Quantum Computing

2. Algorithms

Frank C Langbein

[1]:
```python
from qutip import *
from qutip.qip.operations import *
import numpy as np
```

Deutsch-Jozsa Problem

- Let $f(x)$ be a function mapping n-bit strings x to $\{0,1\}$
  - f is either a **constant** function (taking the same value $c \in \{0,1\}$ for all inputs) or
  - f is a **balanced** function (taking each value 0, 1 for exactly half of the inputs)
- Determine if f is **constant** or **balanced** by making as few calls to f as possible
- Classical algorithm
  - In the worst case, $2^{n-1}+1$ evaluations of f needed (test one more than half of the inputs)
- Quantum algorithm
  - Only one evaluation of f is required
- First algorithm showing separation between quantum and classical complexity
  - Crucially, it uses the phase information

Oracles

- A quantum representation of f is required
  - Note, a reversible classical representation of any Boolean funciton f is

$$[x_1, \ldots, x_n, y] \mapsto [x_1, \ldots, x_n, y \oplus f(x_1, \ldots, x_n)]$$

    where $\oplus$ is the addition modulo 2.
  - We already had this for CCNOT (Toffoli gate)

$$[x_1, x_2, y] \mapsto [x_1, x_2, y \oplus \mathrm{AND}(x_1, x_2)]$$

    as reversible AND gate
- Note, this means a quantum computer can calculate everything a classical computer can
  - There is a reversible version of any Boolean function
  - Represent input x of f as n qubits $|x\rangle$, encoding the bits in x as $|0\rangle$, $|1\rangle$ instead
  - Add an output qubit $|y\rangle$, representing the result of $f(x)$
- We still need the unitary operation U_f for $f(x)$
  - This is referred to as **oracle**
  - "all powerful", telling us the functIon value on any given input (e.g. whether it is correct or not)

Bit Oracles

- Create unitary operator U_f such that

$$U_f|x_n,\ldots,x_1,y\rangle = |x_n,\ldots,x_1,y\oplus f(x_1,\ldots,x_n)\rangle$$

- For a one-bit $f(x_1)=x_1$:
$$U_{\text{const}}|x_1,y\rangle = |x_1,y\oplus x_1\rangle$$
  - This is a CNOT gate (control is x_1 and target is y)
- For three-bit $f(x_1,x_2,x_3)=(x_1 \text{ AND } x_2) \text{ OR } x_3$

$$U_f|x_3,x_2,x_1,y\rangle = |x_3,x_2,x_1,y\oplus f(x_1,x_2,x_3)\rangle$$

  - circuit

Sign Oracles

- We can apply the oracle to general quantum states, not just the Z-basis
  - This enables us to ask quantum questions, not just classical questions
- What if we replace $|y\rangle$ on the first qubit with $|-\rangle$?

$$U_f(|x_n,\ldots,x_1\rangle\otimes|-\rangle) = U_f\left(\frac{1}{\sqrt{2}}|x_n,\ldots,x_1,0\rangle - \frac{1}{\sqrt{2}}|x_n,\ldots,x_1,1\rangle\right)$$

$$= \frac{1}{\sqrt{2}}\left(|x_n,\ldots,x_1,f(x_1,\ldots,x_n)\rangle - |x_n,\ldots,x_1,1\oplus f(x_1,\ldots,x_n)\rangle\right)$$

$$= |x_n,\ldots,x_1\rangle\otimes\frac{1}{\sqrt{2}}\left(|f(x_1,\ldots,x_n)\rangle - |f(x_1,\ldots,x_n)\oplus 1\rangle\right)$$

$$= (-1)^{f(x_1,\ldots,x_n)}(|x_n,\ldots,x_1\rangle\otimes|-\rangle)$$

  - $|-\rangle$ remains unchanged, but we pick up an overall minus sign if $f(x_1,\ldots,x_n)=1$
- This is the same than applying the **sign oracle** O_f on the first n qubits

$$O_f|x\rangle = (-1)^{f(x)}|x\rangle$$

  - For Z-basis states, this does not do much (the overall sign is not observable)
  - For superposition states, this can introduce "relative signs"
- Practically this flips the phase for the states for which f is 1
  - Note $HXH=Z$ and $HZH=X$
    - A many-controlled NOT gate can be turned into a many-controlled phase gate and vice versa by bracketing the gate (not the controls) with H
    - Braket any controls that switch on 0 instead of 1 with X
  - Flip the phase of $|111\rangle$: circuit
  - Flip the phase of $|101\rangle$: circuit
- Balanced example: $f=1$ for even number of bits, otherwise 0 - circuit
- Constant example: $f\equiv 1$ on 3 bits - circuit

Deutsch Josza Algorithm

1. Initial state: $|0 \dots 0\rangle$
2. Apply $H^{\otimes n}$
3. Apply oracle U_f
4. Apply $H^{\otimes n}$
5. Measure all qubits:
   - If all outcomes are 0, then the function is constant
   - Otherwise it is balanced

Constant $f \equiv 1$ example

Balanced f, $f = 1$ for even number of bits, otherwise 0 example

Proof of Deutsch Josza Algorithm

- With normalisation constant C and $x_l, y_l \in \{0,1\}, l \in \{1, \dots, n\}$

$$H|x_1\rangle = C \sum_{y_1 \in \{0,1\}} (-1)^{x_1 y_1} |y_1\rangle$$

$$\begin{aligned}(H \otimes H)|x_2 x_1\rangle &= C^2 \left(\sum_{y_2 \in \{0,1\}} (-1)^{x_2 y_2} |y_2\rangle \right) \otimes \left(\sum_{y_1 \in \{0,1\}} (-1)^{x_1 y_1} |y_1\rangle \right) \\ &= C^2 \sum_{y \in \{0,1\}^2} (-1)^{x_1 y_1 + x_2 y_2} |y\rangle\end{aligned}$$

- This generalizes to

$$H^{\otimes n}|x\rangle = C^n \sum_{y \in \{0,1\}^n} (-1)^{x \odot y} |y\rangle$$

  with

$$x \odot y = \sum_l x_l y_l (\mathrm{mod}\ 2)$$

- First Hadamard transforms give the state

$$C^n \sum_{x \in \{0,1\}^n} |x\rangle$$

- O_f maps this to

$$C^n \sum_{x \in \{0,1\}} (-1)^{f(x)} |x\rangle$$

- The second Hadamard transforms then give

$$C^{2n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \left(\sum_{y \in \{0,1\}^n} (-1)^{x \odot y} |y\rangle \right) = C^{2n} \sum_{y \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x) + x \odot y} |y\rangle$$

- Thus, probability of measruing state $|0 \dots 0\rangle$ is

$$|C^{2n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}|^2 = \begin{cases} 1 & \text{if } f \text{ is constant} \\ 0 & \text{if } f \text{ is balanced} \end{cases}$$

Grover's Search Algorithm

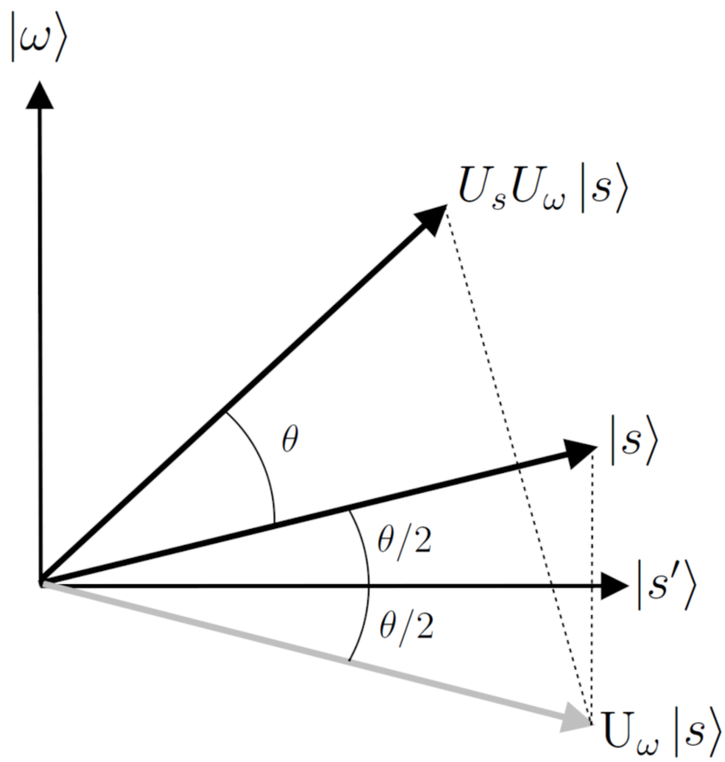
- Given a list of N items, find one with a unique property
  - $f(x) = 1$ iff x is the unique item, 0 otherwise
  - As before, we assume x is an n-bit string
    * Can use a sign oracle
- Classical algorithm
  - In the worst case, we need to look at all N items
  - Expected number of items is $N/2$ (if probability is uniform)
- Quantum algorithm
  - With Grover's algorithm we can find the item in about $\sqrt{N}$ steps
    * Quadratic speed-up
  - Hard limit for possible speedup
- Provides a quadratic speedup generically for many classical problems

Amplitude Amplification

- Core of Grover's algorithm
- Any initial guess for the unique item is fine, so start with a uniform superposition

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

  - Guessing the correct value is therefore 1 in 2^n
  - Expect to guess $N = 2^n$ times to find the item
- Amplitude amplification
  - Increase probability to guess the correct item
- $|s\rangle$ and the item $|\omega\rangle$, we wish to find, span a 2D plane in $\mathbb{C}^N$
  - Not orthogonal, as $|\omega\rangle$ is part of $|s\rangle$
  - Remove $|\omega\rangle$ from $|s\rangle$ and normalise to get $|s'\rangle$, orthogonal to $|\omega\rangle$
  - Implement a rotation to increase probabiltiy of find $|\omega\rangle$



Amplitude Amplification Procedure

0. Start with $|\Psi_0\rangle = |s\rangle = H^{\otimes n}|0\rangle^n$, $t = 0$
   - All states are equally likely, have equal amplitudes $1/\sqrt{N}$
1. Apply $U_\omega = O_f$: $|\Psi_{t'}\rangle = O_f|\Psi_t\rangle$
   - Reflection of $|\Psi_t\rangle$ about $|s'\rangle$
   - Amplitude of $|\omega\rangle$ becomes negative, meaning average amplitude is lowered
2. Apply $U_s = 2|s\rangle\langle s| - I$: $|\Psi_{t+1}\rangle = U_s|\Psi_{t'}\rangle = U_s O_f|\Psi_t\rangle$
   - Two reflections always create a rotation
   - $U_s O_f$ rotate the state closer to $|\omega\rangle$
   - Increases the amplitude of $|\omega\rangle$ and lowers all others
3. $t \leftarrow t + 1$, repeat from 1

Roughly $\sqrt{N}$ repeats suffice to get close to probability 1 for $|\omega\rangle$ * Amplitude grows approximately linear with t: $\sim t/\sqrt{N}$ * Probability is square of amplitude, i.e. $t \sim \sqrt{N}$ * Additional repeats may reduce the amplitude again!

Example 5-bit cirtcuit

- Note, instead of $H^{\otimes n}$, we can initially apply $(HX)^{\otimes n}$
  - Creates superposition state with "mixed phases"
- O_f still inverts the phase of $|\omega\rangle$
- A multi-anti-controlled not gate then provides the equivalent U_s operation

Example 5-bit circuit

Factorisation

- Factorise an integer R with d digits

- Brute force
    - Find all prime numbers p up to $\sqrt{R}$ and check whether p divides R
    - This is exponential in d
- Quadratic sieve
    - Construct a, b such that $a^2 - b^2$ is a multiple of R
    - Check whether $a \pm b$ have common factors with R
    - This is exponential in $\sqrt{d}$
- General number field sieve (GNFS) - best known classical approach
    - Exponential in $d^{1/3}$
- Shor's algorithm
    - Polynomial in d

Period Finding

- Given integers R and a
    - Find smallest positive integer p such that $a^p - 1$ is a multiple of R
    - p is called the period of a modulo R
        - The period of a modulo R is the smallest ineger p such that $a^p = 1(\text{mod } R)$
    - In general well defined if R and a are co-prime (have no common factors)
- E.g. $R = 15, a = 7$:
    - $7^2 = 4(\text{mod } 15)$
    - $7^3 = 4 * 7 = 13(\text{mod } 15)$
    - $7^4 = 13 * 7 = 1(\text{mod } 15)$
    - 7 has period 4 modulo 15

Factoring and Period Finding

- Assume we can find the period r of a modulo R, given a and R are co-prime
- For simplicity, assume $R = f_1 f_2$ has only two prime factors
- Procedure to find the prime factors of R:
    - Pick a random integer a between 2 and $R - 1$
    - $g = GCD(R, a)$ is greatest common divisor (Euclid's algorithm)
    - If we are lucky, a and R have some common prime factors, i.e. g is either f_1 or f_2:
        - We are done.
    - Compute the period p of a modulo R
    - Repeat above (with new random a) until p is even
        - A significant fraction of all integers a have even period
- If p is even:
    - Note, $u - 1 = (\sqrt{u} - 1)(\sqrt{u} + 1)$ with $u = a^p$
    - So $a^{p/2} - 1$ is not a multiple of R (otherwise the period of a modulo R would be $p/2$)
    - If $a^{p/2} + 1$ is not a multiple of R:
        - That means f_1 is a prime factor of $a^{p/2} - 1$ and f_2 is a prime factor of $a^{p/2} + 1$ (or vice versa)
        - So we can find f_1, f_2 by $GCD(R, a^{p/2} \pm 1)$
    - If, unluckily, $a^{p/2} + 1$ is a multiple of R
        - Give up and try a different a (not too common)
- On average we only have to call the period finding machine twice to factor R!
    - The core of Shor's algorithm uses a quantum computer, mainly computing the Qunatum Fourier Transform, to find the period

Quantum Fourier Transform

- QFT circuit and inverse QFT
- Inverse QFT of a periodic state
  - The output has a number of peaks equal to the period of the input
- Prepare a periodic quantum state
  - We do not see the probabilty distribution on a quantum computer, only the measurements
  - We would have to sample the output $O(\sqrt{p})$ times if there were p peaks to get a reasonable idea

Guessing the period from a sample

- Suppose we are sampling from a frequency space of size 1024
  - We do not know the peiord of the input
  - We got a frequency sample 339
  - Looks like… 3 ($1024/3 \sim 341$)
- For huge ranges of possible periods, we use the "continued fractions algorithms for the best rational approximation" (also Diophantine approximation and Pade approximant)

[2]:
```
from fractions import Fraction

def sampled_freq_to_period(sample_freq, num_freqs, max_period):
  f = Fraction(sample_freq, num_freqs).limit_denominator(max_period)
  return f.denominator

print(sampled_freq_to_period (339, 1024, 10))
```

```
3
```

- We may be unlucky and get a sample frequency close to 0, but that is unlikley
- Note, when factoring a number R with d bits, the maximum period will be 2^d
  - We need $O(\lg 2^d) = O(d)$ qubits

Preparing states with unknown periods

- So far we only produced states with modular addition
  - We knew the modulus when we created the circuit!
  - In order to do something interesting, we have to be able to make a period state producing circuit from start to finish and still not know what period its state will have
- Period of $f(x) = 2^x (\bmod 23)$

[3]:
```
print(sampled_freq_to_period(186,512,50))
```

```
11
```

- Period of $f(x) = 7^x (\bmod 58)$

[4]:
```
print(sampled_freq_to_period(878,1024,58))
```

```
7
```

Shor's Algorithm