C S 358H: Intro to Quantum Information Science

Sayam Sethi

November 2024

Contents

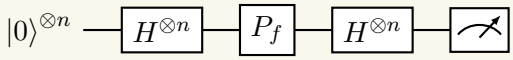
1 Generalized Deustch-Jozsa Problem

Recall that in the Deustch-Jozsa problem we are given oracle access to an unknown function $f\colon \{0,1\} \to \{0,1\}$ and wish to determine if it is constant or balanced. In the generalized Deustch-Jozsa problem we are given oracle access to a function $f\colon \{0,1\}^n \to \{0,1\}$, a function that maps n-bit strings to a single bit. We're promised that the function is either constant or balanced, where balanced means that it has an equal number of 0 and 1 outputs (2^{n-1} of each).

Question 1.1

Question. *Design a quantum algorithm that can determine whether f is constant or balanced using only a single query to f and prove that your algorithm works.*

Proof. We can use the same circuit as in the Deustch-Jozsa problem, but with n qubits:



Here P_f is the Phase oracle and we assume that the ancilla is initialized to $|1\rangle$. Now, consider the state of the system before the final Hadamard gate:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} |x\rangle \tag{1}$$

Notice that $|\psi\rangle$ is equal to $|\phi\rangle = \pm|+\rangle^{\otimes n}$ in the case when f is constant. When f is balanced, it is easy to see that the state is orthogonal to $|\phi\rangle$ (because there are equal number of $+1$ and -1 in $|\psi\rangle$ when f is balanced). Therefore, on applying the final Hadamard gate and measuring all qubits, we either end up with 0^n if f is constant or we end up with $s \neq 0$ when f is balanced. □

Question 1.2

Question. *Show that any classical deterministic algorithm for this problem requires $\Omega(2^n)$ queries. Therefore, quantum computers give an exponential speedup over deterministic classical computers for this problem.*

Proof. We construct an adaptive oracle to show that any classical deterministic algorithm would require at least $2^{n-1}+1$ queries on at least one function. The oracle is defined as:

Adaptive Oracle: $\mathcal{A}$

Note that the oracle maintains a table of the inputs that have already been queried and their outputs

1. For the first 2^{n-1} unique queries, return 0 (and store it in the table).
2. For the next 2^{n-1} unique queries, return 1 for all queries or return 0 for all queries (with equal probability). Also store the query in the table.
3. If a query is repeated, return the same value as the previous time using the table.

Figure 1: An adaptive oracle implementation to show worst case time complexity for any classical deterministic algorithm

Notice that the above oracle is a valid implementation of the function f and thus, can be used as an oracle to an classical algorithm that solves the Deustch-Jozsa problem. Since $\mathcal{A}$ is a valid implementation of such an f, we call it $f_{\mathcal{A}}$ (where $f_{\mathcal{A}}$ is either balanced or constant), any classical algorithm that solves Deustch-Jozsa should also give the correct answer on $\mathcal{A}$. However, it is impossible for any classical deterministic algorithm to output the correct answer without having queried on $2^{n-1}+1$ different inputs since $\mathcal{A}$ is randomized. Any classical deterministic algorithm cannot determine whether $f_{\mathcal{A}}$ is balanced or constant until the last query. Thus, any classical deterministic algorithm requires $\Omega(2^n)$ queries.
Note that we needed to define an adaptive oracle since the classical deterministic algorithm itself could have been adaptive on the queries, where the algorithm decides the next query based on the output it sees. □

Question 1.3

Question. *Explain why, nevertheless, this speedup is not very impressive, in the sense that it is not an exponential quantum speedup over all classical computing for this problem. Give a full explanation and analysis of the any runtime or likelihood of success involved in your explanation.*

Solution. Even though a deterministic classical algorithm requires $\Omega(2^n)$ queries, a randomized classical algorithm can solve the problem with high probability using only super-constant queries. Let the number of queries be c. We can now compute the probability of success after c queries. We define the algorithm as:

Classical Randomized Algorithm for Deustch-Jozsa Problem

1. Query f on c random inputs.
2. If all outputs are the same, output constant. Otherwise, output balanced.

Figure 2: A classical randomized algorithm that succeeds with very high (non-negligible) probability

Now the probability of success for Protocol 2 is 1 when f is constant. In the case when f is balanced, we can compute the probability of failure as:

$$\begin{aligned}\Pr\left[f \text{ is balanced} \wedge \text{ failure}\right] &= 2 \times \left(\frac{2^{n-1}}{2^n} \cdot \frac{2^{n-1}-1}{2^n-1} \cdot \frac{2^{n-1}-2}{2^n-2} \cdots \frac{2^{n-1}-c+1}{2^n-c+1}\right) \\ &\leq 2 \times \frac{1}{2^c} = \frac{1}{2^{c-1}}\end{aligned} \tag{2}$$

Now we can compute the total probability of success as

$$\begin{aligned}\Pr\left[\text{success}\right] &= \Pr\left[f \text{ is constant} \wedge \text{ success}\right] + \Pr\left[f \text{ is balanced} \wedge \text{ success}\right] \\ &\geq \frac{1}{2} \cdot 1 + \frac{1}{2}\left(1 - \frac{1}{2^{c-1}}\right) = 1 - \frac{1}{2^c}\end{aligned} \tag{3}$$

Therefore, as long as we make super-constant queries, we can make the probability of success arbitrarily close to 1. □

2 The Birthday Paradox

Your favorite local radio station is running a new give-away contest that works as follows: Each day at 5pm the phone lines open up to the public to call in and leave their name and birth-date which is then added to a running list. The contest runs until a matching birth-date (month and date) between any two contestants on the list is found. At that point the contest closes and everyone on the list up to that point is a winner. Assume that every birthday is equally likely and that no contestants are born during a leap year (so that we can ignore Feb. 29^{th}).

As usual, you must show or explain your work for each part. You are free to use numerical software of your choice to help solve the problem

Question 2.1

Question. *What is the minimum number of people who need to call in before the probability of a match being found is at least 50%?*

Solution. Note: For all parts, this question is interpreted as trying to find the probability that we get a match with p probability (where $p = 50\%$ or 99% depending on which part the answer is for) at or before the n^{th} person calls. This probability is different from the probability that the station succeeds at exactly the n^{th} call and fails for all $i < n$.
We compute the probability that no match happens until the n^{th} person calls. This will be 1 minus the probability that we want. The probability can be computed as:

$$\begin{aligned} \Pr[\text{no match}] &= \frac{\binom{365}{n} \cdot n!}{365^n} \\ \implies \Pr[\text{match}] &= 1 - \frac{\binom{365}{n} \cdot n!}{365^n} \end{aligned} \tag{4}$$

Therefore, the probability of a match is at least 50% when $n = 23$ (on plugging in Python). □

Question 2.2

Question. *How about the minimum number of people needed before there is a 99% chance of the contest coming to a close?*

Solution. On plugging in the numbers in Python again, the probability of a match is at least 99% when $n = 57$. □

Question 2.3

Question. *Imagine after running the contest for a few days the station decides they aren't being generous enough and so change the rules as follows: Instead of the contest ending as soon as there's a match found between any two contestants on the list it now ends when a match is found between specifically the first caller of the day and any other contestant.*
Under these new rules what is the minimum number of people needed before there is a 50% chance of the contest closing? How about for a 99% chance?

Solution. Again, similar to Question 2.1, we can compute the probability of no match happening until the n^{th} person calls. This will be 1 minus the probability that we want. The probability can be computed as:

$$\begin{aligned} \Pr\left[\text{no match}\right] &= \left(\frac{364}{365}\right)^{n-1} \\ \implies \Pr\left[\text{match}\right] &= 1 - \left(\frac{364}{365}\right)^{n-1} \end{aligned} \tag{5}$$

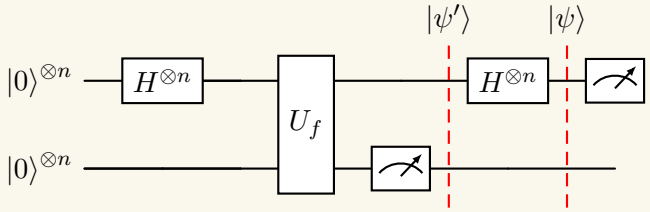
Therefore, the probability of a match is at least 50% when $n = 254$ and it is at least 99% when $n = 1680$ (on plugging in Python). □

3 Two Secrets

Question 3.1

Question. *Suppose the function $f : \{0,1\}^n \to \{0,1\}^n$ is such that there are two n-bit secret strings s and t, where that $s \neq t$ and neither are the all zeros string, so that $f(x) = f(y)$ if and only if $x \oplus y$ is either 0, s, t, or $s \oplus t$.*
Give a quantum algorithm which can find an $a \in \{0,1\}^n$ such that $a \cdot s = a \cdot t = 0$, and $a \neq 0$. Explain/prove that it works.

Solution. We use the same circuit as Simon's problem to solve this problem:



We can write the state $|\psi'\rangle$ as:

$$|\psi'\rangle = \frac{|x\rangle + |x \oplus s\rangle + |x \oplus t\rangle + |x \oplus s \oplus t\rangle}{2} \otimes |f(x)\rangle \text{, for some } x \in \{0,1\}^n \tag{6}$$

Now, on applying the Hadamard gate and ignoring the ancilla qubits, we get:

$$\begin{aligned} |\psi\rangle &= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} \frac{(-1)^{y \cdot x} |y\rangle + (-1)^{y \cdot (x \oplus s)} |y\rangle + (-1)^{y \cdot (x \oplus t)} |y\rangle + (-1)^{y \cdot (x \oplus s \oplus t)} |y\rangle}{2} \\ &= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{y \cdot x} \frac{1 + (-1)^{y \cdot s} + (-1)^{y \cdot t} + (-1)^{y \cdot (s \oplus t)}}{2} |y\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{y \cdot x} \frac{\left(1 + (-1)^{y \cdot s}\right)\left(1 + (-1)^{y \cdot t}\right)}{2} |y\rangle \end{aligned} \tag{7}$$

Therefore, the only states that will have non-zero amplitudes are those where $y \cdot s = y \cdot t = 0$. Therefore, on measuring the state $|\psi\rangle$, we will get a state y such that $y \cdot s = y \cdot t = 0$ with certainty. One thing to note is that $|0\rangle$ is also one of the states, however, the probability of obtaining $|0\rangle$ is only $\frac{1}{2^{n-2}}$. Thus, if we obtain the state $|0\rangle$, we can just run the circuit again and should succeed with a non-negligible probability that approaches 1 as n is increased. Additionally, there exists no non-trivial solutions when $n = 2$ since the probability of obtaining $|0\rangle$ is 1 in that case. □

4 Bernstein-Vazirani

In the Bernstein-Vazirani problem, recall that we're given oracle access to a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$. We're promised that there exists a "secret string" $s \in \{0,1\}^n$ such that $f(x) = s \cdot x \bmod 2$ for all x. The problem is to recover s. The Bernstein-Vazirani algorithm solves this problem with just a single quantum query to f. Consider a variant of this problem where we are promised that $f(x) = s \cdot x \bmod 2$ for *at most* a $(1-\epsilon)$ fraction of the inputs x, and that $f(x) = (s \cdot x + 1) \bmod 2$ for the remaining ϵ fraction of inputs.

Question 4.1

Question. *Calculate a upper bound on the probability that a single run of the Bernstein-Vazirani algorithm nevertheless succeeds in recovering s. Show your work.*

Solution. If we apply the noisy phase oracle in the BV circuit, we get the following state after the Oracle application:

$$\begin{aligned}
|\psi'\rangle &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \text{non-noisy}} (-1)^{s\cdot x} |x\rangle + \sum_{x \in \text{noisy}} (-1)^{s\cdot x+1} |x\rangle \right) \\
&= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \text{non-noisy}} (-1)^{s\cdot x} |x\rangle - \sum_{x \in \text{noisy}} (-1)^{s\cdot x} |x\rangle \right) \\
&= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^n} (-1)^{s\cdot x} |x\rangle - 2 \cdot \sum_{x \in \text{noisy}} (-1)^{s\cdot x} |x\rangle \right)
\end{aligned} \tag{8}$$

Now on applying the Hadamard on $|\psi\rangle$, we get:

$$\begin{aligned}
|\psi\rangle &= |s\rangle - \frac{2}{\sqrt{2^n}} \left(\sum_{x \in \text{noisy}} \left((-1)^{s\cdot x} \cdot \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x\cdot y} |y\rangle \right) \right) \\
|\psi\rangle &= |s\rangle - \frac{2}{\sqrt{2^n}} \left(\sum_{x \in \text{noisy}} \left((-1)^{s\cdot x} \cdot \frac{1}{\sqrt{2^n}} \left((-1)^{x\cdot s} |s\rangle + \sum_{y \in \{0,1\}^n \setminus s} (-1)^{x\cdot y} |y\rangle \right) \right) \right) \\
&= \left(1 - \frac{2}{2^n} \left(\sum_{x \in \text{noisy}} 1 \right) \right) |s\rangle + |\phi\rangle \text{, where } \langle \phi | s \rangle = 0
\end{aligned} \tag{9}$$

Therefore, the amplitude of $|s\rangle$ is bounded by:

$$\Pr\left[|psi\rangle \text{ measures } s\right] = \left(1 - \frac{2}{2^n} \left(\sum_{x \in \text{noisy}} 1 \right) \right)^2 \leq \left(1 - \frac{2}{2^n} \cdot \epsilon \cdot 2^n \right)^2 = (1-2\epsilon)^2 \tag{10}$$

□

Question 4.2

Question. *Explain what happens when $\epsilon = 1/2$. Is there a reason why, in some sense, no algorithm can possibly succeed at recovering s in that case?*

Solution. When $\epsilon = 1/2$, we get that the probability of recovering s is 0. This is because for every string that is mapped correctly, there is another string that is mapped incorrectly and therefore, the amplitudes associated with $|s\rangle$ cancel out.
No algorithm can succeed in recovering s since f acts as a perfectly random bit generator given any input x and therefore it no longer acts like a BV function that computes the parity on some subset of bits of x.
Additionally, this because the noisy function f in some sense acts like a perfect bit commitment scheme, perfectly hiding the bit associated with the presence or absence of noise, i.e., presence of noise indicates the bit 1 and absence of noise indicates the bit 0. The opening is the random input x, with the key being s. □