Deutsch-Jozsa Algorithm

Aaron Baxter

April 19, 2020

1 About the Creators

The Deutsch-Jozsa algorithm was created by quantum physicist David Deutsch and mathematicion Richard Jozsa. Deutsch is considered to be the one the founding fathers of quantum computing. Born in Haifa, Israel, he attended the University of Cambridge and the University of Oxford. He currently teaches at the University of Oxford and is the founding member of the Centre for Quantum Computation at Clarendon Laboratory at Oxford. He currently does not accept a paid position at Oxford, where he is considered a "non-stipendiary Visiting Professor of Physics at the University of Oxford". He has various accolades, notably, the Micius Quantum Prize in 2018 as well as being elected a Fellow of the Royal Society in 2008.

Richard Jozsa is a mathematician from Australia. Josza studied at the University of Oxford. He is currently a physics professor at the University of Cambridge. Josza is credited as one of the founders of quantum teleportation. He is the holder of the Leigh Trapnell Chair in Quantum Computing. Like Deutsch, he was elected a Fellow of the Royal Society in 2019.

2 Introduction

The Deutsch-Jozsa algorithm was created in order to show that a quantum computer can solve the same algorithm faster than classical computer can. The Deutsch-Jozsa algorithm is considered by many to be the "Hello World!" of quantum algorithms. The function for the algorithm is as follows:

Definition 2.1. The function for the algorithm s.t. $n \in \mathbb{Z}^+$:

$$f : \{0,1\}^n \longrightarrow \{0,1\}$$

We are promised that the function is either **constant** or **balanced**.

Definition 2.2. F is a **constant function** if and only if $f(x) = 0$ for every $x \in \{0,1\}^n$ or $f(x) = 1$ for every $x \in \{0,1\}^n$. For example:

x	f(x)
00	0
01	0
10	0
11	0

Table 1: output 0

x	f(x)
00	1
01	1
10	1
11	1

Table 2: output 1

Definition 2.3. F is a **balanced function** if and only if $f(x) = 0$ for half of all $x \in \{0,1\}^n$ and $f(x) = 1$ for half of all $x \in \{0,1\}^n$. For example:

x	f(x)
00	0
01	0
10	1
11	1

Table 3: output 0,1

x	f(x)
00	0
01	1
10	0
11	1

Table 4: output 0,1

3 Classical Runtime of the Algorithm

In order to find whether the algorithm is balanced or constant in classical time, we would need to check $f(x)$ for every bit. Let n be the number of qubits represented in the algorithm. Thus 2^n = number of bits.

Theorem 3.1. *The worst case number of steps for classically computing the Deutsch-Jozsa algorithm takes* $2^{n-1} + 1$ *steps.*

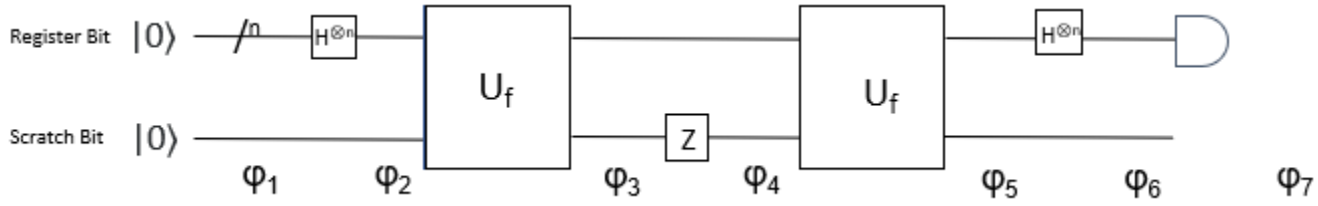
Example 3.2. Let $n = 2$. Let the Deutsch-Jozsa algorithm contain a function where the function output ($f(x)$) of $|00\rangle$ and $|01\rangle$ is 0.

x	f(x)
00	0
01	0
⋮	⋮

Remember, the Deutsch-Jozsa algorithm *promises* us that the function is either constant or balanced. At this point, we cannot determine if the function is constant or balanced. If $x = |10\rangle$ and $f(x) = 0$, then we must conclude that the function is constant. If $f(x) = 1$, then we can determine the function is balanced. This would mean one more than half of all

the bits were compared. We can make a conclusion once we have compared one more than half. By Theorem 3.1, since $n = 2$ then $2^1 + 1 = 3$. Thus at the worst case when $n = 2$, it will take three bits to tell us if the function is constant or balanced. The range of steps to compute the algorithm when $n = 2$ is $2 \leq x \leq 2^{n-1} + 1$

4 Circuit Construction of Algorithm



The *Register* bits are passed through a Hadamard gate which puts them into a superposition state. Both the *Scratch* and *Register* bits are passed through the oracle function. The *Scratch* bit's work is passed through a Z gate and then passed through the oracle function again in order erase the work done to it. The *Register* bits are then passed through a Hadamard gate and their bits are read. The scratch bit is essentially "thrown out". The lowercase phis represent the stages of the circuit. *Eisuke Abe's* presentation on the Deutsch-Jozsa Algorithm [1] demonstrates the equations below needed to interpret the circuit:

- $H^{\otimes n} |x\rangle = \frac{1}{2^{n/2}} \sum_{x=0} |x\rangle = \frac{1}{2^{n/2}} \sum_z (-1)^{x \cdot z} |z\rangle$
- $U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$
- $Z |y\rangle = (-1)^y |y\rangle$

thus:

- $\varphi_1 = |0\rangle |0\rangle$
- $\varphi_2 = \frac{1}{2^{n/2}} \sum_x |x\rangle |0\rangle$
- $\varphi_3 = \frac{1}{2^{n/2}} \sum_x |x\rangle |f(x)\rangle$
- $\varphi_4 = \frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} |x\rangle |f(x)\rangle$
- $\varphi_5 = \frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} |x\rangle |0\rangle$

[1] https://www.appi.keio.ac.jp/Itoh$_g$*roup/abe/pdf/qc*3.*pdf*

- $\varphi_6 = \sum_z \sum_x \frac{(-1)^{f(x)+x\cdot z}}{2^n} |z\rangle |0\rangle$

We use this equation to classify the function $\sum_x \frac{(-1)^{f(x)}}{2^n} |z\rangle$

- $\varphi_7 = \sum_x \frac{(-1)^{f(x)}}{2^n} = \begin{cases} 0, & \text{balanced} \\ \pm 1, & \text{constant} \end{cases}$

Example 4.1. If the function is **constant**, then $f(x) = 0$ and the solution is equal to 1. We will show this below where $n = 3$.

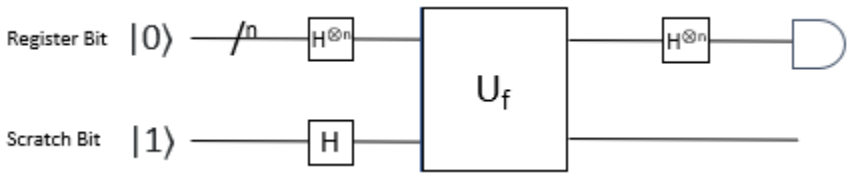
$$\sum_{x=0}^{7} \frac{(-1)^{f(x)}}{2^3} = \frac{(-1)^0 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0}{8} = 1$$

Example 4.2. If the function is balanced, then $f(x) = 0$ half of the time, and $f(x) = 1$ the other half. The solution is equal to 0. Because half of the $-1s$ have a 0 as an exponent, those will equal 1. When they exponent is a 1, they will equal -1. Because there are half of each, they offset to equal 0.We will show this below where $n = 3$.

$$\sum_{x=0}^{7} \frac{(-1)^{f(x)}}{2^3} = \frac{(-1)^0 + (-1)^1 + (-1)^0 + (-1)^1 + (-1)^0 + (-1)^0 + (-1)^1 + (-1)^1}{4} = 0$$

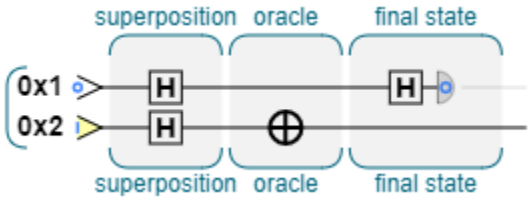
5 Simplified Deutsch-Jozsa circuit

If the algorithm only takes in 2 qubits, the circuit of the Deutsch-Josza algorithm can be simplified. The scratch bit is initialized at 1 instead of starting at 0. The READ gate will read the output of $(f(0) \oplus f(1))$. If the READ gate reads a 0 and $f(x)$ equals 0 for all 0s or 1 for all 1s, then the function is **constant**. If the READ gate reads a 1 and $f(x)$ equals 1 half of the outputs and 0 the other half, then the function is **balanced**.



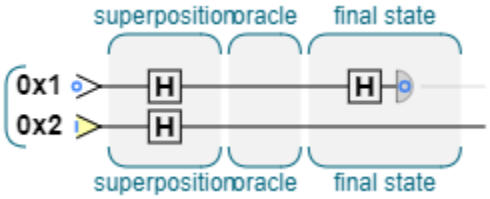
Let's represent the oracle with logic gates and their matrix equivalent. I will be using the QCEngine to show circuits which demonstrate how the matrix and logic gates are implemented. Constant functions can be represented using two different oracles. Balanced functions can also be represented using two different oracles. Note that in the outputs of the multi-qubit registers, the phase carries no bearing on whether the function is constant or balanced.

Example 5.1. In this **constant** circuit, we use the NOT gate on $0x2$.



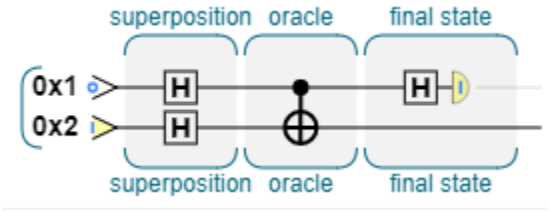
The NOT gate on $0x2$ can be represented in a 2-qubit system in a matrix as $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. We will do an example with this matrix later on. The output on a multi-qubit system contains only 0s in the 2^0 column. Since the READ gate only reads from the 2^0 column($0x1$), the circuit will read 0 100% of the time, thus **constant**. Output:

Example 5.2. In this **constant** circuit, the oracle function does not contain a logic gate.



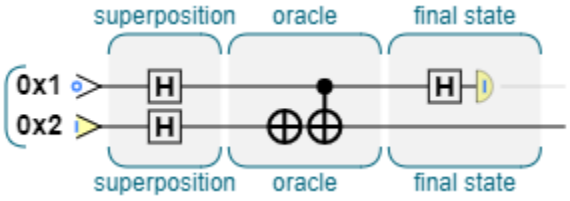
Because we don't use a logic gate, we can represent this through an Identity matrix $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ The output on a multi-qubit system contains only 0s in the 2^0 column. The READ gate only reads from the 2^0 column ($0x2$), therefore, the circuit will read 0 100% of the time, thus **constant**. Output:

Example 5.3. In this **balanced** circuit, we use a CNOT gate with target on $0x2$.

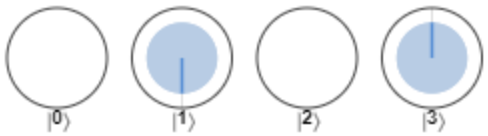


The CNOT gate with target on $0x2$ can be represented in a 2-qubit system in matrix $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. The output on a multi-qubit system contains only 1s in the 2^0 column. The READ gate only reads from the 2^0 column ($0x1$), therefore, the circuit will read 1 100% of the time, thus **balanced**. Output:

Example 5.4. In this balanced circuit, we use a NOT gate on $0x2$ and CNOT gate with target on $0x2$ and control on $0x1$.

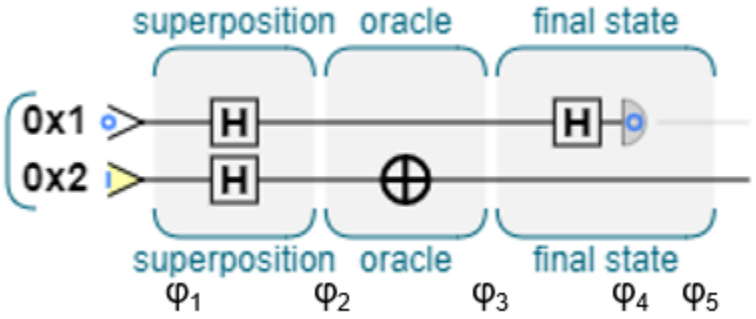


The oracle's gates can be represented in a 2-qubit system in matrix $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. The output on a multi-qubit system contains only 1s in the 2^0 column. The READ gate only reads from the 2^0 column, therefore, the circuit will read 1 100% of the time, thus **balanced**. Output:



6 Example

Lets do an example with two qubits. Let there be a constant function st the oracle function is represented by a NOT gate on $0x2$. We can represent the circuit by:



We will do the matrix multiplication and solve this in phases as we did when we interpreted the circuit.

- φ_1 : $|10\rangle$ can be represented with the column vector $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$
- φ_2 : when we put the qubits in a superposition state, we have to mutiply the column vector by the Hadamard matrix:

$$\begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ -1/2 \end{bmatrix}$$

- φ_3 : Applying the NOT gate's matrix (on $0x2$ to the new column vector, we get:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} -1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

- φ_4 : We then apply the Hadamard gate to only $0x1$. We form the Hadamard's matrix by $I_2 \otimes H$ where H is the 2x2 Hadamard matrix. We multiply the resulting matrix to the column vector:

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 1/\sqrt{2} \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} -1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \\ 0 \end{bmatrix}$$

- φ_5 : Now we read the $0x1$ bit. Let's set up the table of possible bits for 2 qubits and bold the bits represented in our example.

0x2	0x1
-0	**0**
0	1
1	**0**
1	1

Since the read function represents what the output of $f(0) \oplus f(1)$ and the output equals 0, the function in our example is **constant**.

7 QCEngine code used to create circuits

```
qc.clearOutput()
qc.reset(2)
qc.had()
var x = qc.read() //creating number 0-3 to determine oracle function
qc.reset(2)
qc.write(2)
superposition()
var y = oracle(x)
state(y)

function superposition() {
    qc.label('superposition')
    qc.nop()
    qc.had()
    qc.nop()
    qc.label()
}

function oracle(x){
    qc.label('oracle')
    qc.nop()
    switch(x) {
        case 0:
            break
        case 1:
            qc.not(0x2)
            break
        case 2:
            qc.cnot(0x2,0x1)
            break
        case 3:
            qc.not(0x2)
            qc.cnot(0x2,0x1)
            break
    }
    qc.nop()
    qc.label()
    return x

}

function state(y){
    qc.label('final state')
```

```
    qc.nop()
        switch(y) {
        case 0:
            qc.had(0x1)
            qc.print('Constant\n')
            qc.print('Output: ' + qc.read(0x1))
            break
        case 1:
            qc.had(0x1)
            qc.print('Constant\n')
            qc.print('Output: ' + qc.read(0x1))
            break
        case 2:
            qc.had(0x1)
            qc.print('Balanced\n')
            qc.print('Output: ' + qc.read(0x1))
            break
        case 3:
            qc.had(0x1)
            qc.print('Balanced\n')
            qc.print('Output: ' + qc.read(0x1))
            break
    }
    qc.nop()
    qc.label()
}
```

References

[1] Abe, E. (2005, March 22). Deutsch-Jozsa Algorithm. Retrieved from https://www.appi.keio.ac.jp/Itoh_group/abe/pdf/qc3.pdf

[2] Deutsch, D. (n.d.). Home. Retrieved from http://www.daviddeutsch.org.uk/

[3] Gharibian, S. (2015). Lecture 6: Deutsch's Algorithm. Retrieved from http://www.people.vcu.edu/ sgharibian/courses/CMSC491/notes/Lecture 6 - Deutsch's algorithm.pdf

[4] Hui, J. (2019, April 3). QC-Quantum Algorithm with an example. Retrieved from https://medium.com/@jonathan_hui/qc-quantum-algorithm-with-an-example-cf22c0b1ec31

[5] Institute, S. (n.d.). Richard Jozsa. Retrieved from https://simons.berkeley.edu/people/richard-jozsa

[6] javafxpert. (2017). Quantum Computing Exposed: Understanding Deutsch-Jozsa. Retrieved from https://slides.com/javafxpert/quantum-computing-exposed-understanding-deutsch-jozsa/15

[7] Kopczyk, D. (2019, January 27). Quantum algorithms: Deutsch's algorithm. Retrieved from http://dkopczyk.quantee.co.uk/deutschs-algorithm/

[8] Rigetti Computing. (2017, November 15). James Weaver: Implementing the Deutsch-Jozsa Algorithm with Rigetti Forest. Retrieved from https://www.youtube.com/watch?v=WLcNrikK2vw

[9] University of Cambridge. (n.d.). Department of Applied Mathematics and Theoretical Physics. Retrieved from http://www.damtp.cam.ac.uk/person/rj310

[10] Watrous, J. (2006, January 26). Lecture 4: Quantum Teleportation; Deutsch's Algorithm. Retrieved from https://cs.uwaterloo.ca/ watrous/LectureNotes/CPSC519.Winter2006/04.pdf