The Deutsch Jozsa Algorithm

Input: A function *f* on Σ*n* → Σ which is either balanced or constant.

[Constant: *f*(*x*) = 0 for all values of *x* OR *f*(*x*) = 1 for all values of *x*;

Balanced: *f*(*x*) = 0 for half of the inputs and *f*(*x*) = 1 for the other half of the inputs]

Output: “1” if *f* is constant, “0” if it is balanced using a single query to an oracle that computes U*f* →

The Circuit:

Briefly With Details

|  |  |
| --- | --- |
|  |  |

The steps are as follows:

=

Each of the qubits in is acted upon by a Hadamard gate,

=

= + )*n* ⊗ ( –

= − ( –

Now we apply the unitary operation *Uf* to

= − *Uf* ( – )

After applying *Uf*,

= − –

Since – = (–1)*f*(*x*) ( – , (as studied in the class), therefore

= − (–1)*f*(*x*) ( –

At this stage we apply a Hadamard to where *x* represents all possible strings over Σ*n.*

Since

*H* ( = + (–1)0 ) / ,

*H* ( = + (–1)1 ) / ,

and therefore,

*H*  = ( (–1)0.*x* + (–1)1.*x* ) /

*H* = (–1)*xz*

However, consider the situation when *x* is an *n* bit string, i.e., *x* has *n* bits, i.e., *x*1, *x*2, …, *xn*.

In that case,

*H* = *H H … H*

= (–1)*x*1*z*1 (–1)*x*2*z*2 … (–1)*xnzn*

Plugging in this expression in place of in , we get,

= (–1)*f*(*x*) (–1)*x*1*z*1  (–1)*x*2*z*2 … (–1)*xnzn* ( –

= − (–1)*f*(*x*) (–1)*x*1*z*1(–1)*x*2*z*2 ... (–1)*xnzn…*

**=** −  **(–1)*f*(*x*) + *x*1*z*1 +*x*2*z*2 +*xnzn***( – ***where***= *…*.

**We can discard the lower qubit** ( –

and measure the top qubit(s) only which are:

−  **(–1)*f*(*x*) + *x*1*z*1 +*x*2*z*2 +*xnzn.***

**What is the probability of measuring the state , i.e, *z*1 = *z*2 = … = *zn* = 0?**

**It is given by:**

***P*(Measuring all 0s) = ( (–1)*f*(*x*))2**

Observe that if *f*(*x*) is constant, then either *f*(*x*) = 0 for all values of *x*, leading to

*P* =  **( (–1)0**+ **(–1)0  + … (–1)0 )2 = 2*n*/ 2*n*= 1.**

Or *f*(*x*) = 1 for all values of *x* leading to,

*P* =  **( (–1)1**+ **(–1)1  + … (–1)1 )2 = 2*n*/ 2*n*= 1.**

**So if *f*(*x*) is constant, there is a 100% probability of measuring all 0s the first *n* qubits.**

**What happens if *f*(*x*) is balanced.**

**In that case, whenever *f*(*x*) = 0, (–1)*f(x)* = 1, and**

**Whenever *f*(*x*) = 1, (–1)*f(x)* = –1 ,**

**Therefore, the probability is,**

***P* = ((–1)0**+ **(–1)1  + … (–1)0) (this is –1 for half of the inputs and +1 for the other half)**

**= 0/ 2*n*= 0.**

**Hence if *f*(*x*) is balanced, there is no chance of measuring all 0s.**

**This completes the analysis of the algorithm.**

The Bernstein-Vazirani Algorithm

To get an idea of the Bernstein-Vazirani Algorithm, we will work out an example here.

Although the Bernstein-Vazirani Problem is somewhat different as compared to Deutsch-Jozsa Problem, the solution is almost similar.

Consider a two-bit string *u* = \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Choose any string that you would like).

Now create a function on a two-bit string i.e., *x* = *x*1 *x*2 such that

*f*(*x*) = *u* ⋅ *x* = *u*1 *x*1 ⊕ *u*2 *x*2

Example: Suppose *u* = 10.

Then the following function: *f*(00) = 0, *f*(01) = 0, *f*(10) = 1, *f*(11) = 1 satisfies the above condition.

Note that not all functions on two bits are of this type (i.e. there exists a *u* that satisfies the dot-product condition).

The Bernstein Vazirani Problem states that:

[Promise] Suppose we are given a function on Σ*n* → Σ which can be expressed as a dot product of a string *u* and the input *x*. Can you find *u*, and if yes, using how many queries to *f*?

Write, discuss and think the Classical Solution here:

|  |
| --- |
|  |
|  |
|  |
|  |

Quantum Solution: It turns out, that the quantum solution requires only 1 query to the oracle and is surprisingly similar to the Deutsch Jozsa Algorithm. It is as follows:

1. Start with the state:
2. Apply
3. Now apply the oracle to this state to get (–1)*f*(*x*) ( –
4. Now apply Hadamard to the top *n* qubits to get (–1)*f*(*x*) + *x*1*z*1 +*x*2*z*2 +*xnzn*
5. Finally measure the top *n* qubits.
6. The result of the measurement is your answer *u*.
7. Try it now for the above two bit function!

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |