Quantum Computing Notes
Prof. Hamzeh Roumani
EECS, York University, Toronto, Ont., Canada

THE DEUTSCH-JOZSA ALGORITHM

1. The Problem

Consider a function *f:* $\{0,1\}^n$ --> {0,1}. Its domain has 2^n elements and its co-domain has 2. You can think of *f* as a 0/1-valued function of n 0/1 variables. In programming terms, *f* is a boolean function that takes n boolean parameters. And if you treat the n arguments of *f* as the bits of the binary representation of some integer x, then *f* can be thought of as a function that maps an integer in [0,N-1] to either 0 or 1, where $N=2^n$. We assume that *f* is provided as a black-box U_f (an *oracle*) that implements it in hardware.

Given that *f* is either *constant* or *balanced* (the *promise*), determine which case it is. A function is constant if it maps all elements in its domain to the same value and is balanced if it maps half of them to 0 and the other half to 1. It is one way or the other, guaranteed.

2. Examples

n=1: f(0)=0, f(1)=0 → constant
n=1: f(0)=1, f(1)=1 → constant
n=1: f(0)=0, f(1)=1 → balanced
n=1: f(0)=1, f(1)=0 → balanced

n=2: f(0,0)=0, f(0,1)=0, f(1,0)=0, f(1,1)=0 → constant
n=2: f(0,0)=1, f(0,1)=1, f(1,0)=1, f(1,1)=1 → constant
n=2: f(0,0)=0, f(0,1)=0, f(1,0)=1, f(1,1)=1 → balanced
n=2: f(0,0)=0, f(0,1)=1, f(1,0)=0, f(1,1)=1 → balanced
n=2: f(0,0)=1, f(0,1)=0, f(1,0)=0, f(1,1)=1 → balanced
n=2: f(0,0)=0, f(0,1)=1, f(1,0)=1, f(1,1)=0 → balanced
n=2: f(0,0)=1, f(0,1)=0, f(1,0)=1, f(1,1)=0 → balanced
n=2: f(0,0)=1, f(0,1)=1, f(1,0)=0, f(1,1)=0 → balanced

3. A Classical Algorithm

Since we have no direct access to *f* (its formula or its circuit), all we can do is send arguments to the oracle and examine its returns. For example, we can send all 0s to get *f(0,0,…0)*. Next, we evaluate *f(0,0,…1).* If these two returns are different, we are done! The function in that case must be balanced as it is clearly not constant. On the other hand, if they are equal, we cannot draw any conclusion, and hence, must make more queries. The question then becomes: how many queries do we need to make *in the worst case* before we can draw a conclusion?

The answer depends of course on n, the number of variables in *f*, and it grows exponentially with it. This problem is therefore intractable, i.e. no efficient, deterministic (classical) algorithm exists for solving it. In fact, even for n is as small as 300, the number of needed oracle queries exceeds the number of atoms in the observable Universe!

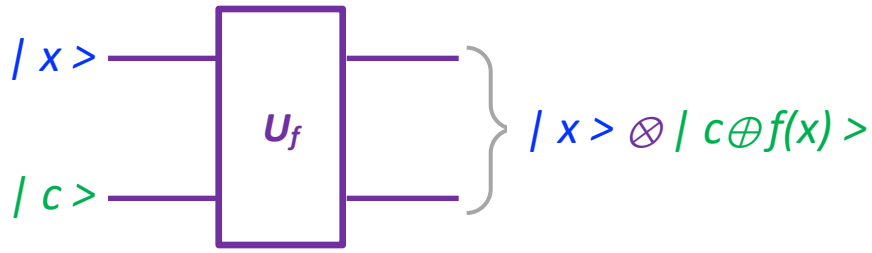
- For n=3, how many different constant functions can there be?
- For n=3, how many different balanced functions can there be?
- Show that $2^n / 2 + 1$ queries are needed in the worst case in the above classical algorithm.
- Show that the complexity of the classical algorithm above is $\Theta(2^n)$.
- Argue that the query complexity of any deterministic classical algorithm is $\Omega(2^n)$.

4. The Quantum Advantage

The explosive growth of the classical algorithm stems from the enormous size the domain of *f* (2^n elements) and the fact that we need to evaluate *f* at over half of them. Superposition, in the quantum realm, allows us to feed the oracle a superposition of all those 2^n possibilities and get a return that captures all values of *f* in one oracle query! This seems to suggest that a quantum algorithm may be able to solve this problem in O(1) instead of $O(2^n)$. We will see shortly that this is indeed the case, and hence, the quantum advantage is quite manifest for this problem. It should be noted that this advantage stems from superposition and is therefore novel--it has no classical counterpart. In particular, it is *not* like parallel computing (e.g. 2^n circuits that compute different values of *f* in parallel) because we have only one circuit in our datapath.

5. The Idea

Computing *f* at a superposition of states does indeed evaluate it at all those states in one shot, but the individual function values are inaccessible to us because they appear in a superposition, and once we measure, we only "see" one of them. For example, take the n=1 case (a function of one variable) and feed the oracle a superposition of the two possible arguments, $|x\rangle = (1/\sqrt{2})\,[\,|0\rangle + |1\rangle\,]$, with $|c\rangle = 0$:



In this case, the output would be:

$|\text{output}\rangle = (1/\sqrt{2})\,[\,|0\rangle \otimes |f(0)\rangle + |1\rangle|f(1)\rangle\,]$

So, yes, both *f(0)* and *f(1)* were computed with just one query but upon measuring, the state will collapse to one or the other, so we can only learn one function value (and we don't even get to choose which). Hence, we need to manipulate the quantum state, before measuring, so that the sought global property (constant versus balanced) can be measured. To that, let us pick |c> = |−> = (|0> − |1>)/√2 in the hope of creating a discriminator for this property. For this |c>, the output would be:

$$|\text{output}\rangle = (1/2)\,[\,|0\rangle \otimes |f(0)\rangle - |0\rangle \otimes |\sim f(0)\rangle + |1\rangle \otimes |f(1)\rangle - |1\rangle \otimes |\sim f(1)\rangle\,]$$
$$|\text{output}\rangle = (1/2)\,[\,|0\rangle \otimes (|f(0)\rangle - |\sim f(0)\rangle) + |1\rangle \otimes (|f(1)\rangle - |\sim f(1)\rangle)\,]$$

We not that if *f(0)* is 0 then its complement would be 1; otherwise, it would be 0. And similarly for *f(1)*. Hence:

$$|f(0)\rangle - |\sim f(0)\rangle = (-1)^{f(0)}\,(|0\rangle - |1\rangle) \quad \text{and} \quad |f(1)\rangle - |\sim f(1)\rangle = (-1)^{f(1)}\,(|0\rangle - |1\rangle)$$

Substituting in the above and factoring out yields this |output>:

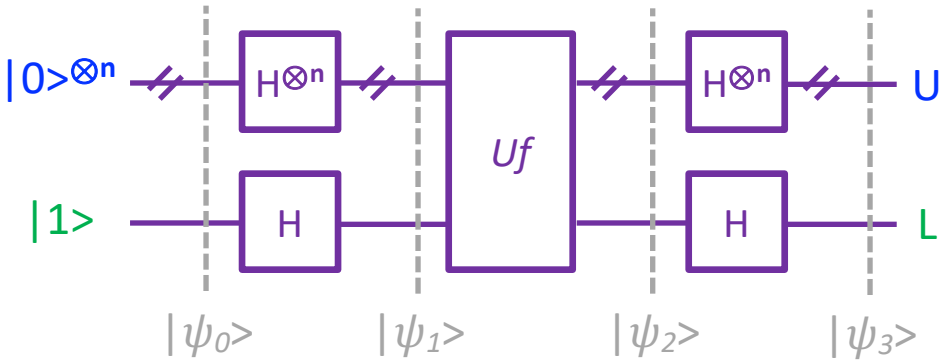
$$(1/2)\,[\,(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\,] \otimes (|0\rangle - |1\rangle) = (1/\sqrt{2})\,[\,(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\,] \otimes |-\rangle$$

With this choice of |c>, the function values became phases that modulate the phases of the two basis states of the upper qubit, and this fleshes out the global property of the function. Indeed, if the function is constant, the two modulating phases would be the same, and the state of the upper qubit becomes |+>. And if the function is balanced, the two phases become opposite, and the state of the upper qubit becomes |−>. These two possible states are orthogonal, and hence, we can distinguish them with a single measurement. The query complexity is thus indeed O(1) for this n=1 case.

- Does the above n=1 algorithm un-compute its ancilla qubit |c>?
- Instead of using |c>=|−>, start with |c>=|0> and add a single-qubit gate (before the U_f oracle) to make the lower input |−>. In addition, make sure the ancilla qubit is un-computed.
- Instead of measuring in the |+>,|−> basis, we like to stick to the standard |0>,|1> basis. Show that this can be achieved by adding a single-qubit Hadamard gate to the upper output of the oracle before measuring.
- Work out the n=2 case and verify that the algorithm still works with O(1) query complexity.

6. The General Case

The input consists of n+1 qubits, n in the upper register and 1 in the lower, all initialized to |0>. Superposition is created in all qubits thru Hadamard gates and is then fed to the U_f gate that implements *f*. The gate's output undergoes a second set of Hadamard gates before we arrive at the final output, which contains n qubits in the **u**pper register U and 1 in the **l**ower L.



$$|\psi_0> = |0> \otimes |0> \otimes |0> \otimes \dots \otimes |0> \otimes |1>$$

$$|\psi_1> = \frac{|0> + |1>}{\sqrt{2}} \otimes \frac{|0> + |1>}{\sqrt{2}} \otimes \frac{|0> + |1>}{\sqrt{2}} \otimes \frac{|0> + |1>}{\sqrt{2}} \dots \otimes |->$$

$$|\psi_1> = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k> \otimes |->$$

$$|\psi_2> = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k> \otimes \frac{|f(k)> - |\sim f(k)>}{\sqrt{2}}$$

$$|\psi_2> = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k> \otimes (-1)^{f(k)} \frac{|0> - |1>}{\sqrt{2}}$$

$$|\psi_2> = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{f(k)} |k> \otimes |->$$

$$|\psi_3> = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{f(k)} \frac{1}{\sqrt{2^n}} \sum_{l=0}^{2^n-1} (-1)^{k.l} |l> \otimes |1>$$

$$|\psi_3> = \frac{1}{2^n} \sum_{l=0}^{2^n-1} \left(\sum_{k=0}^{2^n-1} (-1)^{k.l} (-1)^{f(k)} \right) |l> \otimes |1>$$

The step before last (|ψ_2> to |ψ_3>) involves applying H to all n+1 qubits. For the lower qubit, this transforms |−> to |1>. For the upper n qubits, represented by the ket |k>, this transforms every |0> to (|0> + |1>)/√2 and every |1> to (|0> − |1>)/√2. When these are multiplied out, we get a sum of 2^n terms of the form ±|l>. The sign is negative whenever we use an odd number of 1s in |k>. These 1s appear in |l> at the same positions as in |k>. Hence, you get a −1 when k and l share an odd number of 1s. The symbol l.k denotes the number of common 1s between l and k; i.e. 1s in the same positions. Note that |ψ_3> consists of a sum of 2^n terms each of which involves a sum of 2^n terms.

In the final step, we reversed the order of the two summations so we can group all 2^n occurrences of each |l> ket. These occurrences appear with a sign that depends on the product $(-1)^{k.l}$ x $(-1)^{f(k)}$. If the function *f* is constant then the relevant phase is $(-1)^{k.l}$, which is +1 when k has an even number of 1s and -1 otherwise. In other words, the |l> states with even number of 1s cancel the ones that have an odd number of 1s. The only state that survives is the one without any 1s; i.e. the state l=0. On the other hand, if *f* is balanced then the 0 state would not survive because when l=0, $(-1)^{k.l}$ is always 1 (because l.k = 0). In this case, the |0> state cancels because the other phase, $(-1)^{f(k)}$, is +1 as many times as −1.

In conclusion, we simply measure the upper qubit. If it is |0>, *f* is constant; else, it is balanced.

- Show that if z=|00> then $H^{\otimes 2}|z\rangle = \frac{1}{2}(|00\rangle + |01| + |10\rangle + |11\rangle)$
  Hint: Recall that H|0>=(1/√2)(|0>+|1>) and H|1>= H|0>=(1/√2)(|0>−|1>)
- Show that if z=|01> then $H^{\otimes 2}|z\rangle = \frac{1}{2}(|00\rangle - |01| + |10\rangle - |11\rangle)$
- Show that if |z> is an arbitrary 2-qubit basis state then $H^{\otimes 2}|z\rangle = \frac{1}{2}\Sigma_k (-1)^{z.k}|k\rangle$ $(k = 0..3)$
- Show that the |0> state that survives when *f* is balanced is correctly normalized.
- Work out the n=3 case and verify that |0> is a discriminator of constant vs balanced.

7. A Non-Deterministic Classical Algorithm

If we settle for a *randomized* classical algorithm, one whose conclusion is mostly--but not always-- correct, we can do much better than $O(2^n)$. Rather than evaluating *f* at exponentially many arguments, we can pick a small sample of the domain and base our conclusion on it. If the sample is chosen randomly and uniformly, we can increase our confidence in the conclusion by increasing the sample size. For example, let us pick a sample of only 10 elements and evaluate the function at them. If any two are the same, we declare the function balanced; else we declare it constant. The probability of us being wrong in the constant case is about $(\frac{1}{2})^{10}$ (for large n), which is less than 0.1%. Hence, only O(1) queries are needed to get an answer that is correct with high probability.

- Derive the relation between the success probability and the number of queries needed in the worst case for the randomized classical algorithm above.

- Look up the definition of the complexity classes P and EQP and show that the Deutsch problem defines a relation between them.
- Look up the definition of the complexity classes BPP and BQP and show that the Deutsch problem does *not* define a relation between them.

Remarks

- This algorithm was proposed by David Deutsch and Richard Jozsa c. 1992.
- The *"Walsh-Hadamard"* transform W is a generalization of the single-qubit Hadamard transform H: it applies H to each qubit in a multi-qubit state. In other words, $W = H^{\otimes n}$.
- The phrase *"phase kickback"* refers to cases in which applying a gate to one qubit leads to a phase manifesting in another. For example, the U_f gate normally keeps its upper input $|x\rangle$ unchanged and changes its lower input $|c\rangle$ to $|c \oplus f(x)\rangle$. But in the circuit above, the lower qubit "kicked" its change to the upper output and stayed unchanged. Indeed the lower qubit remained $|-\rangle$ and kicked a phase of $(-1)^{f(x)}$ to the upper qubit. (Compare $|\psi_1\rangle$ to $|\psi_2\rangle$.)
- This algorithm employs the so-called *quantum parallelism* by computing all values of *f(x)* in one shot. The result appears as a superposition of states each of which involves a value of x together with a value of *f* at that x. The algorithm manipulates the phases of these terms in such a way that exposes the sought property. Specifically, the terms interfere destructively (i.e. cancel out) if *f* has a particular global property.
- This success of this algorithm can be traced back directly to the first pillar of quantum computing: *superposition*. Its circuit has the same characteristics as our Apple Gedanken (or its implementation in a Mech-Zehnder interferometer); namely: a fork to induce superposition (a splitter), and a merge (a 2nd splitter) to allow the superposition components to interfere.
- Note that *entanglement*, the second pillar of quantum computing, does not play a key role in this algorithm. The two outputs of U_f seem entangled but phase kickback de-entangles the two registers. Hence, the algorithm's power rests squarely on superposition.