Communication Systems Laboratory
Lab 3: Quantum Computing Algorithms

(Report Due: 21:00, November 3, 2021)

1 Overview

In the previous Lab 2, we have explored some quantum communication protocols and known how to use entanglement as a resource for technological applications.

In this lab, we are going to implement certain *quantum algorithms* via `QISKit` for some computational task. At the first glance, it seems more complicated to implement all that gates needed in this Lab. However, do not be intimidated. The goal of this lab for you is not to write down hundreds of lines of codes. Instead, each problem (i.e. the Deutsch–Jozsa algorithm, Grover's search algorithm, and the period-finding algorithm) in this lab will navigate what's going on along the computation. So that you will have a flavor or at least a feeling about how quantum computation is executed. Have fun with this lab! And note that you can always refer to the `QISKit` Textbook once you encounter any difficulties.

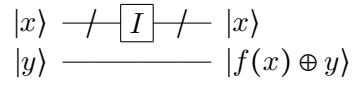
2 Experiments

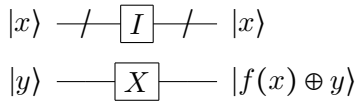
1. **(20 points) The Deutsch–Jozsa algorithm** We implement the following "balanced versus constant" problem taught in class.

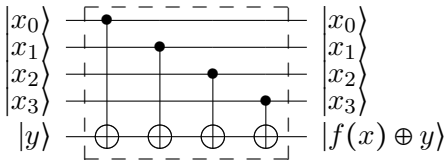
   The "Balanced versus Constant" Problem

   Given: A black box for a function $f : \{0,1\}^n \to \{0,1\}$.
   Promise: The function f is either (a) constant (i.e. $f(x) = 0$ for all x or $f(x) = 1$ for all x); or (b) a *balanced* function in the sense that $f(x) = 0$ or 1 for exactly half of the 2^n inputs x.
   Problem: Determine with certainty whether f is balanced or constant.

   (a) **(10 points)** Let us take $n = 4$. The *quantum oracle* U_f is given by $U_f : |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |f(x) \oplus y\rangle$, where $|x\rangle \equiv |x_0\rangle|x_1\rangle \cdots |x_{n-1}\rangle$ denotes the n-qubit *input register* and $|y\rangle$ is the 1-qubit *output register*. Verify that the following three circuits represents the quantum oracles for a constant, a constant, and a balanced function, respectively.







Hint. For verifying f, you may feed in computational basis states as inputs and measure the corresponding outputs to construct the truth table.

(b) **(15 points)** Using the oracles provided from above, implement the Deutsch–Jozsa algorithm to decide whether the chosen oracle was constant or balanced. (You can either use simulators or IBM's real devices.)

Remark. Problem 1a is to provide you a circuit of the quantum oracle U_f for certain classical oracle f so that you can implement the Deutsch–Jozsa algorithm. However, if you are given an f, e.g. $f(x) = 0$ for $x = 000, 010, 100, 110$ and $f(x) = 1$ otherwise, how do you come up with a circuit of U_f that only contains elementary quantum gates?

This might not be an easy task. So you may have a feeling that constructing an arbitrary quantum oracle is not trivial. Yet, we will be fortunate if we are give a classical circuit implementing f and hence we can also translate into a quantum one and also the desired U_f.

2. **(40 points) Grover's Search:** As shown in class, Grover's search algorithm is a powerful quantum algorithm which provides a square-root speedup for searching a target in an unsorted list. In this problem, we will implement an Grover's search algorithm for a list with size $N = 2^n = 8$.

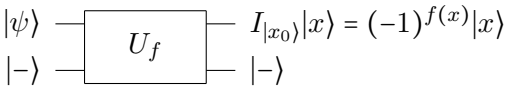
The Unstructured Search Problem

Given: A black box for a function $f : \{0,1\}^n \to \{0,1\}$.
Promise: There are K inputs, say $x_1, x_2, \ldots, x_K$ such that $f(x_1) = f(x_2) = \cdots = f(x_K) = 1$. Otherwise, the functions output zero.
Problem: Find one of the K desired inputs.

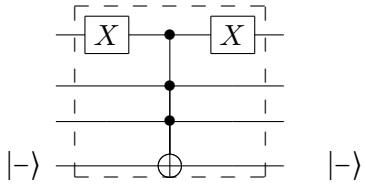
(a) **(5 points)** We are searching in a 3-bit list $\{\text{'000'}, \text{'001'}, \ldots, \text{'111'}\}$ for some target x_0. Assume $x_0 = 011$ is the target we want but we do not know. However, we are given a *oracle*:

$$f(x) = \begin{cases} 1 & x = \text{'011'} \\ 0 & \text{otherwise} \end{cases}.$$

Using the associated quantum oracle $U_f : |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |f(x) \oplus y\rangle$, we want to create a *phase flip* operation, $I_{|x_0\rangle}$, that will flip the sign of the coefficient of the state $|x_0\rangle$:



Now, verify the following circuit does the job of U_f:



You don't have to mathematically prove it. Instead, you may use the rotation gates or `qc.initialize` to create an arbitrary state $|\psi\rangle = \sum_{x\in\{0,1\}^3} a_x|x\rangle$ (by using '`state_simulator`' you will know all the coefficients a_x). Then, pass $|\psi\rangle$ through the above mentioned circuit to see if it does the job: $I_{|x_0\rangle}|\psi\rangle = -a_{011}|011\rangle + \sum_{x\neq 011} a_x|x\rangle$.

Hint. You may need the *multi-controlled Toffoli* gate:

```
qc.mct(list(range(nqubits-1)), nqubits-1)
```

(where `nqubits` is the number of total qubits in the circuit and the second argument means which register acting as the target register).

Remark. You may wonder that there are uncountably many 3-qubit states out there but you can only try just a bunch of them in your lift time. What if it is just a coincidence that you chose the lucky ones in verifying the circuit? How can we be so sure that there is no '*black swan*'?

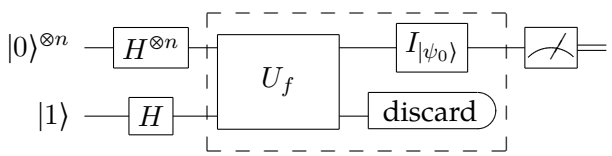
That's true. The '*black-swan event*' might happen especially in high-dimensional spaces. That's exactly the reason why theoretical analysis is substantial and indispensable. However, if the random states you chose, say $|\psi\rangle$, are so arbitrary and non-trivial (e.g. all a_x are non-zero and irrational numbers), what is the odd that the circuit does the job for your lucky $|\psi\rangle$ miraculously but not for some other 3-qubit states? I'm not saying that this is a *correct* way or a *right* attitude in doing research, but it is a convenient and quick way for ruling out what's not right. This is exactly the role of doing implementation as a pre-analysis (and is also one of the goals of this experiment course).

(b) **(15 points)** Next, design a circuit that does the *inversion about mean* operation:

$$I_{|\psi_0\rangle} := H^{\otimes n}\left(2(|0\rangle\langle 0|)^{\otimes n} - I\right)H^{\otimes n},$$

where $|\psi_0\rangle := \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle$ denotes the equal superposition. You may require *multi-controlled Toffoli* again. Draw your circuit (via `QISKit` or anyway).

(c) **(15 points)** Rue the Grover's algorithm: Prepare an equal superposition state as an initial input, and fit it in the U_f gate and the $I_{|\psi_0\rangle}$ gate iteratively as follows:

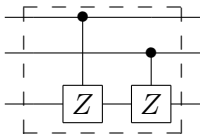


Since we are doing implementation, plot the probability of obtaining each state (including the probability of measuring the target x_0) along each stage (so you may produce several

figures). That is, applying '`qasm_simulator`' after each iteration to see the how the state evolves. After iterating the procedure by $\lfloor \sqrt{N} \rfloor$ times, do you always get the right solution '011' from the measurement outcome?

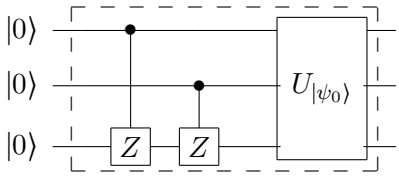
Also, plot the probability of getting the right solution versus the number of iterations. See what's going on up to, say 20 iterations.

(d) **(5 points)** In Problem 2a, we have seen a quantum oracle U_f (taking an input n-qubit $|\psi\rangle$ and the output qubit $|-\rangle$) can serve as a phase flip operation, $I_{|x_0\rangle}$, which will flip *a* target ket vector $|x_0\rangle$. Now, let us consider another example where both '011' and '101' are our targets. We aim to find either one of them. Verify that the following circuit flip the sign of '011' and '101':



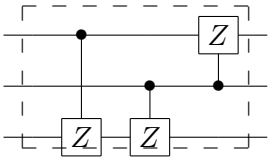
Note that here we do not adopt the quantum oracle U_f but instead we directly apply a 3-qubit circuit to flip the two target ket vectors.

(e) **(5 points)** Implement Grover's search algorithm for finding either '011' or '101':



How many queries do you need for this time to solve the problem? What is the probability of success? How about using IBM's real device?

(f) **(5 points)** Let us further consider another example where '011', '101', '110', '111' are our targets. Use the following circuit as such oracle, and run Grover's algorithm on it. What do you get?



(*Bonus 10 points*) Design a circuit for finding one target in a 4-bit unsorted list. Which one as the target is up to you. You can compare your established circuit of quantum oracle with `grover_problem_oracle` in `qiskit_textbook.problems` (but do not use it in this bonus problem). Then, choose 4 targets and design a circuit for finding them. How many queries do you need? Do you always get the desired target?

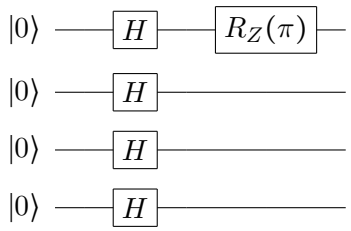
(*Bonus 0 point*) In Problem 2, which part do you feel the most difficult with? Let us optimistically presume that a large-scale and low-noise programmable (say, 100-qubit) quantum computer will appear in a short time. Imagine how to implement Grover's algorithm on it to solve real-world unstructured searching problem. Please provide your thought and perspective. What are

the challenges in front? (Note that I intentionally make this bonus problem 0 point since this is my favorite one in Problem 2 and moreover I think it's invaluable. Does grade matter?)

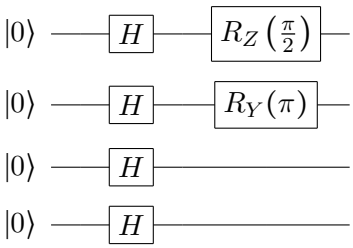
Remark. I know that searching in a 3-bit list is pretty boring. The quantum advantage over the classical one is negligible. So I encourage you to incorporate Grover's algorithm in your final project since searching is ubiquitous and it could be used as a subroutine in many algorithms. Then, implement it on IBM's real device with more qubits (). For example, interested students can refer to algorithms finding minimum [DH], optimization for the support vector machine [ARRZ03], or the k-means clustering [ABG06; ABG12].

3. **(15 points) Quantum Fourier Transform:** In the above Problem 2, we have seen the power of the *amplitude amplification* algorithm—once we flip the sign of our the target state, the *inversion about mean* will amplify the probability amplitude of it. However, in some scenarios even though the state contains different phases in each basis state, we might need other methods to extract the information (i.e. the *frequency*) embedded in the state.

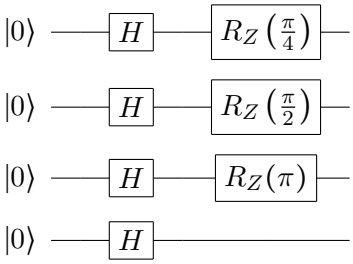
   In this problem, let us consider three states $|\phi_1\rangle$, $|\phi_2\rangle$, and $|\phi_3\rangle$ generated by the following three circuits (1), (2), and (3), respectively. If you measure each of them directly, what do you get?



(1)



(2)



(3)

Now, let us consider the *Quantum Fourier Transform* mod N by:

$$\mathbf{QFT}_N : |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \mathrm{e}^{2\pi \mathrm{i} \frac{x}{N} y} |y\rangle, \quad \forall x \in \{0, 1, \ldots, N-1\},$$

and its inverse:

$$\mathbf{QFT}_N^\dagger : |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \mathrm{e}^{-2\pi \mathrm{i} \frac{x}{N} y} |y\rangle, \quad \forall x \in \{0, 1, \ldots, N-1\}.$$

We are not going to implement $\mathbf{QFT}_N$ via elementary quantum gates here. Instead, we will use the sample code provided by the `QISKit` Textbook to see what $\mathbf{QFT}_N$ is capable of. The interested students may refer to Chapter 3.5 of the `QISKit` Textbook.

Now apply $\mathbf{QFT}_N^\dagger$ to $|\phi_1\rangle$, $|\phi_2\rangle$, and $|\phi_3\rangle$, respectively, and see what do you get. Why is that? Please explain it. What information was retrieved by the quantum Fourier transform?

```
def qft(n):
"""n-qubit QFT the first n qubits in circuit"""
    qc = QuantumCircuit(n)
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cu1(np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT"
return qc
```

```
def qft_dagger(n):
"""n-qubit QFTdagger the first n qubits in circuit"""
    qc = QuantumCircuit(n)
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cu1(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT"
return qc
```

Remark. In this problem, if you apply the amplitude amplification algorithm on $|\phi_1\rangle$, $|\phi_2\rangle$ or $|\phi_3\rangle$, you might not obtain useful information.

4. **(25 points) Period-Finding Algorithm:** In this problem, we will demonstrate a quantum algorithm (by using the quantum Fourier transform) to determine a period of a function f with some constant level of probability that is independent of the size of the domain of f. It can be shown that $O(\sqrt{N})$ queries to f are sufficient and necessary to achieve the goal, while one can use only $O(\log \log N)$ queries in the quantum scenario.

   Some other important mathematical problems such as the *integer factorization* can be reduced to problems of periodicity determination[1]. That's how Peter Shor's ground-breaking algorithm can provide an polynomial-time quantum algorithm while the best classical algorithm still exponentially many computations[2]. (However, we are not going into the details of Shor's algorithm but you can refer to Chapter 3.7 of the `QISKit` Textbook.)

   Consider the following periodicity determination problem.

   Periodicity Determination: Finding the Period r of a Periodic State Given N

   Given: A black box for a function $f : \mathbb{Z}_N \to \mathbb{Z}_M$.
   Promise: The function f is periodic with some period r, i.e. there is a smallest number r such that $f(x + r) = f(x)$ for all $x \in \mathbb{Z}_N$.
   Problem: Find the period r.

   Step 1. Construct a uniform superposition $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$ and send it through the quantum oracle U_f to obtain the *periodic state* $|f\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|f(x)\rangle$.

   To implement this. Let us consider the following example: $N = 2^n = 2^8$, $M = 2^4$, and $f(x) := a^x \bmod 15$ with $a = 7$. Show that f is periodic (using a calculator or `Matlab`).

   We adopt the following sample code from the `QISKit` Textbook to generate the state $|f\rangle$.

```
# Create QuantumCircuit with n_count counting qubits plus 4 qubits for U
#to act on n_count = 8 # number of counting qubits
a = 7 # a variable that can be adjusted later
# The first 8-qubit register for storing x
qr1 = QuantumRegister(n_count, name="q1")
# The second 4-qubit register for storing f(x)
qr2 = QuantumRegister(4, name="q2")
cr1 = ClassicalRegister(n_count, name="c1")
cr2 = ClassicalRegister(4, name="c2")

qc = QuantumCircuit(qr1, qr2, cr1, cr2)
```

[1] In complexity theory, we say that a computational problem A is *reducible* to another problem B is that, if we have a subroutine, say S, for which when run we can solve for problem B, and this subroutine S can also solve problem A.

[2] Thus, many people claimed that Shor's algorithm provides an *exponential speed* over the classical one since that it requires polynomially many queries for periodicity determination. Nevertheless, how do you know that there is no classical algorithms other than the period-finding method that can solve the integer factorization efficiently?

```
# Initialize counting qubits in uniform superposition
for q in range(n_count):
    qc.h(q)

# And ancilla register in state |1>
qc.x(3+n_count)

# Do controlled-U operations
for q in range(n_count):
    qc.append(c_amod15(a, 2**q),
    [q] + [i+n_count for i in range(4)])
```

```
def c_amod15(a, power):
    """Controlled multiplication by a mod 15"""
    if a not in [2,7,8,11,13]:
        raise ValueError("'a' must be 2,7,8,11 or 13")
    U = QuantumCircuit(4)
    for iteration in range(power):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a == 11:
            U.swap(1,3)
            U.swap(0,2)
        if a in [7,11,13]:
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "%i^%i mod 15" % (a, power)
    c_U = U.control()
    return c_U
```

Step 2. Measure the second register (i.e. `qr2`). Use `plot_histogram()` (and `qasm_simulator`) to see what you get (in decimal form). How likely are them? Why is that? (You can answer

this in the end when you completely understand the whole procedure.) Ideally, you should see some $y_0 = f(x_0)$ for some x_0 is the least of x having $f(x) = y_0$.

Step 3. Now, the first register (i.e. `qr1`) should be projected into *equal* superposition of A values: $x = x_0$, $x = x_0 + r$, $x = x_0 + 2r$, ..., $x = x_0 + (A-1)r$ with $A = \frac{N}{r}$ and for which $f(x) = y_0$. In other words, for the y_0 you saw from the second register, `qr2`, the first register, `qr1`, will be a *periodic state*:

$$|\text{per}\rangle = \frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle.$$

Here, note that $0 \le x_0 \le r-1$ has been chosen at random. (why?) On the other hand, if we measure the register $|\text{per}\rangle$ we will see $x_0 + j_0 r$ where j_0 has been picked uniformly at random too. Hence, we have a random period (the j_0-th period) and a random shift (determined by x_0). So overall we get a random number between 0 and $N-1$. Verify this by measuring both `qr1` and `qr2`.

Step 4. Apply $\mathbf{QFT}_N^\dagger$ on $|\text{per}\rangle$. Show that all terms have zero amplitude except for the multiples of $A = \frac{N}{r}$, i.e. you should get $c = k \cdot A$ for $k = 0, 1, 2, \ldots, r-1$. What did you really get (in decimal form)? (You may need the following sample codes.)

```
# Do inverse-QFT
qc.append(qft_dagger(n_count), range(n_count))
# Measure circuit
qc.measure(range(n_count), range(n_count))
```

Step 5. Continuing from the above step, use `Fraction()` to read-off the denominator of $\frac{c}{N} = \frac{k_0}{r}$ for certain c you obtained before. Here, k_0 is the actually multiples of A but you don't known what it is. If k is coprime to r, then you will get the desired period r. Otherwise, the value, say r', you read will be smaller than the true period. So $f(x) \ne f(x+r')$ for all $x \in \mathbb{Z}_N$, which causes an error. Hence in our process, we can check the output value r by evaluating $f(0)$ and $f(r)$ and accepting r as the correct period if they are equal. What is the probability of getting the right period in this case?

Note that k_0 is chosen at random. What is the chance that k_0 happens to be coprime to r? By the *Coprimality theorem* in number theory, it was shown that such chance is of the order $O(1/\log\log r)$. Hence, if we repeat the whole process $O(\log\log r) < O(\log\log N)$ times we will obtain a coprime k_0 in at least one case with a constant level of probability. (You may need the following sample codes.)

```
import pandas as pd
from fractions import Fraction
rows, measured_phases = [], []
```

```
for output in counts:
    decimal = int(output, 2)  # Convert (base 2) string to decimal
    phase = decimal/(2**n_count) # Find corresponding eigenvalue
    measured_phases.append(phase)

for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(15)
    rows.append([phase, "%i/%i" % (frac.numerator, frac.denominator), frac.denominator])
# Print as a table
headers=["Phase", "Fraction", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

Try different $a = 2, 8, 11$ or 13 to see what you get. Note that the goal of this Problem 4 is for you to understand the whole process of the period-finding algorithm by using quantum Fourier transform.

Remark. To really implement the period-finding algorithm for general purposes, we would need some quantum circuits for generating the periodic state. Here, we adopted a specific function[3] $f(x) = a^x \bmod 15$. How about the scenarios that the modular value is other than 15? Beauregard proposed a method to conveniently generate the circuit we want [Bea03]. Hence, I highly encourage the interested students may implement the complete Shor's algorithm step by step as your final project. You can also refer to the ProjectQ which is an open source platform for quantum computing.

3 Lab Report

There is no format/typesetting requirements for your lab report, but you have to make your report decent and looking nice. In the report, you should address the results of the exercises mentioned above. You should also include your simulation program in the appendix of the report. Include whatever discussions about the new findings during the lab exercise, or the problems encountered and how are those solved. Please properly cite the literature if you referred to. Do not limit yourself to the exercises specified here. You are highly encouraged to play around with your simulation program on self-initiated extra lab exercises/discussions.

This is the last Lab for implementing quantum information-processing tasks in this course. I believe that you have had basic understanding of the quantum circuit model, quantum algorithms, and how to use the `QISKit` to perform small-scale simulation (or using IBM's real devices). However, there are a plethora of open software out there. Please see the *Quantum Software Toolbox* page in our

[3] Note that for some r we have $f(r) = a^r \bmod N = 1$. We call such r the *order* of the function f. So the periodicity determination is equivalent to the order determination.

NTU COOL course website. It is highly recommended that you can play around some of them and do your final project by using some of them.

References

[DH] C. Durr and P. Hoyer, "A quantum algorithm for finding the minimum,"

[ARRZ03] D. Anguita, S. Ridella, F. Rivieccio, and R. Zunino, "Quantum optimization for training support vector machines," *Neural Networks*, vol. 16, no. 5-6, pp. 763–770, 2003. DOI: 10.1016/s0893-6080(03)00087-x.

[ABG06] E. Aïmeur, G. Brassard, and S. Gambs, "Machine learning in a quantum world," in *Can. AI 2006*, 2006, pp. 431–442. DOI: 10.1007/11766247_37.

[ABG12] ——, "Quantum speed-up for unsupervised learning," *Machine Learning*, vol. 90, no. 2, pp. 261–287, 2012. DOI: 10.1007/s10994-012-5316-5.

[Bea03] S. Beauregard, "Circuit for shor's algorithm using $2n + 3$ qubits," 2003. arXiv: quant-ph/0205095.