Lecture 7: Randomization in computer science

Rajat Mittal

IIT Kanpur

Deutsch-Jozsa problem showed an advantage of a quantum computer over classical *deterministic* computer in terms of queries. Due to measurements, randomness is intrinsic to the model of quantum computation. It makes more sense to compare quantum computing with randomized computing.

Exercise 1. Randomized computation is a model in which we are allowed to base our computation over random bits in addition to the input. We will give a brief overview in the next section. Go ahead and read the Wikipedia article on randomized algorithms.

You will show in the assignment: there exists a randomized algorithm with constant number of queries for Deutsch-Jozsa problem. So, Deutsch-Jozsa algorithm only gives a constant speed up for quantum vs deterministic algorithm model.

We will introduce randomized algorithms in the next section, let us take a look at approximation algorithms before.

Approximation algorithms: Most of the introductory algorithms you might have seen are all deterministic algorithms, e.g., sorting and network flows. That means, we always get the *exact* answer when the algorithm ends. The task there is to put a bound on the running time of such an algorithm.

Exercise 2. Read about Dijkstra's shortest path algorithm.

For most of the problems we will be concerned with, coming up with a deterministic algorithm is too hard or the bound on running time is too bad. In real/practical life, we accept approximate answers, this leads us to the notion of approximation algorithms.

In general, approximating a solution means our answer is close to the correct answer. If we want it in terms of additive error, the answer by the algorithm is at most ϵ far away from the correct answer. For example, you might want to approximate the value of π up to error .01, that means the output should be in $(\pi - .01, \pi + .01)$ range.

In case of multiplicative error, we want the algorithm to not be away from the correct answer by more than an ϵ factor. Again, for the problem of approximating π, the answer should be in the range $(\pi/2, 2\pi)$.

1 Randomized model of computation

What does it mean to have randomness in our algorithms? Suppose with the input x, algorithm A can also use a random string (say $r \in \{0,1\}^k$ for some k chosen by algorithm). Hence, algorithm A is a function of the combined input x and r. A randomized/probabilistic algorithm is called to be successful, if for every x it outputs the correct answer *for most of the random strings* r.

There are many ways to come up with a random string in the classical case, coin-toss, clock or many others. For the quantum computer, applying a Hadamard on $|0\rangle$ and measuring in standard basis works. Let us take an example of a randomized algorithm.

1.1 How can randomization help

Calculating the value of π is a natural question. We will see a *statistical* way to approximate the value of π (this example is taken from Mitzenmacher and Upfal [2]). Consider a square of area 4 and a circle inside it of radius 1 (look at Figure 1).

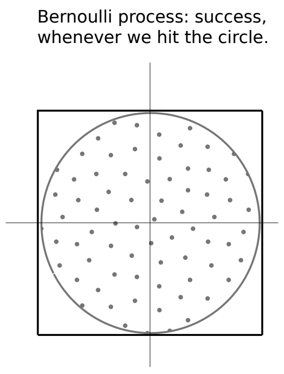

Fig. 1. Hitting a dart at a square.

Exercise 3. If we hit a dart uniformly randomly at the sqaure, what is the probability that it hits the circle?

This is not a difficult question to answer. The area of the square is 4, and the area of the circle is π. So, the probability that we land inside the circle is $\pi/4$. We can come up with a randomized strategy to compute the value of π.

Let X be the indicator random variable that a point lands inside the circle, then we know $E[X] = \pi/4$. Let us pick n random points (uniformly) from the square. Denote the outcome by n independent copies of X, say $X_1, X_2, \cdots, X_n$.

Exercise 4. What do we expect $Y = \frac{1}{n}(\sum_{i=1}^{n} X_i)$ to be?

From linearity of expectation, $E[Y] = \pi/4$. The simple algorithm is: choose n points randomly in the square and say m of them hit the circle; our approximation of π is $4m/n$ (we equate the expectation of Y with our empirical estimate m/n).

Intuitively, as $n \to \infty$, we expect that our estimate will be close to the correct answer with high probability.

Exercise 5. How do we show this formally?

Extra Reading: Formal Proof

From the discussion above, our first guess should be a direct application of law of large numbers.

Exercise 6. What is the variance of X?

Since it is a Bernoulli random variable, the variance is $p(1-p)$, where $p = \pi/4$. Applying law of large numbers,

$$P[\left|Y - \frac{\pi}{4}\right| \geq a] \leq \frac{p(1-p)}{na^2}.$$

In other words, if we want an approximation error less than $4a$ (error of a in estimation of $\pi/4$) and failure probability at most δ, we must hit the square at least $n = \frac{p(1-p)}{\delta a^2}$ times. This is a linear dependence on $1/\delta$ (some might consider it to be too big a number).

Though, realize that the hits to the square were independent, we can improve our bounds on failure probability by using Chernoff bounds.

$$P(|nY - nE[X]| > \epsilon n E[X]) \leq 2e^{\frac{-nE[X]\epsilon^2}{3}}.$$

Cancelling n and using $E[X] = p$,

$$P(\left|Y - \frac{\pi}{4}\right| > \epsilon\frac{\pi}{4}) \leq 2e^{\frac{-np\epsilon^2}{3}}.$$

Like before, if we want an approximation error at most $4a$ (error of a in estimation of $\pi/4$) and failure probability at most δ, then

$$a = \epsilon\frac{\pi}{4} \text{ and } \delta = 2e^{\frac{-np\epsilon^2}{3}}.$$

Substituting ϵ and p,

$$\delta = 2e^{\frac{-4na^2}{3\pi}}.$$

This gives that we should repeat the experiment $n = \frac{3\pi}{4a^2}\ln\frac{2}{\delta}$ times. Notice that we should repeat the experiment only $O(\ln 1/\delta)$ times, instead of $O(1/\delta)$ times (as given by law of large numbers). The dependence on δ has reduced considerably (by application of Chernoff bound).

Note 1. This form of sampling, when we independently sample from a distribution to estimate its expectation, is called Monte Carlo sampling. You can read much more about it from the internet, Wikipedia and other resources.

Notice that we intuitively defined randomized algorithm so that the output is correct on most of the random strings r. What do we mean by the statement, "output is correct on most of the random strings r"? There are more than one ways to formalize this. We already saw two different ways, we could ask the exact solution or an approximate solution.

1.2 Randomized algorithms for problems with Boolean answers

We go back to the question of output criteria. For many problems in computer science, the output is $0, 1$. For example, even for vertex cover, we can ask, is the vertex cover bigger than k (called the search version).

Exercise 7. Can you convert question of finding the min vertex cover into this search version with logarithmic many repetitions?

We will be mainly interested in quantum algorithms for problems with Boolean answers (answer is in $\{0, 1\}$). In such cases, there are different ways to define a randomized algorithm. First Let's allow errors on both sides.

Definition 1 (Bounded error). *An algorithm A accepts the language L in the one sided error model if,*

- *If $x \in L$, then $\Pr_r(A(x,r) = 1) \geq \frac{2}{3}$.*
- *If $x \notin L$, then $\Pr_r(A(x,r) = 0) \geq \frac{2}{3}$.*

Notice that we take the probability over r's and not input x's. In other words, our algorithm should work for *all* x's, it will answer correctly on high fraction $(2/3)$ of r's for any x.

Many a times, a slightly different model, *one sided error* model, is useful.

Definition 2 (One-sided error). *An algorithm A accepts the language L in the one sided error model,*

- *If $x \in L$, then for at least half the r's, $A(x,r) = 1$. In other words, $\Pr_r(A(x,r) = 0) < \frac{1}{2}$.*
- *If $x \notin L$, then $A(x,r) = 0$ for all r's. In other words, $\Pr_r(A(x,r) = 0) = 1$.*

Notice that bounded error model is a weaker notion than one sided error algorithm, i.e., a one sided algorithm is a bounded error algorithm too.

From the definition of one sided error model, if we get 1 then the input is definitely in the language. If we get 0 then x will not be in language with high probability. Note that the probability is computed only over the random strings used in the algorithm. In other words, the probability condition is true for all inputs x (and not just on a high fraction of inputs).

For instance, if we just want our algorithm to work on large fraction of inputs, since number of primes are small, it is easy to come up with an algorithm for testing if a number is prime. We will not call such an algorithm a randomized algorithm.

Exercise 8. Can you think of such an algorithm.

1.3 Example of a randomized algorithm

A randomized algorithm is best illustrated by an example. We will come up with an algorithm for primality testing. The following algorithm is called Miller-Rabin. It is a one sided error randomized algorithm that runs in polynomial time.

The Miller-Rabin algorithm is a one sided error algorithm to test whether an input n is composite or not. It is based on the following simple exercise.

Exercise 9. If $r^2 = 1 \mod n$ where n is a prime, prove that $r = \pm 1 \mod n$.

By Fermat's theorem, we know that $r^{n-1} = 1 \mod n$ if n is prime. Suppose $n - 1 = 2^l g$ where g is an odd number. From the previous exercise,

- either there exists, $0 < l' < l$, for which $r^{2^{l'} g} = -1 \mod n$.
- or $r^g = 1 \mod n$.

Exercise 10. Prove the above assertion.

When n is composite, there still are some r's satisfying the above property. It turns out that there are not many such r's for any composite n. The Miller-Rabin algorithm checks the above condition for a random $0 < r < n$ for an input $n > 2$ [1].

```
if $n = 0 \mod 2$ then
    Output "Composite";
end
$l \leftarrow 0$ ;
$g \leftarrow n - 1$ ;
while $g = 0 \mod 2$ do
    $l \leftarrow l + 1$;
    $g \leftarrow \frac{g}{2}$;
end
/* This finds the form $n - 1 = 2^l g$.                                    */
Pick an $r$ between 0 and $n$.;
Compute $r^g, r^{2g}, \cdots, r^{n-1}$ ;
if $r^{n-1} \neq 1 \mod n$ then
    Output "Composite";
end
$l' \leftarrow l$;
while $l' \geq 1$ do
    if $r^{2^{l'}g} = 1$ and $r^{2^{(l'-1)}g} \neq \pm 1$ then
        Output "Composite";
    end
    $l' \leftarrow l' - 1$ ;
end
Output "Prime";
```

Algorithm 1: Miller-Rabin algorithm

Exercise 11. Where did we use the random bits in this algorithm?

From the discussion before the algorithm it is clear that if the number is prime, the algorithm will output "Prime". If the number is composite, it can be shown that it outputs "Composite" with probability more than $\frac{3}{4}$ [1]. Hence this is a one sided algorithm to check if a number is composite or not. We consider the algorithm to be efficient if its running time is polynomial in the length of the input. In the assignment you will show that this algorithm runs in $poly(\log n)$ time.

1.4 Error amplification

We defined one sided error algorithm by saying that they could fail on positive inputs with probability $\leq \frac{1}{2}$.

Exercise 12. Why did we choose this probability to be $\frac{1}{2}$.

It turns out that this constant $\frac{1}{2}$ is not important. Any constant strictly less than 1 will work. This is because, we are interested in the asymptotics of the running time (in terms of the length of x). That means, we don't care even if the algorithm takes twice or thrice or any constant times the original running time (till the constant does not depend upon the length of x).

Suppose we have an algorithm which succeeds on positive inputs with some probability (say $\frac{1}{4}$). If the algorithm is repeated multiple times on the input x, say k times, then the probability that a positive input x is rejected all the k times is,

$$\left(1 - \frac{1}{4}\right)^k .$$

This quantity can be brought close to any constant $\epsilon > 0$ with constant k. Since this will increase the runtime by only a constant factor (we only care about asymptotics), we can choose any constant bigger than 0 instead of 1/2 for defining one sided error algorithm.

Exercise 13. Convince yourself that the previous argument implies, any constant strictly bigger than 0 will work in the definition of one sided error model.

Most of the time, if not mentioned, we will assume that the algorithm is a bounded error algorithm. Again, as in one sided case, the constant $\frac{2}{3}$ is not important, any constant strictly greater than $\frac{1}{2}$ will work. In this case, we will repeat the algorithm k times and take the majority vote to decide the output. Suppose the original algorithm (the one we are repeating) gives the correct answer with probability more than $\frac{1}{2} + \epsilon$. We assume that ϵ is a constant. Then after repeating it k times, using Chernoff bound, the probability that we get the wrong answer is less than [3],

$$e^{-2\epsilon^2 k}.$$

Exercise 14. Read about Chernoff bound.

Note 2. The number of times we need to repeat the algorithm, k, in bounded error or one sided error case is independent of size of input and only depends on probability we want to achieve. So, k will be a constant.

Note 3. We have defined bounded error randomized algorithms for decision problems, but they can similarly be defined for other problems. The algorithm should compute the required quantity with high probability, where probability is over the random strings.

Approximation vs randomization:
Many people get confused between randomized and approximation algorithms. There is a nice analogy with darts to clarify the difference.

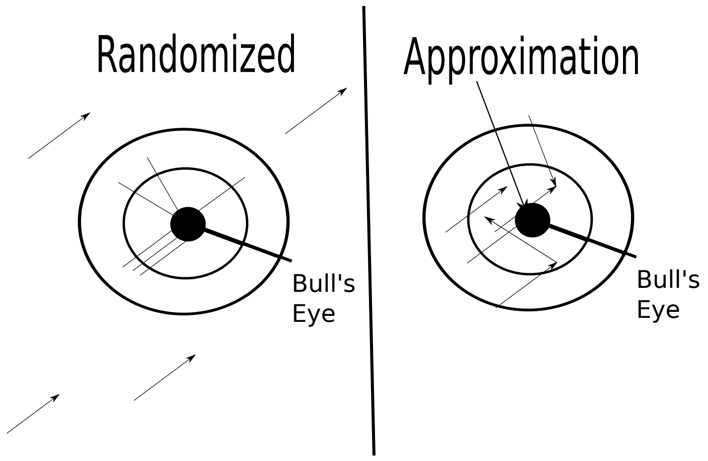

Fig. 2. Difference between randomized and approximation algorithms

Suppose computing a function is equivalent to hitting the bull's eye in the game of darts. Approximation algorithm is a dart player who hits close to the bull's eye all the time. There is no guarantee that she will hit bull's eye. Randomized algorithm is a dart player who hits the bulls eye most of the time. Other times, she may not even hit the board (Fig. 2).

As we saw before, randomization can be introduced for any output criteria. We might want to find an approximate solution. If we find an approximate solution for most of the random choices in the algorithm, it will become an randomized approximation algorithm (for example, approximating the value of π).

Randomness in quantum algorithms: We have introduced classical randomized algorithms, where randomness was inserted by basing decisions on a chosen random string. Notice that quantum algorithms are inherently random because of the definition of measurements. This means that for most of the quantum algorithms, the answer need not be deterministic even if the input is fixed (Deutsch-Jozsa was an exception).

In such cases where the answer is not fixed for a fixed input, we want to succeed with high probability over the inherent randomness in the algorithm. It is natural that we compare quantum algorithms with classical randomized algorithms.

2 Assignment

Exercise 15. Give a constant query randomized one sided error algorithm for Deutsch-Jozsa problem.

Exercise 16. Suppose we have an algorithm which gives the correct answer with probability $\frac{2}{3}$. How many times should we repeat it to get the success probability higher than $1 - 1/n$?

Exercise 17. Show that the running time of Miller-Rabin is polynomial.

Exercise 18. Show that vertex cover is a special case of set cover problem.

References

1. B. Kleinberg. Introduction to algorithms, 2010.
2. M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2017.
3. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge, 2010.