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 randomized algorithm model.

In randomized computation, the main idea is to allow algorithm to use randomness and *relax the output criteria*. Before randomized algorithms, we look at another relaxed output condition, leading to approximation algorithms.

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.

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.

Exercise 2. Suppose you want to find the maximum in the list. What kind of approximation could you be happy with?

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 should be 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 with a factor 2, the answer should be in the range $(\pi/2, 2\pi)$.

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).

Exercise 3. If we hit a dart uniformly randomly at the square, 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 $X_1, X_2, \cdots, X_n$.

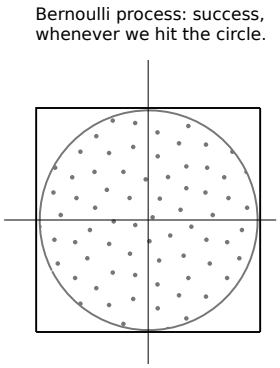

Fig. 1. Hitting a dart at a square.

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,

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

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.

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

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

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

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.

2 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.

Note 2. 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.

For the deterministic algorithms, we need to always return the correct answer and the cost of the algorithm is counted as the worst case cost for any input x. Analogously, we can say that a randomized algorithm should always answer correctly and the cost of the algorithm is worst case over all pairs x, r.

Exercise 7. Show that such a strict criteria of correctness and cost will lead us back to deterministic algorithm.

To have a different model of computation, we need to relax our correctness/cost criteria. There are two different ways to capture the performance of the algorithm.

- Relaxing cost (Las Vegas style): the cost of the algorithm is defined to be the expected cost (over r). Though, the algorithm answers correctly on all r's.
- Relaxing correctness (Monte Carlo style): the algorithm gives the correct answer on most of the r's, but the cost is worst case over all r's. In general, we say that the algorithm should give the correct answer on 2/3-rd fraction of r's. We will discuss this 2/3 constant later.

It turns out that we can convert any algorithm with relaxed cost into an algorithm in the relaxed correctness model with just a constant factor increase. You will show this in the assignment(Exercise 29). If not stated, our default choice would be to relax correctness and consider worst case cost over all possible r's.

Approximation vs randomization:

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

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).

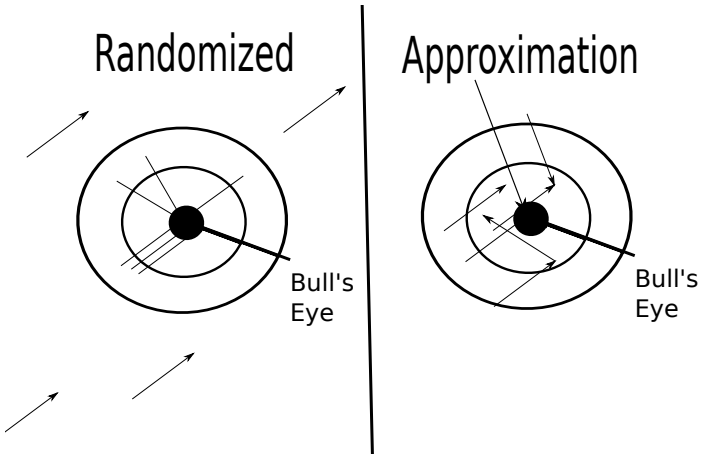

Fig. 2. Difference between randomized and approximation algorithms

2.1 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 in the set $\{0, 1\}$. For example, even for vertex cover, we can ask, is the vertex cover bigger than k (called the search version).

Exercise 8. 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 bounded 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 9. Can you think of such an algorithm.

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.

2.2 Error amplification

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

Exercise 10. 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 11. 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.

Exercise 12. Why should the constant be bigger than 1/2?

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 $e^{-2\epsilon^2 k}$ (refer to any standard probability textbook with Chernoff bound, for example [3]).

Exercise 13. Read about Chernoff bound.

Note 3. 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 4. 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.

3 A few examples

A randomized algorithm is best illustrated by an example.

3.1 Computing recursive majority

Our first example will be in the query model [4] (attributed to Ravi Bopanna). Suppose we are interested in computing $f_d := Maj_3^d$; in other words, it is the recursive composition of the majority function on 3 bits. The input contains 3^d bits and we are interested in minimizing the number of bits queried. Please look at Figure 3.

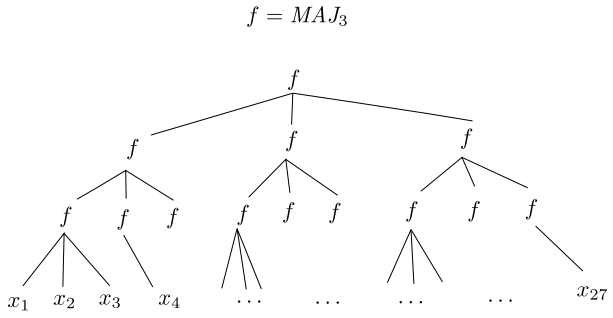

Fig. 3. The function $f_3 = Maj_3^3$

Exercise 14. Show that if we want to deterministically answer f_1, we need 3 queries.

It can be shown that you need at least 3^d queries (all inputs) for solving f_d deterministically (hint: induction). We first construct a randomized algorithm which always works, but its expected number of queries is *small* (Las Vegas style).

Exercise 15. How will you solve f_d using randomness, so that its expected cost is small for every input? What about f_1?

Notice that f_d can be viewed as a tree with d depth, each node having 3 children. The strategy seems pretty natural. At any point in the tree (node), we randomly pick two children and compute the output. If both of them give the same value, we don't need to compute the third children and save cost in expectation.

Let $c(f_d, x)$ be the expected cost of this strategy on input x (and $c(f_d)$ be the worst case expected cost). If input x gives the same value to all 3 children,

$$c(f_d, x) \leq 2c(f_{d-1}).$$

The expensive case is when two of the children have the same value, but the third one is different,

$$c(f_d, x) \leq (1/3)\,(2c(f_{d-1})) + (2/3)\,(3c(f_{d-1})) \leq (8/3)\,c(f_{d-1}).$$

In other words, for every x,

$$c(f_d) \leq (8/3)\,c(f_{d-1}).$$

So, the expected cost is $(\frac{8}{3})^d$ instead of 3^d, a significant saving. The following trick converts expected cost into worst cost. The cost increases by a constant factor, and error increases by a little bit (can be made small by repetition, Section 2.2).

Suppose the expected cost is E. Run the old algorithm $10E$ times. If the algorithm stops, we answer according to the old algorithm. If it does not stop, we answer arbitrarily. By Markov's inequality, the algorithm will not stop with probability at most $1/10$. So the new algorithm's worst case cost is 10 times the original. The error has only increased by an additive factor of $1/10$ (assuming we always answer incorrectly when answering arbitrarily).

This is a general strategy for converting expected cost into worst case cost. Hence, we can make the usual randomized algorithm with worst case cost $O((\frac{8}{3})^d)$.

Exercise 16. Notice that $n = 3^d$, where n is the number of variables. We know that deterministically we need 3^d queries, express number of randomized queries in terms of n.

Another example can be constructed by recursively combining $NAND$ function on 2 bits [4].

3.2 Extra reading: Miller-Rabin algorithm for primality testing

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 17. 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 18. 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 19. 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.

4 Separation between quantum and randomized computation

An asymptotic separation between randomized and quantum model of computation can be shown by slightly modifying the Deutsch-Jozsa problem (with essentially the same algorithm). That algorithm is known as Bernstein-Vazirani algorithm.

The number of queries needed for Bernstein-Vazirani (in both randomized as well as quantum model) are very small as compared to the size of the input. We would like an asymptotic separation where the number of queries in the randomized model are close to the arity of the function. This will lead us to Simon's algorithm.

4.1 Extra reading: 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 5. 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 20. Show that Deutsch-Jozsa algorithm (exactly the same algorithm) will work in this case.

Note 6. 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.

Notice that if the input size is n, the number of quantum queries are 1 and number of randomized queries are $\log n$. Even though the separation is exponential, an additive term of $\log n$ can explain this speedup. Ideally, we would like to get a separation where the randomized queries are comparable to the input size, enters *Simon's algorithm.*

4.2 Simon's algorithm

We will discuss Simon's algorithm now, arguably a better separation between the two models.

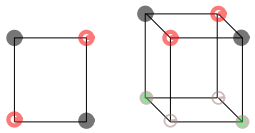

Fig. 4. What is the pattern in the colors of the vertices?

Take a look at the *square* given above. If I ask you to figure out the pattern of color of the vertices, it is pretty easy. The diagonal vertices are colored the same. When asked the same question about the cube, can you see the pattern? We will identify left with right (side), up with down (vertical) and inside with outside (horizontal) movement. Then, you can see that moving a horizontal and a side step from any vertex, gives another vertex of the same color.

What happens if the *hypercube* is of dimension 500? Is it going to be still easy to find the movement direction? If we find two vertices of the same color, the direction will be clear. Although, there are 2^{500} vertices, and intuitively we need to look at around $\sqrt{2^{500}}$ vertices (birthday paradox) to find two vertices of the same color. It turns out that this problem can be solved much more efficiently on a quantum computer. Let us define *Simon's problem* formally.

Input: A string of length n, $x = (x_1, x_2, \cdots, x_n)$, where each x_i (not just i) is a k bit binary string ($n = 2^k$). You can also think of it as a function f from $\{0,1\}^k$ to $[n]$. So, x_i gives the function value $f(i)$ as before. Here, we use i for both, as an index between 1 to n and as a binary string of length k.

Promise: There exists a hidden shift $s \in \{0,1\}^k$ such that $x_i = x_j$ *if and only if* $i = j \oplus s$. Again, notice that when we write $i = j \oplus s$, i, j, s are viewed as binary strings and *XOR* is taken entry-wise. On the other hand, when writing x_i, i corresponds to the index (a number) between 1 and n.

Output: We need to find the hidden shift s.

Note 7. The difference from previous settings is that the function range is $[n]$ instead of $\{0,1\}$. Actually, the range is not that important, it should be big enough to make sense of the promise.

We will assume that one query to the input gives the entire string x_i. The query oracle is in the standard form, $O_x|i,0\rangle \to |i,x_i\rangle$, where both parts of the register have k qubits.

Note 8. Even if we assume that there are $k2^k$ indices and every query outputs a bit, the exponential separation will not be affected.

Exercise 21. Why is the above statement true?

We will begin with our usual strategy of creating a superposition over all inputs and computing their function value in superposition.

So, the algorithm starts in the state $|0,0\rangle$, where both $|0\rangle$'s are in the state space of k qubits. Applying Hadamard transform to the first part will create a superposition over all inputs,

$$\frac{1}{\sqrt{2^k}}\sum_{j=0}^{n-1}|j,0\rangle.$$

Now we query the oracle (of the input) and get,

$$\frac{1}{\sqrt{2^k}}\sum_{j=0}^{n-1}|j,x_j\rangle.$$

Notice that the second part of the register is same iff the indices in the first part differ by s. If we measure the second register at this stage, we get the state

$$\frac{1}{\sqrt{2}}(|j\rangle+|j\oplus s\rangle), \tag{1}$$

for some j.

We have ignored the second part of the register, since that part of the register has been measured. How is this state useful to us?

Note 9. For algebra enthusiasts: Another viewpoint about Simon's problem will be of help. Our input is on $\mathbb{Z}_2^n$ and the hidden shift s generates a subgroup K_s of order 2 in this group. The promise can be reinterpreted as, $x_i = x_j$ if and only if i and j belong to the same coset with respect to K_s. We need to find the subgroup (or its generator) K_s to solve Simon's problem.

Exercise 22. Convince yourself that the above statement is true.

So, after measurement, we get a state of the form given in Eq. 1, called *coset states*. What information can we derive from these coset states about K_s?

Applying Hadamard transform (Fourier transform on $\mathbb{Z}_2^n$) on the coset state in Eq. 1,

$$\frac{1}{\sqrt{2^{k+1}}}\left(\sum_{l\in\{0,1\}^n}((-1)^{j.l}+(-1)^{(j\oplus s).l})|l\rangle\right).$$

Exercise 23. When will state $|l\rangle$ have amplitude non-zero?

Looking at the formula, it is non-zero if and only if $s.l = 0 \mod 2$. Measuring this state, we will get an l such that $s.l = 0 \mod 2$. You can easily check that each such l has an equal probability to appear after the measurement.

Note 10. l specifies a character of $\mathbb{Z}_2^n$ given by $\chi_l(a) = (-1)^{l.a}$. Intuitively, Fourier transform puts weight only on those characters which are trivial on K_s.

Applying Hadamard on the coset states and measuring, we get an l for which $s.l = 0 \mod 2$. So, one coset state gives us one linear equation for s. If we get $k-1$ *linearly independent* l's, we can find s. The obvious choice is to create lot ($O(k)$) of cosets states and hope that we have $k-1$ linearly independent l's, then s can be found using Gaussian elimination.

Exercise 24. What is Gaussian elimination. What field are we working over?

The only thing to show is, if we have $O(k)$ coset states then we get $k-1$ linearly independent l's with high probability.

Exercise 25. Suppose we have h linearly independent l's. What is the probability that we get a linearly independent l in the next coset state measurement?

Note 11. We are working over the field $\mathbb{F}_2$.

This probability is at least half, because the vector space spanned by h vectors will cover 2^h vectors out of 2^k vectors. Hence, at every step, we get a linearly independent vector with probability at least $\frac{1}{2}$. So in $O(k)$ iterations, we will have $k-1$ linearly independent vectors with high probability.

Exercise 26. Make the previous statement precise and prove it.

Hence with $O(k)$ queries (since it takes one query to prepare a coset state) we can give a randomized algorithm for Simon's problem on a quantum computer.

Simon [5] also showed that any randomized algorithm will take at least $\Omega(\sqrt{2^k})$ queries to solve this problem with high probability (calculations similar to birthday paradox). Hence, we get an instance where quantum and classical query complexity differ exponentially.

We saw two quantum algorithms in the last few lectures. Deutsch-Jozsa provided exponential separation between quantum and classical computers in the deterministic setting. Bernstein-Vazirani shows speed-up in the randomized setting (1 quantum query vs $\log n$ randomized queries, where n is the size of the input). *Simon's algorithm* gives exponential separation in the randomized setting ($\log n$ as compared to $\Omega(\sqrt{n})$).

A note of caution, the speed-ups are in the query model, they do not imply exponential separation in the usual circuit model.

5 Assignment

Exercise 27. Give a constant query randomized one sided error algorithm for Deutsch-Jozsa problem.

Exercise 28. 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 29. Suppose a randomized algorithm A gives the correct answer on all r's and has expected cost E. Convert it into an algorithm with worst case cost $10E$ introducing at most $1/10$ error. Hint: Use Markov's inequality.

Exercise 30. Show that the running time of Miller-Rabin is polynomial.

Exercise 31. Show that vertex cover is a special case of set cover problem.

Exercise 32. Why don't we consider the model of randomized algorithms where cost and correctness, both are relaxed. That means, the cost is the expected cost and we only need to be correct on most of the r's. Hint: A constant factor increase in the cost of the algorithm can be ignored.

Exercise 33. Suppose in the Simon's problem, $x_j = x_i$, iff there exist $v \in V$ for which $j = i \oplus v$. Here V is a subspace of $\mathbb{Z}_2^n$. What l will we get at the end of the Simon's algorithm?

Hint: The usual case corresponds to the one dimensional subspace.

References

1. B. Kleinberg. Introduction to algorithms, 2010. `http://www.cs.cornell.edu/courses/cs4820/2010sp/handouts/MillerRabin.pdf`.
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.
4. M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. *27th Annual Symposium on Foundations of Computer Science (sfcs 1986), Toronto, ON, Canada, 1986*, pages 29–38, 1986.
5. D. R. Simon. On the power of quantum computation. *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on: 116–123*, pages 116–123, 1994.