Workshop on Quantum Computation using IBM Q

Salvador E. Venegas-Andraca

Tecnológico de Monterrey, Escuela de Ingeniería y Ciencias

Table of Contents

1 Deutsch-Jozsa Algorithm

Problem statement

We're given a black box quantum computer known as an **oracle** that implements some function

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

Problem statement

We're given a black box quantum computer known as an **oracle** that implements some function

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

i.e. it takes $n-$digit binary values as input and produces either a 0 or a 1 as output for each such value. We are *promised* that the function is either **constant** (0 on all outputs or 1 on all outputs), or **balanced** (returns 1 for half of the input domain and 0 for the other half); the task is to determine if f is constant or balanced by using the oracle.

Problem statement

What is the meaning of $f : \{0,1\}^n \to \{0,1\}$?
Let's see what happens when $n = 1$:

$$f : \{0,1\}^{n=1} \to \{0,1\} \text{ means:}$$

$$\text{since } \{0,1\}^1 = \{0,1\},$$

f can take the values:

$$f(0) = 0, f(0) = 1, \qquad f(0) = 0, f(0) = 1$$
$$f(1) = 0, f(1) = 1, \qquad f(1) = 1, f(1) = 0$$

Problem statement

Let's see what happens when $n = 2$:

$$f : \{0,1\}^{n=2} \to \{0,1\} \text{ means:}$$

$$\text{since } \{0,1\}^2 = \{(0,0),(0,1),(1,0),(1,1)\},$$

f can take the values:

$$\begin{array}{ll} f(0,0)=0, f(0,0)=1, & f(0,0)=0, f(0,0)=1 \\ f(0,1)=0, f(0,1)=1, & f(0,1)=1, f(0,1)=0 \\ f(1,0)=0, f(1,0)=1, & f(1,0)=0, f(1,0)=1 \\ f(1,1)=0, f(1,1)=1, & f(1,1)=1, f(1,1)=0 \end{array}$$

et cetera

Problem statement

$$f : \{0,1\}^{n=3} \to \{0,1\} \text{ means:}$$

since $\{0,1\}^3 = \{(0,0,0),(0,0,1),\ldots,(1,1,0),(1,1,1)\}$,

f can take the values:

$$\begin{array}{ll}
f(0,0,0) = 0, f(0,0,0) = 1, \ldots & f(0,0,0) = 0, f(0,0,0) = 1 \\
f(0,0,1) = 0, f(0,0,1) = 1, \ldots & f(0,0,1) = 1, f(0,0,1) = 0 \\
\vdots & \vdots \\
f(1,1,0) = 0, f(1,1,0) = 1, \ldots & f(1,1,0) = 0, f(1,1,0) = 1 \\
f(1,1,1) = 0, f(1,1,1) = 1, \ldots & f(1,1,1) = 1, f(1,1,1) = 0
\end{array}$$

Tecnológico de Monterrey

et cetera

IBM

Problem statement

Remember: the task is to determine if f is **constant** or **balanced**.

Motivation

The Deutsch-Jozsa problem is specifically designed to be easy for a quantum algorithm and hard for any deterministic classical algorithm.

Motivation

The Deutsch-Jozsa problem is specifically designed to be easy for a quantum algorithm and hard for any deterministic classical algorithm.
The motivation is to show a black box problem that can be solved efficiently by a quantum computer with no error, whereas a deterministic classical computer would need a large number of queries to the black box to solve the problem.

Classical solution

For a conventional deterministic algorithm where n is the number of bits, $2^{n-1} + 1$ evaluations of f will be required in the worst case.

Classical solution

For a conventional deterministic algorithm where n is the number of bits, $2^{n-1}+1$ evaluations of f will be required in the worst case.
To prove that f is **constant**, just over half the set of inputs must be evaluated and their outputs found to be identical (remembering that the function is *guaranteed* to be either balanced or constant, not somewhere in between).

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$f(0,0) \quad = \quad 0$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \end{aligned}$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 0 \end{aligned}$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 0 \\ f(1,1) &= 0 \end{aligned}$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 0 \\ f(1,1) &= 0 \end{aligned}$$

we knew, from the third step, that f is a **constant** function, since we were *promised* that f is either balanced or constant.

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$f(0,0) \quad = \quad 1$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \end{aligned}$$

Tecnológico de Monterrey

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \\ f(1,0) &= 1 \end{aligned}$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \\ f(1,0) &= 1 \\ f(1,1) &= 1 \end{aligned}$$

Classical solution

So, for $n = 2$, we need $2^{2-1} + 1 = 2 + 1 = 3$ evaluations of f to prove wheter or not f is constant. For example:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \\ f(1,0) &= 1 \\ f(1,1) &= 1 \end{aligned}$$

we knew, from the third step, that f is a **constant** function, since we were *promised* that f is either balanced or constant.

Classical solution

The best case occurs where the function is balanced and the first two output values that happen to be selected are different. For example:

$$f(0,0) \quad = \quad 0$$

Classical solution

The best case occurs where the function is balanced and the first two output values that happen to be selected are different. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 1 \end{aligned}$$

Classical solution

The best case occurs where the function is balanced and the first two output values that happen to be selected are different. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 1 \\ f(1,0) &= 0 \end{aligned}$$

Classical solution

The best case occurs where the function is balanced and the first two output values that happen to be selected are different. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 1 \\ f(1,0) &= 0 \\ f(1,1) &= 1 \end{aligned}$$

Classical solution

The best case occurs where the function is balanced and the first two output values that happen to be selected are different. For example:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 1 \\ f(1,0) &= 0 \\ f(1,1) &= 1 \end{aligned}$$

we knew, from the *second* step, that f is a **balanced** function, since we were *promised* that f is either balanced or constant.

Classical solution

The same applies to the other f:

$$f(0,0) \quad = \quad 1$$

Classical solution

The same applies to the other f:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 0 \end{aligned}$$

Classical solution

The same applies to the other f:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \end{aligned}$$

Classical solution

The same applies to the other f:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \\ f(1,1) &= 0 \end{aligned}$$

Classical solution

The same applies to the other f:

$$
\begin{aligned}
f(0,0) &= 1 \\
f(0,1) &= 0 \\
f(1,0) &= 1 \\
f(1,1) &= 0
\end{aligned}
$$

we knew, from the *second* step, that f is a **balanced** function, since we were *promised* that f is either balanced or constant.

Classical solution

Another example, f:

$$f(0,0) \quad = \quad 0$$

Classical solution

Another example, f:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \end{aligned}$$

Classical solution

Another example, f:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \end{aligned}$$

Classical solution

Another example, f:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \\ f(1,1) &= 1 \end{aligned}$$

Classical solution

Another example, f:

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \\ f(1,1) &= 1 \end{aligned}$$

we found out until the *third* step, that f is a **balanced** function, since we were *promised* that f is either balanced or constant.

Classical solution

One final example, f:

$$f(0,0) \quad = \quad 1$$

Classical solution

One final example, f:

$$
\begin{aligned}
f(0,0) &= 1 \\
f(0,1) &= 1
\end{aligned}
$$

Classical solution

One final example, f:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \\ f(1,0) &= 0 \end{aligned}$$

Classical solution

One final example, f:

$$\begin{aligned} f(0,0) &= 1 \\ f(0,1) &= 1 \\ f(1,0) &= 0 \\ f(1,1) &= 0 \end{aligned}$$

Classical solution

One final example, f:

$$
\begin{aligned}
f(0,0) &= 1 \\
f(0,1) &= 1 \\
f(1,0) &= 0 \\
f(1,1) &= 0
\end{aligned}
$$

we found out until the *third* step, that f is a **balanced** function, since we were *promised* that f is either balanced or constant.

Classical solution

For a conventional **randomized** algorithm, a constant k evaluations of the function suffices to produce the correct answer with a high probability (failing with a probability $\epsilon \leq 1/2^{k-1}$). However, $k = 2^{n-1} + 1$ evaluations are still required if we want an answer that is *always* correct.

Quantum Algorithm

The oracle computing $f(x)$ from x has to be a quantum oracle which doesn't decohere x. It also mustn't leave any copy of x lying around at the end of the oracle call.

Quantum Algorithm

The oracle computing $f(x)$ from x has to be a quantum oracle which doesn't decohere x. It also mustn't leave any copy of x lying around at the end of the oracle call.
The algorithm begins with the $n+1$ bit state $|0\rangle^{\otimes n}|1\rangle$. That is, the first n bits are each in the state $|0\rangle$ and the final bit is $|1\rangle$. A Hadamard transform is applied to each bit to obtain the state

$$
\begin{aligned}
H^{\oplus n} H(|0\rangle^{\oplus n}|1\rangle &= (H^{\oplus n}|0\rangle^{\oplus n})(H|1\rangle) \\
&= \left(\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\right)\left(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right) \\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|0\rangle-|1\rangle)
\end{aligned}
$$

Quantum Algorithm

Let's try this procedure with $n = 2$ and a function f that is balanced of the form

$$\begin{aligned} f(0,0) &= 0 \\ f(0,1) &= 0 \\ f(1,0) &= 1 \\ f(1,1) &= 1 \end{aligned}$$

Quantum Algorithm

$$
\begin{aligned}
H^{\oplus 2} H (|0\rangle^{\oplus 2} |1\rangle &= (H^{\oplus 2} |0\rangle^{\oplus 2})(H |1\rangle) \\
&= \left(\frac{1}{\sqrt{2^2}} \sum_{x=0}^{2^2-1} |x\rangle \right) \left(\frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right) \\
&= \frac{1}{\sqrt{8}} \sum_{x=0}^{3} |x\rangle (|0\rangle - |1\rangle)
\end{aligned}
$$

Quantum Algorithm

$$H^{\oplus 2} H (|0\rangle^{\oplus 2} |1\rangle = \frac{1}{\sqrt{8}} (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes (|0\rangle - |1\rangle)$$

$$= \frac{1}{\sqrt{8}} (|000\rangle + |010\rangle + |100\rangle + |110\rangle - |000\rangle - |011\rangle - |101\rangle - |111\rangle)$$

Quantum Algorithm

We have the function f implemented as a quantum oracle. The oracle maps the state $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$, where $\oplus$ is addition modulo 2 (XOR function).

Quantum Algorithm

We have the function f implemented as a quantum oracle. The oracle maps the state $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$, where $\oplus$ is addition modulo 2 (XOR function).
Applying the quantum oracle gives

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle \left(|f(x)\rangle - |1 \oplus f(x)\rangle\right)$$

Quantum Algorithm

In our example, $n = 2$

$$\frac{1}{\sqrt{2^3}} \sum_{x=0}^{3} |x\rangle \left(|f(x)\rangle - |1 \oplus f(x)\rangle\right) =$$

thus,

$$\begin{aligned} &= \frac{1}{\sqrt{8}}(|00\rangle \otimes (|f(0,0)\rangle - |1 \oplus f(0,0)\rangle) \\ &+ |01\rangle \otimes (|f(0,1)\rangle - |1 \oplus f(0,1)\rangle) \\ &+ |10\rangle \otimes (|f(1,0)\rangle - |1 \oplus f(1,0)\rangle) \\ &+ |11\rangle \otimes (|f(1,1)\rangle - |1 \oplus f(1,1)\rangle)) \end{aligned}$$

Tecnológico de Monterrey

Quantum Algorithm

And, according to our definition of f:

$$
\begin{aligned}
&= \frac{1}{\sqrt{8}}(|00\rangle \otimes (|0\rangle - |1 \oplus 0\rangle) \\
&+ |01\rangle \otimes (|0\rangle - |1 \oplus 0\rangle) \\
&+ |10\rangle \otimes (|1\rangle - |1 \oplus 1\rangle) \\
&+ |11\rangle \otimes (|1\rangle - |1 \oplus 1\rangle))
\end{aligned}
$$

Quantum Algorithm

Finally, we apply the XOR function:

$$= \frac{1}{\sqrt{8}}(|00\rangle \otimes (|0\rangle - |1\rangle)$$
$$+ |01\rangle \otimes (|0\rangle - |1\rangle)$$
$$+ |10\rangle \otimes (|1\rangle - |0\rangle)$$
$$+ |11\rangle \otimes (|1\rangle - |0\rangle))$$

$$= \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |101\rangle - |100\rangle + |111\rangle - |110\rangle)$$

Quantum Algorithm

For each x, $f(x)$ is either 0 or 1. Testing these two possibilities, we see the above state is equal to

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\left|x\right\rangle\left(\left|0\right\rangle-\left|1\right\rangle\right)$$

Quantum Algorithm

For each x, $f(x)$ is either 0 or 1. Testing these two possibilities, we see the above state is equal to

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle \left(|0\rangle - |1\rangle\right)$$

At this point the last qubit may be ignored.

Quantum Algorithm

In our case, ignoring the last qubit:

$$\frac{1}{2}\sum_{x=0}^{3}(-1)^{f(x)}|x\rangle|0\rangle = \frac{1}{2}((-1)^{0}|00\rangle|0\rangle + \\ + (-1)^{0}|01\rangle|0\rangle + \\ + (-1)^{1}|10\rangle|0\rangle \\ + (-1)^{1}|11\rangle|0\rangle)$$

$$= \frac{1}{2}(|000\rangle + |010\rangle - |100\rangle - |110\rangle)$$

Quantum Algorithm

We apply a Hadamard transform to each qubit to obtain

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\left[\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle\right]=\frac{1}{2^n}\sum_{y=0}^{2^n-1}\left[\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right]|y\rangle$$

Quantum Algorithm

We apply a Hadamard transform to each qubit to obtain

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\left[\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}\left|y\right\rangle\right]=\frac{1}{2^n}\sum_{y=0}^{2^n-1}\left[\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right]\left|y\right\rangle$$

where $x \cdot y = x_0 y_0 \oplus x_1 y_1 \oplus \cdots \oplus x_{n-1} y_{n-1}$ is the sum of the bitwise product.

Quantum Algorithm

In the case of our example:

$$\frac{1}{4}\sum_{x=0}^{3}(-1)^{f(x)}\left[\sum_{y=0}^{3}(-1)^{x\cdot y}\left|y\right\rangle\right] = \frac{1}{4}\sum_{y=0}^{3}\left[\sum_{x=0}^{3}(-1)^{f(x)}(-1)^{x\cdot y}\right]\left|y\right\rangle$$

Quantum Algorithm

Finally we measure the probability of measuring $|0\rangle^{\otimes n}$,

$$\left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2$$

which evaluates to 1 if $f(x)$ is constant (constructive interference) and 0 if $f(x)$ is balanced (destructive interference).