CS 599 P1: Introduction to Quantum Computation — Boston University, Fall 2025
Instructor: Alexander Poremba — **Scribe:** David Ou

LECTURE #8: BLACK-BOX ORACLES & DEUTSCH-JOZSA ALGORITHM

Last time, we saw that quantum correlations are much more powerful than classical correlations. In this lecture, we continue along those lines and explore whether quantum computers can solve certain—possibly even contrived—problems faster than classical computers. We set up the discussion by framing problems as Boolean decision tasks and introducing the *oracle model*, where algorithms only have black-box access to a function. After reviewing how to implement such functions reversibly on a quantum computer, we see the first examples of quantum speedup: **Deutsch's problem**, where one quantum query replaces two classical ones, and the **Deutsch–Jozsa algorithm**, which extends this idea to show an exponential separation between quantum and deterministic classical algorithms. This marks the beginning of our study of quantum algorithms built on superposition, interference, and oracle queries.

1 Computational Problems

The central motivating question that we are going to be focusing is the following:

Motivating Question

Can a quantum computer solve *certain problems* much faster than a classical computer?

To tackle this question, we first have to ask: what exactly is ... a *problem*?
Several lectures ago, we encountered the notion of a *Boolean function*; this is a mapping

$$f : \{0,1\}^n \ \rightarrow \ \{0,1\}^m,$$

which transforms an n-bit string into an m-bit output. It turns out that Boolean functions provide natural framework for describing computational problems. For example, when $m = 1$, they naturally capture *decision problems*, i.e. problems with a simple "yes" or "no" answer. Examples include:

1. **Constraint satisfaction problems**: Here, a canonical example is the *Boolean satisfiability (SAT)* problem. Given a Boolean formula in which the constraints are represented by *clauses*; for example,

   $$\Phi(x_1, x_2, x_3, x_4) = (x_1 \vee \neg x_2 \vee x_3) \ \wedge \ (x_2 \vee x_3 \vee x_4) \ \wedge \ (x_1 \vee x_3 \vee \neg x_4),$$

   we can define the Boolean function f (taking as input a binary description of Φ) with

   $$f_{\text{SAT}}(\Phi) = \begin{cases} 1 & \text{if } \Phi \text{ is } \textit{satisfiable}, \\ 0 & \text{otherwise.} \end{cases}$$

   In other words, the task is to decide whether there exists an assignment of the variables x_1, x_2, x_3, x_4 that makes $\Phi(x_1, x_2, x_3, x_4)$ evaluate to 1.

2. **Graph problems**: Consider the *graph k-coloring* problem. Given a graph $G = (V, E)$ and an integer k, we want to know whether the vertices of G can be assigned colors from $\{1, \ldots, k\}$ such that no edge has both endpoints of the same color. This can be expressed as a Boolean function f over the binary encoding of (G, k):

$$f_{\text{col}}(G, k) = \begin{cases} 1 & \text{if } G \text{ is } k\text{-colorable,} \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, other graph decision problems (such as whether a graph has a cut of size at least K) can be captured in the same Boolean framework.

3. **Number-theoretic problems**: As an example, take *primality testing*. For an integer N, say which is encoded in binary, we can define the function

$$f_{\text{prime}}(N) = \begin{cases} 1 & \text{if } N \text{ is prime,} \\ 0 & \text{otherwise.} \end{cases}$$

Another example is the *factoring decision problem*. Given an integer N and a bound K, both of which are encoded in binary, we can define the function

$$f_{\text{factor}}(N, K) = \begin{cases} 1 & \text{if } N \text{ has a nontrivial factor} \leq K, \\ 0 & \text{otherwise.} \end{cases}$$

Both of these illustrate how number-theoretic decision problems can be expressed naturally as Boolean functions on binary-encoded integers.

These examples illustrate how Boolean functions serve as a foundation for modeling a wide range of computational problems. With this setup, we now ask: *if a computational problem is specified by some Boolean function f, can a quantum computer solve it more efficiently than a classical one?*

But how exactly do we even execute such a function on a quantum computer?

Quantum oracles for Boolean functions. Given a Boolean function

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

the natural question is: *how do we even compute $f(x)$ on a quantum computer?* Since quantum evolution must be unitary (and hence reversible), we need a method that preserves reversibility even when f itself is not. In Lecture 5, we saw how to get around the issue of reversibility. The key idea is to embed f into a reversible transformation U_f by carrying the input alongside the output:

1. **Initialization:** Prepare $n + m$ qubits,

$$|x_1, \ldots, x_n\rangle \otimes |0, \ldots, 0\rangle,$$

where the first n qubits encode the input and the remaining m qubits serve as an output register.

2. **Unitary evaluation:** Define the unitary U_f for the function f via

$$U_f : |x\rangle|y\rangle \ \mapsto \ |x\rangle|y \oplus f(x)\rangle,$$

for all $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ and $y = (y_1, \ldots, y_m) \in \{0,1\}^m$. In particular,

$$U_f\Big(|x_1, \ldots, x_n\rangle\,|0, \ldots, 0\rangle\Big) = |x_1, \ldots, x_n\rangle\,|f(x_1, \ldots, x_n)\rangle.$$

This construction ensures reversibility: applying U_f twice restores the original state.

2 Computational Problems with Quantum Speed-Ups?

Not surprisingly, not every task which is specified by a Boolean function f leads to a problem with meaningful quantum advantage. Ideally, f should describe a problem with structure that a quantum computer can exploit, and which a classical computer cannot. The key challenge is to identify such problems.

2.1 Black-Box Oracles

In this lecture, we consider the *black-box oracle model*. Here, the function f is hidden inside a "black box" whose inner workings are unavailable to us. These models are extremely insightful in theoretical computer science; particularly in complexity theory and cryptography.

Definition: Black-Box Oracle Problem

A black-box oracle problem is specified by a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$. Any algorithm seeking to solve the problem does not receive a description of f in the clear; instead, the algorithm merely gets access to the input/output behavior of the function:

In other words, an algorithm may submit "queries" to the function; each time a query to x is submitted, the algorithm learns the output $f(x)$ but nothing else.

In the quantum case, the black box oracle corresponds to a reversible unitary implementation of f; in other words, the oracle expects an $n + m$ qubit input and evaluates the unitary

$$U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle.$$

In this setup, the oracle serves as a thought experiment: by restricting us to black-box access, we can cleanly compare the number of queries. Perhaps a quantum computer can solve **black-box oracle** problems (specified by some Boolean function) "much faster" than a classical computer?

3 Deutsch Algorithm

In 1985, David Deutsch introduced the first quantum algorithm that demonstrated a provable speed-up over classical computation. Although the advantage is modest—a factor of $2\times$—the result is nevertheless quite significant, as it marks the very first explicit example of quantum advantage in the query (oracle) model. While the problem itself is quite simple, but it illustrates in a clear and tangible way how quantum interference can be harnessed to reduce the number of queries required to solve the problem.

3.1 Deutsch's Problem

David Deutsch proposed the following computational task:

Deutsch's Problem

We are given black-box oracle access to a Boolean function

$$f : \{0,1\} \longrightarrow \{0,1\}.$$

and the task is to decide whether the function is *constant* on its two inputs or *distinct*, i.e.,

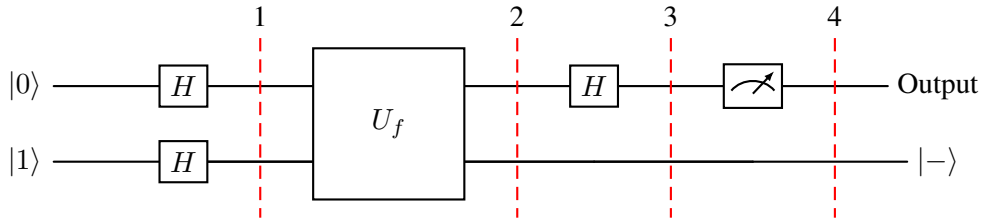
$$f(0) = f(1) \quad \text{or} \quad f(0) \neq f(1).$$

Let us now distinguish between the classical and quantum *query* complexity—the number of calls to f which are necessary to solve the problem.

- **Classical case:** Any deterministic classical algorithm clearly must query the oracle twice—once for $f(0)$ and once for $f(1)$. With only a single query, the two cases cannot be distinguished.
- **Quantum case:** A quantum algorithm, by contrast, can solve the same problem with just *one* query. The key idea is to prepare a "superposition of inputs", apply the oracle only once, and then use interference via a Hadamard gate so that a measurement of the first qubit reveals the answer deterministically.

3.2 Deutsch's algorithm

The quantum circuit for Deutsch's algorithm is shown below. Two qubits are initialized, Hadamard gates create superpositions, the oracle U_f is applied once, and finally another Hadamard plus measurement extracts the answer from the first qubit:



3.3 State Evolution

We now follow the computation step by step.

1. **Initialization and Hadamards:** Starting from $|0\rangle\,|1\rangle$, Hadamards create a superposition:

$$\begin{aligned} |\psi_1\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\ &= \frac{1}{2}\Big(|0\rangle\,(|0\rangle - |1\rangle) + |1\rangle\,(|0\rangle - |1\rangle)\Big). \end{aligned}$$

   The first qubit is now in an equal superposition of $|0\rangle$ and $|1\rangle$, while the second is in the $|-\rangle$ state.

2. **Oracle Action:** The oracle U_f acts as $U_f|x, y\rangle = |x,\, y \oplus f(x)\rangle$. Applying it term by term:

$$\begin{aligned} U_f|\psi_1\rangle &= \tfrac{1}{2}\big[|0\rangle\,\big(\,|f(0)\rangle - |1\oplus f(0)\rangle\,\big) + |1\rangle\,\big(\,|f(1)\rangle - |1\oplus f(1)\rangle\,\big)\big] \\ &= \tfrac{1}{2}\Big[\,(-1)^{f(0)}\,|0\rangle\,(|0\rangle - |1\rangle) + (-1)^{f(1)}\,|1\rangle\,(|0\rangle - |1\rangle)\Big] \\ &= (-1)^{f(0)}\tfrac{1}{\sqrt{2}}\Big[|0\rangle + (-1)^{\,f(0)\oplus f(1)}\,|1\rangle\Big]\otimes\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &\equiv_{\substack{\text{global}\\ \text{phase}}} \tfrac{1}{\sqrt{2}}\Big[|0\rangle + (-1)^{\,f(0)\oplus f(1)}\,|1\rangle\Big]\otimes|-\rangle\,. \end{aligned}$$

The second qubit remains in the fixed state $|-\rangle$, and can therefore be ignored. All relevant information is encoded in the first qubit.

3. **Final Hadamard:** Apply a Hadamard to the first qubit:

$$|\psi_2\rangle = \tfrac{1}{\sqrt{2}}\Big(|0\rangle + (-1)^{\,f(0)\oplus f(1)}\,|1\rangle\Big)\,.$$

Using $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$, we can write

$$\begin{aligned} H|\psi_2\rangle &= \tfrac{1}{\sqrt{2}}\Big(H|0\rangle + (-1)^{\,f(0)\oplus f(1)}H|1\rangle\Big) \\ &= \tfrac{1}{\sqrt{2}}\Big(\tfrac{|0\rangle+|1\rangle}{\sqrt{2}} + (-1)^{\,f(0)\oplus f(1)}\tfrac{|0\rangle-|1\rangle}{\sqrt{2}}\Big) \\ &= \tfrac{1}{2}\Big[(1+(-1)^{\,f(0)\oplus f(1)})\,|0\rangle + (1-(-1)^{\,f(0)\oplus f(1)})\,|1\rangle\Big]. \end{aligned}$$

Therefore, the final state becomes

$$H|\psi_2\rangle = \begin{cases} |0\rangle\,, & \text{if } f(0) = f(1), \\ |1\rangle\,, & \text{if } f(0) \neq f(1). \end{cases}$$

4. **Measurement:** Finally, measure the first qubit. The outcome directly reveals the property of f:

$$\Pr\left[\boxed{\nearrow} = |0\rangle\right] = 1 \quad \text{if } f(0) = f(1),$$

$$\Pr\left[\boxed{\nearrow} = |1\rangle\right] = 1 \quad \text{if } f(0) \neq f(1).$$

Thus, the measurement deterministically distinguishes the two cases after a *single* oracle query.

4 Deutsch–Jozsa Algorithm

The Deutsch–Jozsa algorithm, introduced in 1992, is one of the first demonstrations of an *exponential speedup* of quantum computation over deterministic classical computation in the query (oracle) model.

4.1 Deutsch-Josza Problem

Deutsch and Jozsa proposed the following black-box oracle problem.

Deutsch-Josza Problem

Given oracle access to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with the promise that f is either

- **constant**, i.e., $f(x)$ has the same output for all x and either $f(x) \equiv 0$ or $f(x) \equiv 1$; or it is
- **balanced**, i.e., exactly half of all inputs give $f(x) = 0$ and the other half give $f(x) = 1$,

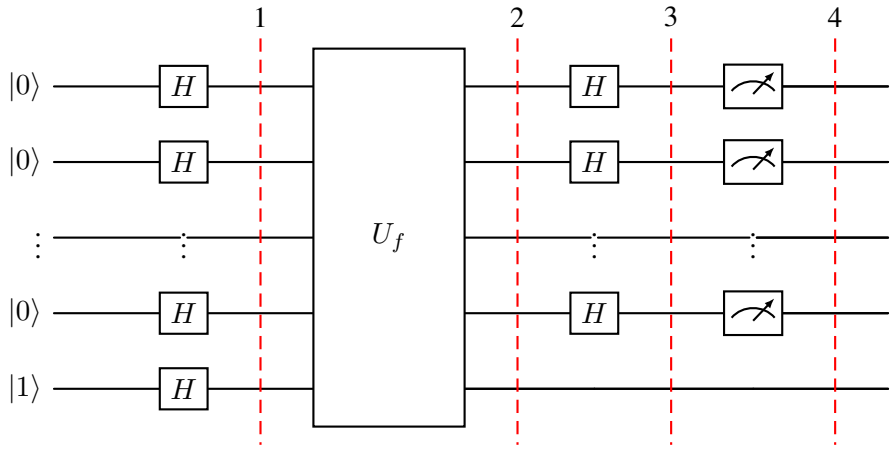
decide, *with certainty*, which is the case.

Let us now distinguish between the classical and quantum *query* complexity—the number of calls to f which are necessary to solve the problem.

- **Classical:** If we are restricted to a deterministic classical algorithm, we may have to check more than half of the 2^n possible inputs to be sure: in the worst case, this requires $2^{n-1} + 1$ queries.
- **Quantum:** The Deutsch–Jozsa algorithm solves the problem with only a *single* oracle call.

4.2 Deutsch–Jozsa algorithm

The circuit uses $n + 1$ qubits: n for the input register and one ancilla qubit prepared in the state $|1\rangle$, which is then transformed into $|-\rangle$ by a Hadamard gate.



The high-level idea is:

1. Prepare a uniform superposition of all possible inputs.
2. Use the oracle U_f to encode $f(x)$ in the *phase* of each branch via *phase kickback*.
3. Apply Hadamards again to enforce *interference*.
4. Measure: a single readout reveals whether f is constant or balanced.

4.3 Step-by-Step Evolution

1. **Initialization and Hadamards.**

   Review of the Hadamard gate. For a single qubit $|x\rangle$ with $x \in \{0, 1\}$,

   $$H|x\rangle = \tfrac{1}{\sqrt{2}}\big(|0\rangle + (-1)^x|1\rangle\big), \qquad H|0\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle),\ H|1\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

   Extension to n qubits.

   $$H^{\otimes n}|x\rangle = \tfrac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{\langle x,y\rangle}|y\rangle,$$

   where

   $$\langle x, y\rangle := x_1y_1 \ \oplus\ x_2y_2 \ \oplus \cdots \oplus\ x_ny_n,$$

   with $\oplus$ denoting addition modulo 2.

   Initialization.

   $$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle.$$

   After Hadamards. Applying $H^{\otimes n}$ to the input and H to the ancilla yields

   $$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x\in\{0,1\}^n} |x\rangle\big(|0\rangle - |1\rangle\big).$$

2. **Oracle action (phase kickback).**

   The oracle acts as $U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$. Applying it to $|\psi_1\rangle$,

   $$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x\in\{0,1\}^n} |x\rangle\big(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle\big).$$

   For each x,

   $$|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle = (-1)^{f(x)}(|0\rangle - |1\rangle),$$

   hence

   $$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle\ (|0\rangle - |1\rangle).$$

   Using $(|0\rangle - |1\rangle) = \sqrt{2}\,|-\rangle$, we can rewrite

   $$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle|-\rangle.$$

   Since the ancilla stays fixed, we may factor it out:

   $$|\psi_2\rangle \ \equiv\ \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle.$$

3. **Final Hadamards.**

   Apply $H^{\otimes n}$ on the input register:

$$H^{\otimes n}|x\rangle = \tfrac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{\langle x,y\rangle} |y\rangle .$$

   Therefore,

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y\in\{0,1\}^n} \left(\sum_{x\in\{0,1\}^n} (-1)^{f(x)} (-1)^{\langle x,y\rangle} \right) |y\rangle .$$

4. **Measurement and decision rule.**

   Let Y denote the random variable for the final measurement outcome at the end of the circuit. Then, the probability of observing the outcome y is given by

$$\Pr[Y = y] = \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} (-1)^{\langle x,y\rangle} \right|^2 .$$

   In particular,

$$\Pr[Y = 0^n] = \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} \right|^2 = \begin{cases} 1, & f \text{ constant}, \\ 0, & f \text{ balanced}. \end{cases}$$

   Algorithm: measure the input register once; output "constant" iff $Y = 0^n$, else "balanced."

4.4 Intuitive Picture

Why does this work? The trick is that the oracle call writes $f(x)$ not into a register we can directly read, but into the *phase* of the superposition. This is invisible locally, but becomes strikingly visible after the Hadamard transform.

- If f is constant, every branch of the superposition carries the same phase. All amplitudes line up and interfere *constructively* at $|0^n\rangle$, so the measurement is guaranteed to yield 0^n.
- If f is balanced, half the branches get a $+1$ phase and half a -1 phase. These contributions cancel *exactly* at $|0^n\rangle$, forcing the measurement outcome to be something else.

This is the essence of the Deutsch–Jozsa speedup: a single global check (via interference) substitutes for exponentially many classical queries.

Takeaway

The Deutsch–Jozsa algorithm highlights the two hallmark resources of quantum computing:

1. **Superposition**: one query explores all inputs in parallel.
2. **Interference**: amplitudes recombine to eliminate wrong answers and amplify the right one.

Even though the problem is artificial, the method—encoding information into global phases and extracting it with interference—is a template that recurs in many powerful quantum algorithms.