alpha) \leq \frac{Var(X)}{\alpha^2}$$

Equivalently (plugging in $\mathrm{k}\sigma for\ \alpha$ above, where $= \mathrm{P}(Var(X))$:

$$\mathrm{P}(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

4.2 Algorithm

The main goal of the algorithm is to find two elements that are close together in sorted order of S and the median lie between them. Let us assume d,$u \in S$ such that:
1. In sorted order of S $d \leq m \leq u$(the median is between d and u)
2. for C=$(s \in S : d \leq s \leq u)$, C=—$n/logn$—(total number of elements between

d and u are small)

To find d and u, we sample with replacement(each element is chosen uniformly random from S, independent of the previous choice). Let R$\lceil n^{\frac{3}{4}} \rceil$ containing elements chosen from S with sampling with replacement.

4.2.1 The Algorithm

1. Pick a set R$\lceil n^{\frac{3}{4}} \rceil$ of elements in S, chosen independently and uniformly at random with replacement.
2. Sort the set R.
3. Let d be the ($\lfloor n^{\frac{3}{4}} - n^{\frac{1}{2}} \rfloor$)th smallest element in the sorted set R.
4. Let u be the ($\lfloor n^{\frac{3}{4}} + n^{\frac{1}{2}} \rfloor$)th smallest element in the sorted set R.
5. By comparing every element in S to d and m, compute the set
C=($s \in S : d \leq s \leq u$)
l_d = (s$\in S : s \leq d$)
l_u = (s$\in S : s \geq u$)
6.*if* $l_d > n/2$ or $l_u > n/2$ then Algorithm fails.
7. if $|c| \leq 4n^{\frac{3}{4}}$ then sort C else Fails.
8. Output the ($\lfloor n/2 \rfloor + l_d + 1$)th element of sorted order of C[2].

4.3 Analysis of Algorithm

Randomized median algorithm counts numbers smaller than l_d and greater than l_u in $O(n)$ time. Since C is small($|C| = n/logn$) it can be sorted in linear time. We just have to find ($\lfloor n/2 \rfloor - l_d + 1$)th element in C, which will be m. Our Algorithm either fails or gives the correct answer in linear time. The algorithm fails for the following events:
1. $\epsilon 1 : Y_1 = |r \in R | r \leq m| < 1/2n^{\frac{3}{4}} - n^{\frac{1}{2}}$
2. $\epsilon 2 : Y_2 = |r \in R | r \geq m| < 1/2n^{\frac{3}{4}} - n^{\frac{1}{2}}$
3. $\epsilon 3 : |C| > 4n^{\frac{3}{4}}$

If one of the above events occurs then our algorithm will fail so the probability of failure of our Algorithm is:

$$\Pr(\epsilon 1) + Pr(\epsilon 2) + Pr(\epsilon 3)$$

If we compute the probability of failure of the algorithm using Markov's and Chebyshev's Inequalities, it will come to:

$$\Pr(\text{Failure}) = \Pr(\epsilon 1) + Pr(\epsilon 2) + Pr(\epsilon 3)$$
$$\Pr(\text{Failure}) \leq n^{\frac{-1}{4}}$$

In this case probability of failure is very less and we can repeat this algorithm

until success. This iterative algorithm never fails, but it has a random running time. Every run of this algorithm is independent, so the success of a run doesn't depend on the previous run. The number of runs until success is a geometric random variable and it has a linear expected running time.

5 The Miller-Rabin Primality Test

This algorithm is used to check whether a given number n is prime or not. We use a very sophisticated randomized algorithm for this which is able to check the primality of a number in polynomial time[7].

5.1 Algorithm

Let n be any positive integer, and we want to determine if n is a prime number. The algorithm for this is as follows:

1. Repeatedly divide the number $(n-1)$ by 2 until the number obtained is not further divisible by 2.
2. Through this, determine the numbers s, and d, such that $(n-1) = 2^s d$. Note that here d is not divisible by 2.
3. Now, pick a number at random from the set $1, 2, ...,$(n-1). Let this randomly chosen number be called b.
4. Evaluate the following numbers:
(b^d mod n),(b^{2d} mod n),($b^{2^2 d}$ mod n), ...,($b^{2^{(s-1)} d}$ mod n)
5. Now, check if any of the numbers calculated in the above step is equal to $(n-1)$. If any one of them is equal to $(n-1)$, then the original number n is prime. If not, then conclude that n is a composite number.

5.2 Analysis of Algorithm

The Miller-Rabin Primality Test is a randomized algorithm because of the second step in the procedure above, in which the algorithm picks a number randomly out of a given set[7].

As this is a randomized algorithm, its effectiveness is heavily dependent on the correctness of this algorithm. Following is the theorem that governs the probability of the algorithm giving a correct result:

5.2.1 Theorem:

If n is a prime number, the above algorithm will give the output as prime with full certainty. If n is a composite number, then the above algorithm will show output as prime with a probability of 1/4.[7]

So then the question here is: How do we know that this algorithm will definitely give us a correct output?

By the above theorem, we can definitely say that if the algorithm gives the output as composite, then the number n is indeed composite. But the issue is

if the input is composite, then still there is a $\frac{1}{4}$ chance that our algorithm will give the output as prime. So how do we tackle this issue?

5.3 Where does probability come into the picture?

To tackle the above-mentioned issue, we repeatedly execute the Miller-Rabin Primality Test algorithm on any given input. In each repetition, the algorithm will select any b from the set at random and will generate an output based no that. By doing this, we will be able to significantly reduce the error margins. This is explained in detail as follows[7].

The probability of getting an incorrect output is 1/4. As each repetition is an independent iteration, the probability of error after k such iterations would be equal to $(1/4)^k$. Thus we can minimize the error probability by just increasing the number of times we execute this algorithm on any given input[7].

To understand this better, let us say we run this algorithm for k=10 times. Then the probability of error would be $(1/4)^{10}$, which is equal to a one in 1000000 probability. An error of this magnitude is considered extremely small and could very well be neglected for all practical applications.

Thus to summarise, on any given input number n, we would run the Miller-Rabin Primality Test algorithm for enough number of times (>= 10). If, in any of these rounds, the output of the algorithm is composite, we can thus say with certainty that the number n is composite for sure. Else if none of the rounds gives the output as composite, then we can say with a sufficiently large probability that the number n is a prime number[7].

5.4 Advantage of choosing this randomized algorithm

The Miller-Rabin Primality Test algorithm is the most commonly used algorithm for detecting whether a given number is prime or not. Even though there is a negligible error factor associated with this algorithm, it is easily preferred over any other deterministic algorithm because of the fact that being a randomized algorithm, this algorithm is extremely fast having just a polynomial running time[7].

6 Repetition Theorem

An important notion in probability theory and randomized algorithms is the Repetition Theorem. It is used to reduce the likelihood of an inaccurate randomized algorithm by repeating the procedure numerous times and taking the majority outcome.

Randomized algorithms are frequently employed to address hard issues in computer science and mathematics, such as optimization, cryptography, and machine learning. Nevertheless, these algorithms rely on random choices, sometimes resulting in inaccurate results. The Repetition Theorem shows how to lessen the likelihood of this happening.

According to the theory, if a randomized algorithm independently runs k times, the probability of it producing the wrong answer falls exponentially with k. In particular, if the algorithm has a probability p of producing the correct answer, the probability of producing the incorrect response after k iterations is given by:

$$\text{P(Wrong answer)} \leq e^{\frac{K*(1-p)^2}{2}}$$

In other words, the more times the algorithm is run, the less likely it is to produce an inaccurate response. This is due to the fact that the probability of all k repetitions giving wrong answers at the same time reduces exponentially with k.

The Repetition Theorem is useful in the construction of error-correcting codes, Monte Carlo simulations, and randomized methods for graph problems, among other things. The repetition theorem, for example, can be utilized in constructing error-correcting codes to lessen the likelihood of a transmitted message being distorted by noise.

The repetition theorem can be used in Monte Carlo simulations to estimate the value of a complicated function by averaging the outcomes of several randomized assessments. This approach has many applications, encompassing physics, engineering, and finance.

To sum up, everything stated so far, the Repetition Theorem is a strong principle that enables us to reduce the likelihood of a randomized algorithm delivering wrong results by repeating it several times. This improves the reliability and accuracy of randomized algorithms, making them a valuable tool in many computer science and mathematics fields.

7 Conclusion

This brings the paper to a conclusion. If you've read it extensively, you'll realize how crucial probability is in randomised algorithms. Each of the algorithms covered in this paper is quite significant. Min-cut algorithms are commonly used in network analysis to discover the network's weakest links. It may be used to optimize network design and routing by finding the lowest number of edges that must be eliminated to disconnect a network. Instead of the mean method and randomised algorithm, the median-based ML technique delivers the best estimate of the Bernoulli distribution function, and probabilistic analysis plays a key part in it. The Randomized Method for Calculating the Median can be used in machine learning to pick features and reduce dimensionality. The approach can assist in selecting the most important characteristics or qualities of a dataset and can be used to decrease the dataset's dimensionality. The Miller-Rabin primality testing algorithm is used in various applications to detect whether or not a number is prime. For example, whenever a website is accessed, this primality test is used to establish a secure connection to the website with your device. Thus for such applications, it is essential that the speed of testing is fast, and this is exactly the reason why a randomized algorithm is used for this.

8 References

[1] H. Pham and D. H. Pham, "A median-based machine-learning approach for predicting random sampling bernoulli distribution parameter," Vietnam Journal of Computer Science, vol. 06, no. 01, pp. 17–28, 2019.

[2] Mitzenmacher, M. and Upfal, E. (2017) Probability and computing: Randomized and probabilistic techniques in algorithms and data analysis. Cambridge; New York ; Port Melbourne ; Delhi ; Singapore: Cambridge University Press.

[3] CS 388R: Randomized Algorithms, Fall 2019 (no date) Randomized algorithms. Available at: https://www.cs.utexas.edu/ ecprice/courses/randomized/fa19/ (Accessed: March 5, 2023).

[4] R. Motwani and P. Raghavan, in Randomized algorithms, Cambridge: Cambridge University Press, 2018.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, in Introduction to algorithms, Cambridge, Massachusett: The MIT Press, 2022.

[6] B. Smith, "This is how prime numbers keep your online shopping secure," ABC News, 19-Jan-2018. [Online]. Available: https://www.abc.net.au/news/science/2018-01-20/how-prime-numbers-rsa-encryption-works/9338876. [Accessed: 04-Mar-2023]

[7] Rao, A. (2019) "Lecture 27: Randomized algorithms: Primality Testing and fingerprinting," Washington University. Available at: https://courses.cs.washington.edu/courses/cse312/19sp/schedule/lecture27.pdf (Accessed: March 5, 2023).