Lecture 7: Randomized algorithms

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 one sided 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. After that, we will see a problem (Bernstein-Vazirani) for which quantum algorithm performs better than the randomized setting.

1 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 and are fine even if the algorithm *almost certainly* outputs the correct answer. This leads us to various relaxations possible in the definition of an algorithm.

1.1 Randomized algorithms

The first relaxation is the one where, 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.

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. First, let us define the *one sided error* model.

Definition 1 (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, $\underset{r}{Pr}(A(x,r) = 0) < \frac{1}{2}$.*
- *If $x \notin L$, then $A(x,r) = 0$ for all r's. In other words, $\underset{r}{Pr}(A(x,r) = 0) = 1$.*

It means, 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 3. Can you think of such an 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 4. 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 5. 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 6. 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.

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

Exercise 7. 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 8. Convince yourself that the previous argument implies, any constant strictly bigger than 0 will work in the definition of one sided error model.

We can also allow errors on both side and further relax the definition of an algorithm.

Definition 2 (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 $\Pr_r(A(x,r) = 0) \geq \frac{2}{3}$.*

Most of the time, if not mentioned, we will assume that the algorithm is a bounded error algorithm. Notice that it is a weaker notion than one sided error algorithm, i.e., a one sided algorithm is a bounded error algorithm too. Even in this case, 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.

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 [2],

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

Exercise 9. Read about Chernoff bound.

Note 1. 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 2. 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.

1.2 Bernstein-Vazirani algorithm

We noticed that Deutsch-Jozsa has a small (constant query) randomized algorithm. Can we show quantum advantage over randomized model of computation? It turns out that a small modification in the problem statement achieves the required result. The modified problem is known as Bernstein-Vazirani problem.

The Bernstein-Vazirani problem is very similar to the Deutsch-Jozsa problem. Suppose we have a promise on the input $x \in \{0,1\}^{2^k}$, that $x_i = i.a$ for some $a \in \{0,1\}^k$. The Bernstein-Vazirani problem is to find this a. Notice that the index i can be interpreted as a binary string of length k (index varies from 0 to $2^k - 1$). These is again an instance of a promise problem, where out of 2^{2^k} possible inputs x, only 2^k inputs are allowed to be given (for each a).

Note 3. Bernstein-Vazirani problem is similar to the Deutsch-Jozsa problem because $a = 0$ case corresponds to constant input and other a's correspond to the balanced input.

Exercise 10. Show that Deutsch-Jozsa algorithm (exactly the same algorithm) will work in this case.

Note 4. After we measure in the standard basis, the output will give us a.

So this problem can also be solved by just one query in the quantum setting. To solve this problem even in the bounded error setting, we need at least $O(k)$ queries. This follows from an information theory argument (one query will give information about at most one bit of a), but the exact argument requires the understanding of information theory.

1.3 Approximation algorithms

We already saw one kind of relaxation, where we needed to find the correct answer with high probability. For the other relaxation, the task will be to optimize some quantity.

An example will illustrate this best. Suppose we are given a universe $U = \{1, 2, \cdots, n\}$ and a collection of subsets of size 2 (for people familiar with graphs, you can think of them as undirected edges), say $S_1, S_2, \cdots, S_m$. The task is to find the minimum cover (a subset $C \subseteq [n]$) such that $C \cap S_i \neq \emptyset$ for all i. This is known as the *vertex cover problem*.

It is known that solving this problem exactly seems to be hard (this problem is NP-complete). Can we at least give a solution which is close to the best (optimal) solution? Let us write a simple greedy algorithm. Define $E = \{S_1, S_2, \cdots, S_m\}$ and $C = \emptyset$ and run the below steps repeatedly till E becomes empty.

- Pick a random set from E, say S.
- Include both elements of S in C.
- Remove all sets from E which share an element with S.

Exercise 11. Show that the solution given will not be more than twice the size of the optimal vertex cover. Hint: You will need at least one element from each S you have picked.

Note 5. Even though we have picked sets randomly, it is not really a randomized algorithm. For all choices of random strings, we get a factor 2 approximation.

Notice that we did not try to solve the best possible solution, it was enough to come up with something which is not much bigger than the optimal solution (if we wanted to maximize a quantity, the solution should not be much smaller than the optimal).

Exercise 12. Read about the set cover problem and a $\log n$ approximation algorithm for it.

To define an optimization problem precisely, given an input G, the algorithm should compute the minimum value of $f(S)$ for all possible choices of valid solutions S. For example, in case of vertex cover, G will be the universe and collection of subsets, and S will be a cover.

Exercise 13. What will be f for vertex cover?

An approximate algorithm A is the one which outputs an answer $A(G)$ which is not very large compared to the optimal answer.

Definition 3 (Approximation algorithm). *An approximation algorithm A for a minimization problem f on input G satisfies,*

$$f(A(G)) \leq c \cdot f(S^0).$$

Where S^0 is the optimal choice, $A(G)$ is the answer given by the algorithm and c is some constant greater than 1. *c is known as the* approximation factor.

Exercise 14. What will the formulation of approximation algorithm for maximization problem?

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

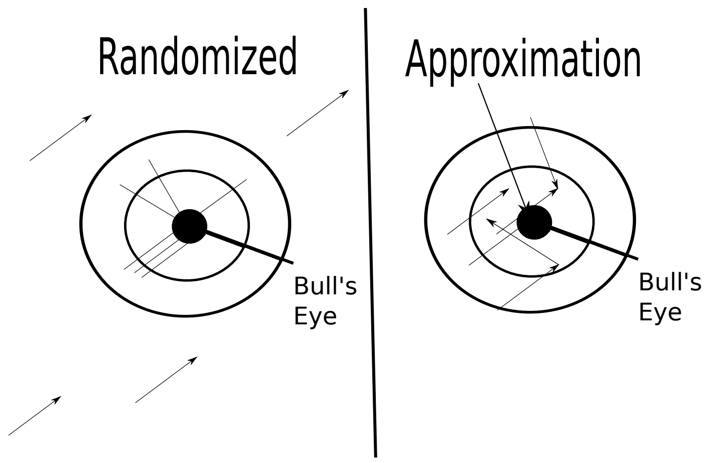

Fig. 1. Difference between randomized and approximation algorithms

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. 1).

It is even possible to consider randomized approximation algorithms.

Exercise 15. What could be the definition of an approximate randomized algorithms?

2 Assignment

Exercise 16. Give a constant query randomized one sided error algorithm for Deutsch-Jozsa problem.

Exercise 17. 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 18. Show that the running time of Miller-Rabin is polynomial.

Exercise 19. Show that vertex cover is a special case of set cover problem.

References

1. B. Kleinberg. Introduction to algorithms, 2010.
2. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge, 2010.