Communication Systems Laboratory
Lab 2: Applications of Quantum Information Processing

(Report Due: 21:00, October 21, 2020)

1 Overview

In the previous Lab 1, we have properly installed the QISKit package, and we have known how to create a basic quantum circuit by composing certain quantum gates. Further, we are able to prepare an arbitrary 1-qubit state and also create a maximally entangled state shared between two qubit systems (known as the EPR pair or the Bell state). In this Lab, we are going to explore how quantum resources such as superposition and entanglement can yield new applications in quantum information processing. (Actually, we have seen that Quantum Random Number Generation offers exponential large capacities for generating random numbers by harnessing the power of superposition). Specifically, we will use the QISKit package to implement (i) superdense coding—an entanglement-assisted classical communication scheme; (ii) quantum teleportation—a quantum communication scheme; and (iii) the BB84 protocol—a quantum key distribution protocol.

Also note that in Lab 1, we actually simulate the quantum circuit by the QISKit simulator, which is purely a *classical computer*. In this Lab, you can still use the QISKit simulator to simulate the quantum circuit, or you can choose to use IBM's real device, which is a small prototype of noisy quantum processor. I highly encourage you to try them both, in order to see what is the gap between theory and reality. To that end, you might need the following codes to call the least busy quantum computer from IBM (otherwise you might have to wait in the long queue):

```
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
shots = 256

# Load local account information
IBMQ.load_account()
# Get the least busy backend
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x:
x.configuration().n_qubits >= 2
and not x.configuration().simulator
and x.status().operational==True))
print("least busy backend: ", backend)
# Run our circuit
job = execute(qc, backend=backend, shots=shots)
```

```
# Monitoring our job
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

You can log in the website of *IBM Quantum Experience* to see which quantum computers are currently available[1].

2 Experiments

1. **(30 points) Superdense Coding:** In this problem, you will implement the superdense coding as taught in class by using the QISKit , which allows us to send 2-classical bits simultaneously by using an 1-qubit noiseless channel and an EPR pair.

   (a) **(10 points)** The goal of Alice is to send arbitrary 4-bit classical strings to Bob for which she is allowed to freely use a 2-qubit noiseless quantum channel. Moreover, we assume that a third party, say Charlie, can prepare arbitrary quantum entanglement (e.g. the EPR pair) shared between Alice and Bob.

   In this problem, please randomly generate bit strings at Alice, e.g. by

   ```
   from numpy.random import randint
   import numpy as np

   # Setting the random seed
   np.random.seed(seed=0) # You may choose any seed instead of just `0'
   n = 4 # This is the number of bits in the string
   alice_bits = randint(2, size=n)
   ```

   and use the 'qasm_simulator' in QISKit to simulate the above mentioned superdense coding. If we regard every 2-bit string as a 'symbol' or a 'package' that delivers our classical information, please calculate the corresponding *symbol error rate* or *package error rate* (i.e. you may perform measurement and the protocol several times to compare if Bob's received symbols or packages are really what sent by Alice). Further, please calculate the *bit error rate*. For example, if the bit string '01' is mistaken for '11', we say a bit error happens when sending two bits. Hence, the bit error rate for it is 50%.

   (b) **(15 points)** Please use the real quantum computer provided by IBMQ to run the above mentioned superdense coding and calculate the corresponding symbol error rate and the bit error rate.

[1] Further information could be found from the examples provided in the QISKit Textbook or online resources like https://medium.com/qiskit/qiskit-backends-what-they-are-and-how-to-work-with-them-fb66b3bd0463.

2. **(45 points) Quantum Teleportation:** In this problem, you will run quantum teleportation for communicating a qubit information.

   (a) (20 points) The goal of Alice is to send an arbitrary 1-qubit state to Bob. Again, we assume that a third party, Charlie, can prepare an EPR pair shared between Alice and Bob. So firstly, use your code in Problem 2-(g) of Lab 1 (or any other reasonable methods) to prepare an arbitrary 1-qubit state. Send it to Bob via teleportation.

   Since we are doing implementation, we must verify that whether the state received by Bob is still the same as Alice's original state. There are several ways to do that:

   (i) Use the '`statevector_simulator` in QISKit to show the mathematical descriptions of Alice's original state, say $|\psi\rangle_A$, and Bob's received state, say $|\psi'\rangle_B$ (as described in Section 3.3 of the QISKit Textbook), and calculate their inner product $|\langle\psi|\psi'\rangle|^2$. However, note that you couldn't use the '`statevector_simulator` on real devices because it's not a usual quantum gate operation[2].

   (ii) After Bob received the state, perform the *inverse* operation for what Alice prepared the state. Then you should get back the $|0\rangle$ state Use the '`qasm_simulator`' to see if this is true (as described in Section 3.3 of the QISKit Textbook again).

   (iii) The last one is my favorite. Alice prepares the state identically and independently on two different registers, and just send one of them to Bob. After Bob's received it, use the Swap test in Problem 4 of Lab 1 (by a third party, say an omnipotent Judge) to see if they are the same.

   You should run this simulation several time, probably with different preparation of the states at Alice to calculate the performance of the teleportation. It is highly recommend that you use IBM's real quantum computers.

   Hint: Since quantum teleportation relies on classical communication, you might need something like `qc.x(qubit).c_if(crx, 1)`, where it means that applies the X gate on the quantum register '`qubit`' if the classical register '`crx`' reads 1.

   (b) (25 points) The *Entanglement Swapping*, or called the *Quantum Repeater*. In this problem, we will see an application of quantum teleportation. Suppose Alice wanted to create a Bell state $|\Phi^+\rangle$ shared between herself and Bob. However, Alice and Bob are separated in a long distance, and somehow due to environmental effects (such as air pollution etc.) Alice is not able to establish the shared entanglement (e.g. by locally preparing $|\Phi^+\rangle$ and sending one of the qubit over to Bob). Luckily, their friend Charlie who locates in the midway between Alice and Bob such that Alice can share a Bell state, say $|\Phi^+\rangle_{AC_1}$, between herself and a subsystem of Charlie, and at the mean while Bob can share a Bell state, say $|\Phi^+\rangle_{C_2B}$, between himself and another subsystem of Charlie.

[2] Actually, you can perform the so-called *quantum state tomography* to pin down the mathematical description of the state. However, this would require multiple states of it. In other word, if you only have one copy of an unknown state, you couldn't know it's mathematical description by performing just one single measurement unless you measure it in the right basis (but how would you know).

Now, since Charlie shares a Bell state with Bob (i.e. $|\Phi^+\rangle_{C_2B}$), Charlie exploits such entanglement to send his qubit (his subsystem C_1) to Bob via teleportation. At the end of the protocol, Alice is now sharing a Bell state, say $|\Phi^+\rangle_{AB}$, between herself and Bob as desired. Please implement such entanglement swapping.

Hint: Create 4 qubit-registers, say $q_0q_1q_2q_3$. Alice posses q_0; Charlie posses both q_1 and q_2, and Bob has q_3. Just ignoring q_0, Charlie performs the Bell measurement on q_1q_2 and tell Bob his measurement outcome so that Bob can correct his state. Lastly, note that that there are several ways to verify that you end up with the desired $|\Phi^+\rangle_{q_0q_2}$ such as just reading the outcomes, plotting it via `plot_bloch_multivector()`, reversing it to $|00\rangle_{q_0q_2}$.

3. **(30 points) The BB84 Protocol:** In the previous problems, we have seen applications of using entanglement as a quantum resource for communicating both classical information (i.e. superdense coding) and quantum information (i.e. quantum teleportation). In this problem, we will see how quantum information processing can be used for distributing keys in cryptography.

   The *BB84 Protocol* is one of the most basic and substantial quantum key distribution scheme invented by Charles Bennett and Gilles Brassard in 1984 [BB84]. The goal is for two parties, Alice and Bob, to share the same key sequence. However, if Alice and Bob are separated, they must employ some communication scheme to share the key. Either in theory or in practice there are eavesdroppers listening the process of the key sharing. At worst, Alice and Bob might still use their key without being aware that their secret key has been known by a malicious adversary. To address this issue, there are several approaches in classical cryptography been proposed. However, most of the methods are expensive in implementation.

   The advantage of the BB84 protocol that it is not only simple but *information-theoretically secure*[3] as well. Most importantly, it can be implemented by just an 1-qubit computer. Hence, it has been commercialized by several companies. In what follows, we will implement parts of the BB84 protocol by adapting the source codes given in the QISKit Textbook. So you will have a flavor of how it works and how superposition as a quantum resource can help for encryption.

   Before starting the protocol, make sure that you have imported necessary toolboxes:

```
from qiskit import *
from qiskit.visualization import plot_histogram
from numpy.random import randint
import numpy as np
from math import pi
```

   (i) *Step 1*. Firstly, Alice wanted to send a (uniformly) random key which is composed by a

[3] We won't touch the precise definition of the information-theoretic security here, but it generally means that the protocol satisfying certain security criterion cannot be broken (with high probability) no matter how much computational power possessed by the eavesdroppers.

bit sequence. She wants to employ the power of quantum information processing to send such key to Bob *securely* hopefully.

```
n = 100; # number of qubits used in the BB84
np.random.seed(seed=0)

## Step 1
# Alice generates bits
alice_bits = randint(2, size=n)
print(alice_bits)
print("Alice's first bit = %i" % alice_bits[0])
```

Here, choose whatever number of qubits and random seeds you want.

(ii) *Step 2*. Secondly, Alice can naıvely encode bit 0 into $|0\rangle$ and encode bit 1 into $|1\rangle$. Then, Alice can use quantum communication such as quantum teleportation to send her qubits to Bob. However, if eavesdropper (say Eve) knows that Alice used the computational basis to encode the key[4], She will just intercept Alice's sent qubits and measure them with respect to the computational basis. Eve will know the key precisely and then resend the qubits (in a way just as Alice did before) to Bob. Clearly, there is no quantum advantage if Alice only uses the computation basis to encode her key, and Eve has intercepted the keys entirely.

Actually, Alice has the freedom to choose different bases to encode the key. For example, she can uniformly generate another bit sequence. If it is '0', then she uses the computational basis as before; otherwise, she chooses the $\{|+\rangle, |-\rangle\}$ basis to encode the key, i.e. $0 \mapsto |+\rangle$ and $1 \mapsto |-\rangle$.

Since the random bit sequence (for determining the bases) is generated and known by Alice solely, Eve would not know exactly which one Alice chose. So here are the resources Alice exploited: the classical randomness (which is common in classical cryptography) and the degrees of freedom for which Alice can choose different bases to encode her key (and this is due to the *quantum coherence* or equivalently the superposition). The sample code of Step 2 is given as the following, where the function `encode_message(,)` can be found in Section 3.12 of the QISKit textbook or you can accomplish it by yourself (which I highly encouraged):

[4]We didn't talk about how Eve knows that Alice used the computational basis. However, how do we know that Eve *does not* know that? If Alice used the computational basis all the time, someone might leak this little secret to others. So in cryptography we seldom put constraints on the eavesdropper. Yeah, I agree with you. Cryptographer always thought that enemies are malicious just like most of theoretical computer scientists are so pessimistic since they always consider the worst case. But thanks to all the cryptographers, the financial and information systems work (hopefully) safely in our daily lives.

```
## Step 2
# Create an array to tell us which qubits are encoded in which bases
alice_bases = randint(2, size=n)
message = encode_message(alice_bits, alice_bases)
print(alice_bases)
print("Alice's first chosen basis = %i" % alice_bases[0])
```

(iii) *Interception!* Now Eve tries to intercept the qubits sent by Alice. As mentioned before, Eve does not know which bases Alice chose because the random bit sequence (i.e. `alice_bases`) is completely random to Eve.

What can Eve do? (Now you might be have some senses how BB84 could work.) Well, let Eve try the following. She also (uniformly) generates her random bit to determine which bases she wanted to perform measurement in. For example, if her random bit turns out to be '0', she measures the qubit with respect to the computational basis. If the measurement outcome was '0', she sent $|0\rangle$ to Bob; otherwise, she sent $|1\rangle$ to Bob. If her random bit turns out to be '1', she measures the qubit with respect to the $\{|+\rangle, |-\rangle\}$ basis. Again, if the measurement outcome was '0', she sent $|0\rangle$ to Bob; otherwise, she sent $|1\rangle$ to Bob.

This can be done by the following:

```
## Interception!!
eve_bases = randint(2, size=n)
intercepted_message = intercept_message(message, eve_bases)
print(eve_bases)
print("Eve's first chosen basis = %i" % alice_bases[0])
print(intercepted_message)
print("Eve's first intercepted message = %i" % intercepted_message[0])
```

```
def intercept_message(message, bases):
backend = Aer.get_backend('qasm_simulator')
measurements = []
for q in range(n):
if bases[q] == 0: # measuring in Z-basis
message[q].measure(0,0)
if bases[q] == 1: # measuring in X-basis
message[q].h(0)
message[q].measure(0,0)
message[q].h(0) # preparing the post-measurement state
result = execute(message[q], backend, shots=1, memory=True).result()
```

```
measured_bit = int(result.get_memory()[0])
measurements.append(measured_bit)
return measurements
```

(iv) *Step 3.* Now it is the time that Bob gets the key. Recall that Alice chose a random basis (the $\{|0\rangle, |1\rangle\}$ basis or the $\{|+\rangle, |-\rangle\}$ basis) in attempt to fool Eve. But this will fool Bob too. If Bob asked Alice to announce the bit sequence for determining her basis at this moment, Eve would hear that too and hence the protocol will be compromised (that is, Eve can wait until Alice and Bob compare their bit sequence publicly)!

Consequently, Bob has no choice but to do exactly what Eve did in the previous phase, which can be summarized as the following sample code:

```
## Step 3
# Decide which basis to measure in:
bob_bases = randint(2, size=n)
bob_results = measure_message(message, bob_bases)
print(bob_bases)
print("Bob's first chosen basis = %i" % bob_bases[0])
```

(v) *Step 4.* Okay. Now Bob has obtained his measurement outcome as a bit sequence. Let us first assume that there no interception by Eve. If Alice and Bob both chose the same basis (say for a certain qubit) in Step 2 and Step 3, then Bob's measurement outcome should exactly coincide with Alice's bit. Otherwise, Bob's measurement outcome should be statistically independent to Alice's bit, e.g. if Alice's bit is '0' then Bob should get '0' or '1' equally likely. (Why? Please figure this out or you can refer to the QISKit Textbook.)

So, Alice and Bob would like to check whether their chosen bases at Step 2 and Step 3 coincide or not. To that end, Alice and Bob now announce their random bit sequence (for determining their bases) and compare them publicly. (Note that the comparison comes after Bob finishing his measurement.) For bits for which they chose the wrong bases, they just discard them:

```
## Step 4
# Remove bits where Alice's chosen bases are not equal to that of Bob's
alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
print(alice_key)
print(bob_key)
```

Here, the function `remove_garbage(,,)` can also be found in the QISKit Textbook.

(vi) *Step 5.* In average, Alice and Bob will discard half of their bit sequences. If there is no eavesdroppers, Alice and Bob would have exactly the same bit sequence. Then, They can use the sequence as the shared secret key. However, in reality there might be communication errors or eavesdroppers, so it could be possible that Alice and Bob got some bit sequences. So in the BB84 protocol, they will randomly choose some of their remaining key sequence (say e.g. half of the remaining sequence again) to compare whether they match or not. If they spot any differences in the announced bit sequence, they will suspect that eavesdroppers were listening. So they will abort the protocol and discard the key sequence entirely.

To address these issues, the BB84 concludes with the so-called *information reconciliation* and the *privacy amplification*, which are techniques coming from classical cryptography. Remarkably, Alice and Bob can sacrifice some key length so that Eve has no knowledge of their shared key sequence whatever with high probability. Further, by using some techniques in quantum information theory and quantum error correction codes, the BB84 protocol can be proved to be *information-theoretically secure* [SP00]. However, the rigorous treatments are beyond the scope of this course.

The goal of this problem for you is to clearly understand the above procedure.

Remark. Here, we have used many qubits to implement the BB84 protocol, but actually it can be achieved by using just an 1-qubit circuit (note that we only used an 1-qubit Hadamard gate and an 1-qubit measurement in the protocol) and by repeating it many times. That's the reason why it has been successfully built into commercial devices. (Also note that the BB84 protocol uses quantum communication; this might require a multiple-qubit system to prepare entanglement for teleportation.)

(a) (10 Points) After Alice and Bob announcing their bit sequence (for determining the bases), Eve can hear that too and throw away those bits as Alice and Bob did. Now Alice and Bob got their shared secret key, but Even might still guess some bits of the key correctly. In this problem, your goal is to run the above codes to calculate the probability of Eve guessing Alice's bit correctly.

You should obtain a probability $c \in (0, 1)$. As mentioned in Step 5, if Alice and Bob spotted any difference by using half of their remaining bits (say, using 50 bits to compare), they abort the protocol. Then, the probability that Alice and Bob do not abort the protocol and at the same time Eve guesses correctly Alice's bits will be c^{50}, which is exponentially small in the number of qubits used for comparison. In the other words, Eve's presence is almost negligible.

Hint: Since you are the programmer, you are in God's view. After executing Step 3, you got everything you need, i.e. the key sequence that Alice and Bob will keep, and Eve's intercepted messages. Now you should be able to calculate the percentage of what Eve guessed correctly.

(b) (20 Points) Eve has the freedom to choose the bases (even though she does not know which one Alice chose). You can try the scenario that Eve just fixes her measurement with respect to the computational basis or with respect to the $\{|+\rangle, |-\rangle\}$ basis. Now what's the probability of Eve guessing correctly? (This does not count the points but you are very encouraged to try it.)

In this problem, please try the following scenario: Eve fixes the basis

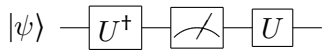
$$\left\{\cos\tfrac{\pi}{8}|0\rangle + \sin\tfrac{\pi}{8}|1\rangle, \sin\tfrac{\pi}{8}|0\rangle - \cos\tfrac{\pi}{8}|1\rangle\right\}$$

which is called the *Breidbart basis*, and find the probability by running your codes again.

Hint: In the QISKit , the measurement is by default with respect to the computational basis. However, just like Problem 3-(h) in Lab 1: you can put some quantum gates before the measurement such that it acts just like the measurement with respect to desired basis according to Born's rule. For example, let us assume that you wanted to perform the measurement with respect to the basis $\{|b\rangle, |b^\perp\rangle\}$, where $|b\rangle = U|0\rangle$ and $|b^\perp\rangle = U|1\rangle$ for some gate U (a unitary operator). Then, if you want to use it to measure the state $|\psi\rangle$. By Born's rule, we have

$$|\langle\psi|b\rangle|^2 = |\langle\psi|U|0\rangle\rangle|^2 = |(U^\dagger|\psi\rangle)^\dagger|0\rangle|^2.$$

So it is equivalently to apply $U^\dagger$ before the usual measurement:



Note that $U^\dagger = U^{-1}$ by unitarity, and applying the gate U after the measurement is to prepare the post-measurement state (i.e. the state that the state collapses to). The hint is that you can try the Rotation-Y gate:

$$R_Y(\theta) := \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

or the $U_3(\theta, \phi, \lambda)$ gate to achieve your goal.

Remark. The Bridbart basis can be viewed in the 'midway' between the computational basis and the $\{|+\rangle, |-\rangle\}$. You can see this from the Bloch sphere representation.

After thought. Now you might know what is the key ingredients why the BB84 protocol works—(i) Alice has the freedom to choose between the computational basis and the $\{|+\rangle, |-\rangle\}$ basis (noting that the classical system does not have such privilege), and (ii) once Eve measures the received qubits, they collapsed immediately (which is the consequence of the *No-Cloning Theorem*). This can be viewed as the fact that Eve cannot *perfectly distinguish* among the states $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$ simultaneously. You might wonder that can Alice choose other bases as well? Yes, indeed Alice can choose, e.g. some basis $\{|b\rangle, |b^\perp\rangle\}$, but this would not give the equal-likely outputs at Eve's end if Eve did not choose the right basis; and this would give Eve chances or information to guess which bits Alice sent most likely.

On the other hand, to do just like what the $\{|+\rangle, |-\rangle\}$ basis did, Alice can choose the basis $\{|\mathrm{i}\rangle = \frac{|0\rangle+\mathrm{i}|1\rangle}{\sqrt{2}}, |-\mathrm{i}\rangle = \frac{|0\rangle-\mathrm{i}|1\rangle}{\sqrt{2}}\}$ instead of $\{|+\rangle, |-\rangle\}$. What's the performance of it then? Actually, there are infinitely many such bases you can choose from. (You can see this from the Bloch sphere representation.) We call $\{|0\rangle, |1\rangle\}$ and $\{|+\rangle, |-\rangle\}$ the *mutually unbiased bases*. The formal definition is the following: the bases $\{|e_1\rangle, \ldots, |e_d\rangle\}$ and $\{|f_1\rangle, \ldots, |f_d\rangle\}$ satisfy that $|\langle e_j|f_k\rangle|^2 = \frac{1}{d}$ for all $j, k \in \{1, \ldots, d\}$.

Now you may wonder why not Alice generate a trit sequence to determine which of the three bases, i.e. $\{|0\rangle, |1\rangle\}$, $\{|+\rangle, |-\rangle\}$, and $\{|\mathrm{i}\rangle, |-\mathrm{i}\rangle\}$? More precisely,

$$\begin{cases} \text{encode } 0/1 \text{ into } |0\rangle/|1\rangle, & \text{if the random trit is '0'} \\ \text{encode } 0/1 \text{ into } |+\rangle/|-\rangle, & \text{if the random trit is '1'} \\ \text{encode } 0/1 \text{ into } |\mathrm{i}\rangle/|-\mathrm{i}\rangle, & \text{if the random trit is '2'} \end{cases}.$$

This will fool Eve more but also fool Bob too such that Bob might need to throw away $\frac{2}{3}$ of his sequence.

(*Bonus 10 points*) What is the performance (the probability of Eve guessing correctly Alice's bits) in this scenario? Which basis Eve can choose at best? This should be the 'midway' between the three mutually unbiased bases.

Further Remark ☺. In a qubit system, there are three mutually unbiased bases as mentioned above. You couldn't find more, but you can find more in higher dimensional systems. Some quantum cryptography protocols thus employ high-dimensional quantum systems to encrypt the keys. If you are interested in this topic, I highly recommend you to do some literature survey and implement/invent some more advanced quantum cryptography schemes.

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. For example, you can see the QISKit Textbook. We remark that the QISKit Textbook offers some useful functions such as the random state '`random_state()`'. To recall such functions, you have to firstly install the Git, and download the `qiskit_textbook` package via the command:

```
pip install git+https://github.com/qiskit-community
↪ /qiskit-textbook.git#subdirectory=qiskit-textbook-src
```

and '`import qiskit_textbook`'.

References

[BB84] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, vol. 175, 1984. [Online]. Available: http://researcher.watson.ibm.com/researcher/files/us-bennetc/BB84highest.pdf.

[SP00] P. W. Shor and J. Preskill, "Simple proof of security of the BB84 quantum key distribution protocol," *Physical Review Letters*, vol. 85, no. 2, pp. 441–444, 2000. DOI: 10.1103/physrevlett.85.441.