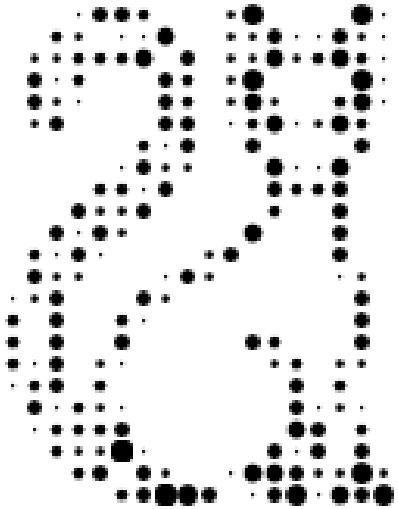
Introduction to Quantum Computing

Carla Silva, Vanda Azevedo, Diogo Fernandes, Inês Dutra

Department of Computer Science, University of Porto, Portugal



[dcc]

DEPARTAMENTO DE CIÊNCIA DE COMPUTADORES
FACULDADE DE CIÊNCIAS DA UNIVERSIDADE DO PORTO

Syllabus

- Definitions
- Quantum Basics
- Quantum hardware
- Quantum assembler
- Quantum programming
- Classical x quantum programming
- Challenges

What is quantum computing?

- It is to use quantum-mechanical phenomena to solve computational problems
- Quantum-mechanical phenomena:
  - superposition
  - entanglement
  - Quantum computers execute these operations and results are collected using **measurements**

Quantum-mechanical phenomena

- Superposition: allows the **qubit** to be in both states at the same time.
  - It can be defined as a weighted combination between the basic states 0 and 1.
- Entanglement: occurs when pairs or more than two particles are generated or interact in a way that the quantum state of each particle can't be described independently of the state of the other(s).

bit x qubit

- A classical von Neumann machine uses as a representation unit a **bit** which can have values 0 or 1
  - corresponding to low voltage values and high voltage values, respectively
- A quantum machine uses the **qubit** as a representation unit.
  - A qubit can assume two values, 0 and 1, simultaneously, thanks to superposition.

bra-ket notation

- **bra-ket** is a standard quantum mechanics notation for representing quantum states.
- In order to calculate the scalar product of vectors, the notation uses angle brackets $\langle$, $\rangle$, and a vertical bar $|$.
- The scalar product is then $\langle\phi|\psi\rangle$ where the right part is the "psi ket" (a column vector) and the left part is the bra – the Hermitian transpose of the ket (a row vector).
- Bra-ket notation was introduced in 1939 by Paul Dirac and is also known as the Dirac notation.

bit x qubit

- In more detail, a qubit is a quantum system in which the Boolean states 0 and 1 are represented by a pair of normalised and mutually orthogonal quantum states.
- The two states form a computational basis and any other (pure) state of the qubit can be written as a linear combination:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

with α and β probability amplitudes, which can be complex numbers.

bit x qubit

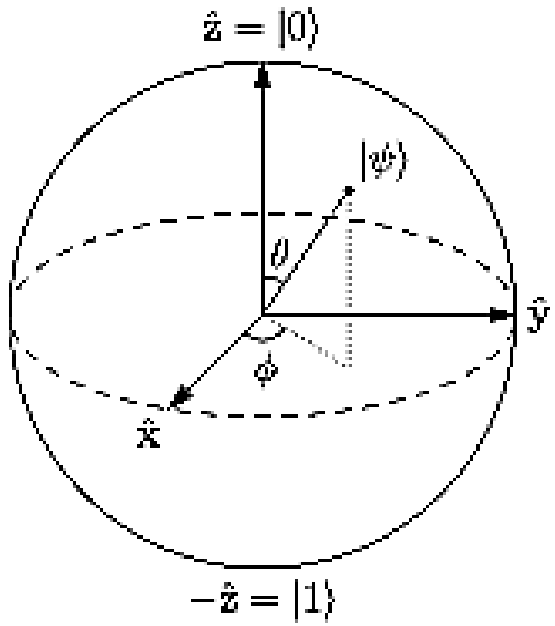
- In a **measurement** the probability of the bit being in $|0\rangle$ is $|\alpha|^2$ and $|1\rangle$ is $|\beta|^2$,

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

- Two real numbers can describe a single qubit quantum state, since a global phase is undetectable and the conservation probability is $|\alpha|^2 + |\beta|^2 = 1$

Bloch sphere

- The bloch sphere is a graphical representation of the several qubit states.
- We can describe the states of the qubit by
  $|\psi\rangle = \cos(\frac{\theta}{2})\,|0\rangle + \sin(\frac{\theta}{2})e^{i\phi}\,|1\rangle$
  with $0 \leq \phi < 2\pi$, and $0 \leq \theta \leq \pi$.
- Qubit states ($\mathbb{C}^2$) correspond to the points on the surface of a unit sphere ($\mathbb{R}^3$).

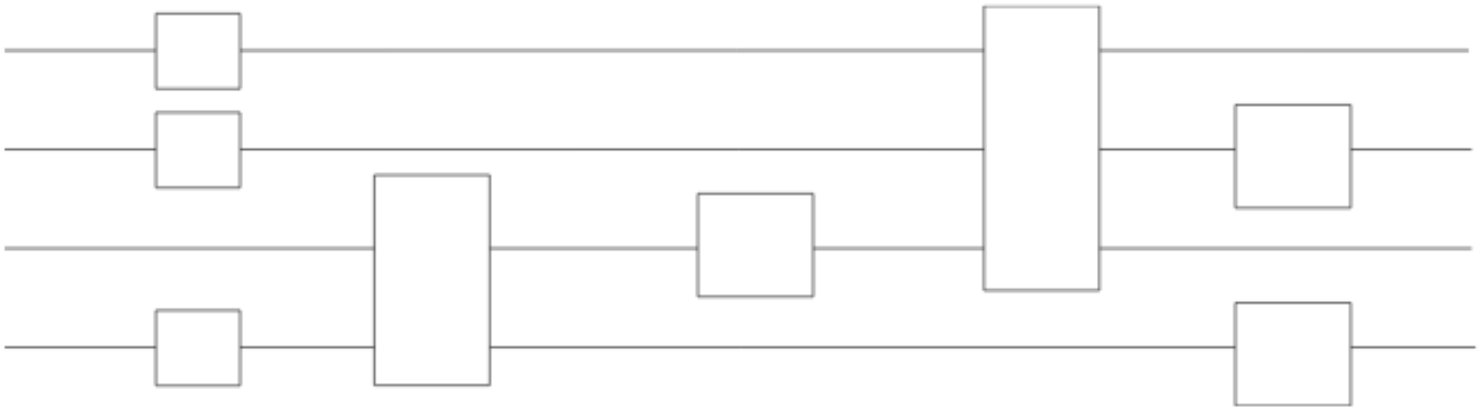


Measurement

- A measurement is the way we can output qubit states.
- The quantum system is deterministic until measurement.
- Measurement provides only one of the superposed states.
- The only one state becomes the new state and interferes with computational procedures.
- This state, is determined with some probability ≤ 1, implying uncertainty in the result states and cannot be copied ("cloned").
- Environmental interference may cause a measurement-like state collapse (decoherence).

Recalling: Circuit Model of Computation

- Circuits are networks composed of wires that carry bit values to gates that perform elementary operations on those bits.
- **gates** are basic units that perform boolean logical operations.
- Example of a generic circuit with space (width) 4, depth 5, composed of 8 gates:



Recalling: Circuit Model of Computation

- Of particular interest to quantum computing are the **reversible circuits**.
- A reversible circuit is the one where we can guess the inputs, given the outputs.
- The simplest reversible circuit (gate) is the NOT operator.
  - given the output 1, we know the input is 0 and vice-versa.
- On the other hand, the AND gate is not reversible, but it can be made reversible! (more on that later...)

Recalling: Circuit Model of Computation

- A circuit can be decribed by boolean linear algebra.
- A quantum circuit can not use exactly the same algebra because its states are **probabilistic**.
- As mentioned before, we represent a quantum circuit using vectors.
  - assuming just one bit with probability p_0 of being 0 and probabiity p_1 of being 1, we represent this information as a 2-dimensional vector:

$$\begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$$

  - Obviously, for deterministic circuits where the wire state is 1, $p_0 = 0$ and $p_1 = 1$.

An algebra for probabilistic circuits

- Let's take as an example a circuit for the NOT operation.
- In this algebra, we would write:

$$\text{NOT} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  and vice-versa.
- If the wire state is 1, in which case $p_1 = 1$, the NOT operation would produce as output the right hand matrix.

An algebra for probabilistic circuits

- Therefore, the NOT circuit can be represented by:

$$NOT \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

An algebra for probabilistic circuits

- To apply the gate to a wire in a given state:

$$\text{NOT} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} p_1 \\ p_0 \end{pmatrix}$$

An algebra for probabilistic circuits

- Assume now that you have to describe the state associated with a given point in a probabilistic circuit having two wires.
- The probabilities of 0 and 1 of the first wire are p_0 and p_1, respectively.
- The probabilities of 0 and 1 of the second wire are q_0 and q_1, respectively.
- Then the probabilities associated with each of the possible four states are:

$$prob(ij) = p_i q_j$$

with $i = 0, 1$ and $j = 0, 1$, and ij one of the possible combined states of the two wires.

An algebra for probabilistic circuits

- The combined states of both wires is the **tensor product** of the 2-dimensional vectors for the states of the first and second wires separately:

$$\begin{pmatrix} p_0 q_o \\ p_0 q_1 \\ p_1 q_0 \\ p_1 q_1 \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} \otimes \begin{pmatrix} q_0 \\ q_1 \end{pmatrix}$$

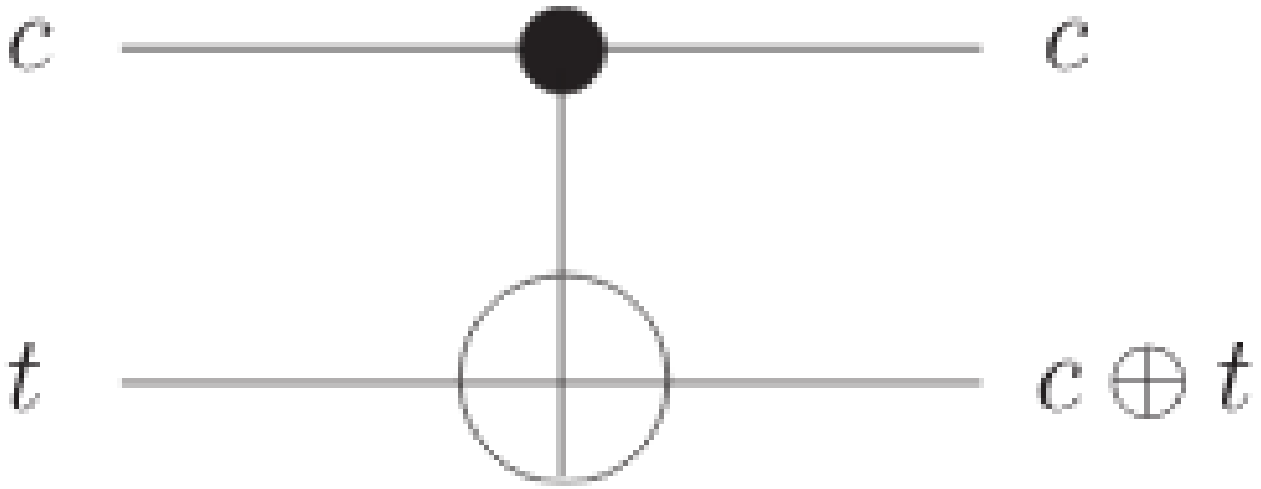
Example: the CNOT gate

- We can represent gates applied to more than one wire.
- As an example, we can use the **Controlled NOT** (CNOT).
- The CNOT gate acts on two bits, the **control** bit and the **target** bit.
- Its action is to apply the NOT operation to the target if the control bit is 1, and do nothing otherwise.
- In contrast to a classical computer gate, CNOT can implement multiple operations: a NOT or an exclusive OR (XOR) depending on the qubit states.

Example: the CNOT gate

- CNOT is represented as follows (operation in two wires):

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Example: the CNOT gate

- The control bit is always unaffected by the CNOT gate.
- If we fix the value of the control bit to $c = 0$ or to $c = 1$, CNOT maps the target bit to:

$$t \oplus c$$

with $\oplus$ representing the logical exclusive-or operation, or addition modulo 2.

- In other words, the CNOT transforms the quantum state:

$$\text{CNOT}(a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle) = a\,|00\rangle + b\,|01\rangle + c\,|11\rangle + d\,|10\rangle$$

Example: the CNOT gate

- Example with control bit in state 1 and target bit in state 0.
- The 4-dimensional state vector is given by:

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

- Applying the CNOT to this vector produces exactly the desired behaviour:

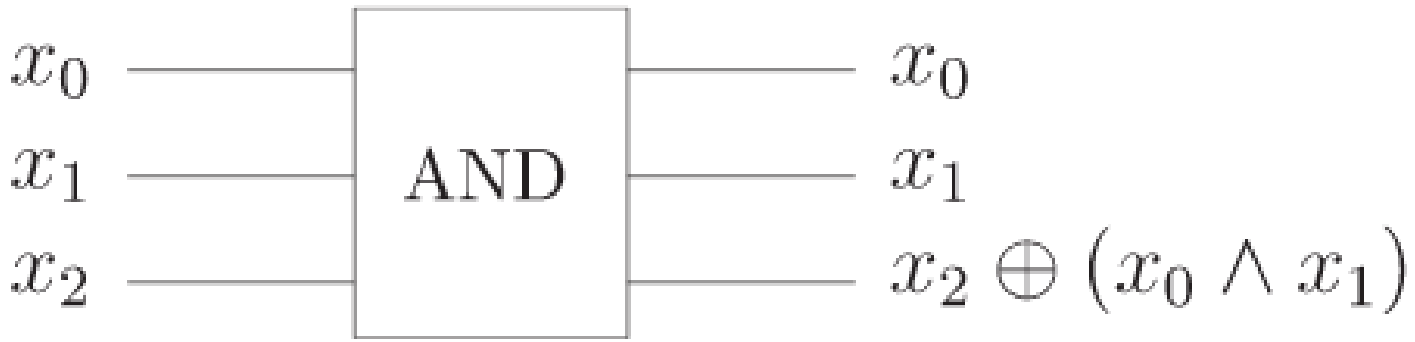
$$\text{CNOT} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Entanglement!

- If the first bit is in state: $\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$, and the second is in state: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- Applying CNOT will produce: $\begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix}$
- which can not be factorized into the tensor product of two independent probabilistic bits.
- In other words: The states of two such bits are correlated (**entangled**).

Reversible Computation!

- **Any** (generally irreversible) computation can be transformed into a reversible computation.
- It suffices to add some additional input and output wires, if necessary.
- Example: reversible AND



Reversible Computation: AND

- The reversible AND corresponds to the **Toffoli** quantum gate.
- It is a generalization of CNOT, where instead of having just one control bit, we have two bits that control if the third bit will be flipped.
- In contrast to the classical AND gate, Toffoli adds one extra input and two extra outputs (copies of the two inputs).
- By fixing the additional input bit to 0 and discarding the copies of x_0 and x_1 we can simulate the non-reversible AND gate.
- In general, if we have an irreversible circuit with depth T and space S, it is possible to construct a reversible version that uses a total of $O(S + ST)$ space and depth T.

Quantum hardware

- Quantum machines use technologies other than gates.
- For example, some hardware are built to support the development of applications that involve multiple solutions where one wants to find the best or near-optimal.
- These are called **quantum annealing** machines.
- In this set of lectures we focus on the **gate-based quantum model**.
- Currently, the available quantum models are:
  - Gate-based
  - Adiabatic
  - Topological
  - Measurement-based

Quantum gates

- Quantum gates/operations are represented as matrices which act on a qubit represented by a 2×2 unitary matrix U.
- As seen before, the operation is to multiply the matrix representing the gate with the vector that represents the quantum state.

$$|\psi\rangle = U\,|\psi\rangle$$

- 2-qubit gates are 4×4 matrices.
- 3-qubit gates are 8×8 matrices.

Examples of quantum gates

1-qubit gates (Pauli gates):
(σ_x already shown: NOT)

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_h = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

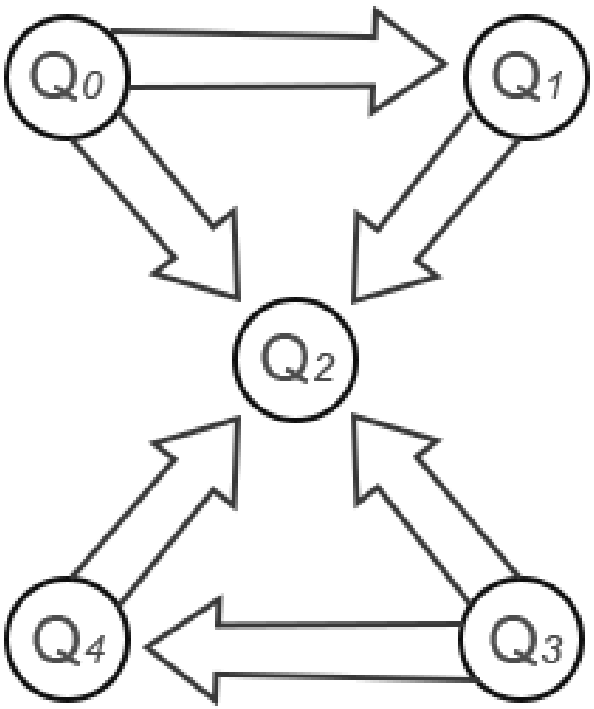
$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

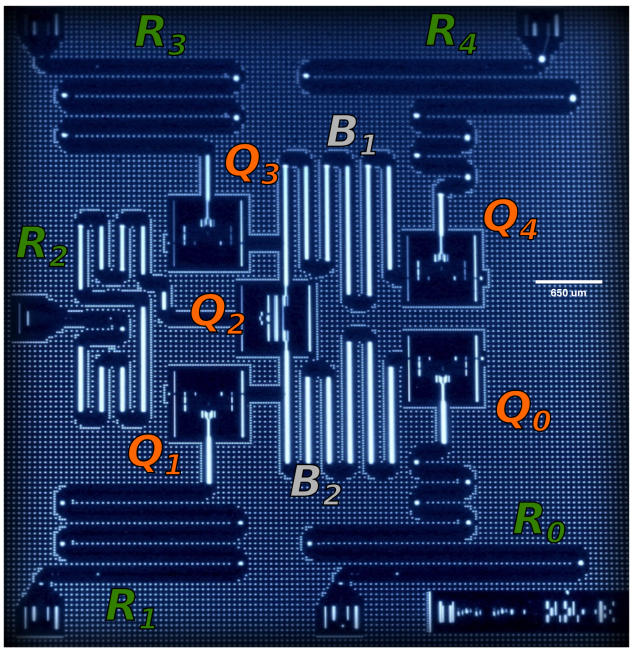
Quantum hardware: qubit connections

- qubits need to be connected in order to implement operations.
- Not all possible connections are implemented:
  - for physical reasons (interference of one qubit with another)
  - to prevent cycles

Quantum hardware: qubit connections

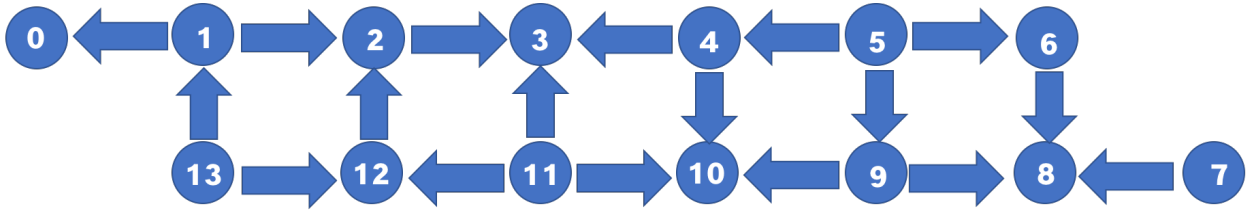
- Example of a qubit topology and machine (IBM ibmqx2):

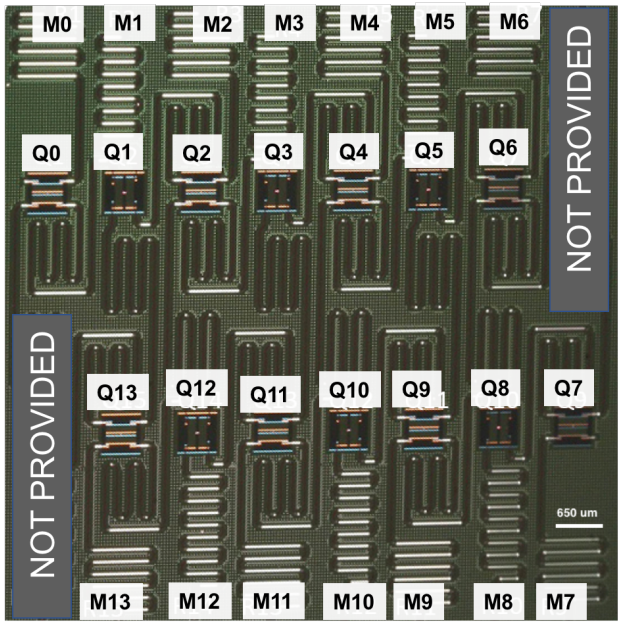




Quantum hardware: qubit connections

- Another example: the Melbourne machine with 20 qubits





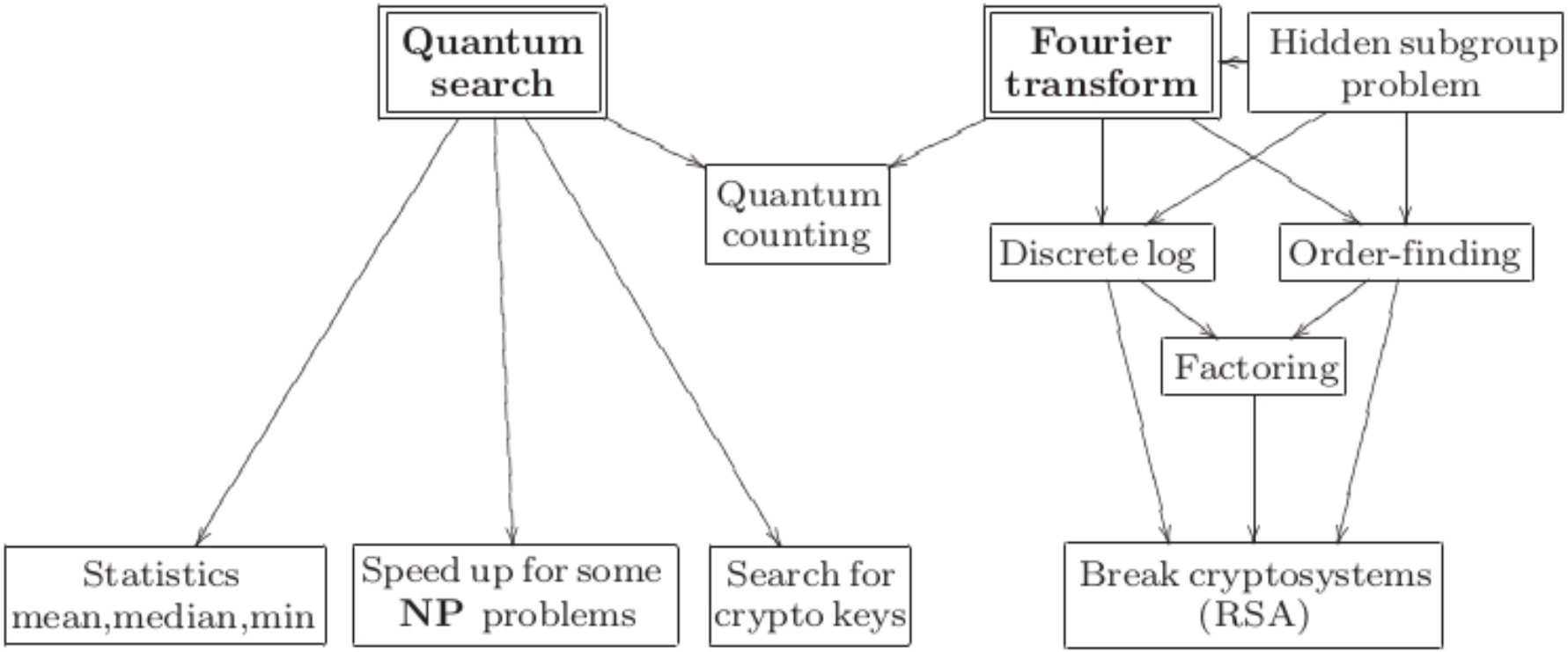
Quantum hardware: qubit connections

- When programming a quantum machine, we compose gates.
- When composing gates, we need to connect qubits.
- Because of the topology of each quantum hardware, it is not possible (or ideal) choose any qubit connection.
- For example, in the Melbourne machine, one shouldn't try to make connections directed from qubits q0 to q1, but the other way around.

Quantum hardware: qubit connections

- In the IBM quantum machines, the compiler handles this issue and transforms the user program in order to make a better utilization of the machine's topology.
- Experiments show that circuits with various mappings can solve the same problem.
- Research is under way to improve the compiler algorithms.

Quantum Algorithms



Algorithm categorization taken from Quantum Computation and Quantum Information, by Nielsen Page and Chuang.

Quantum Algorithms

According to Nielsen and Chuang, quantum algorithms can be divided in two main types:

- Algorithms based on Shor's quantum Fourier transform
  - some of the most popular known algorithms are built on top of Shor's with a exponencial speedup over their classical counterparts.
    - the classic Fast Fourier transform runs on $O(n \log n)$ time and the Quantum Fourier transform runs on $O(\log^2 n)$
- Algorithms based on Grover's, which is mainly used on quantum search. This method gives a quadratic speedup to some of the best classical algorithms.
  - if the classical algorithm runs on the worst case on $O(N)$, Grover's search algorithm can be improved to run on $O(\sqrt{N})$

Quantum Complexity

- In classical computing there are two main complexity classes:
  - P: class of problems that can be solved by an algorithm in polynomial time.
  - NP: class of problems to which it is unknown if there are solutions in polynomial time.
- In quantum computing there are equivalent complexity classes:
  - **BQP**: Bounded-error Quantum Polynomial time which is the class of decision problems that can be solved by a quantum computer in polynomial time.
  - **QMA**: Quantum Merlin Arthur, which is, in an informal way, the set of decision problems for which when the answer is YES, there is a polynomial-size quantum proof (a quantum state) which convinces a polynomial-time quantum verifier of the fact with high probability. Moreover, when the answer is NO, every polynomial-size quantum state is rejected by the verifier with high probability.

IBM Quantum Experience

- IBM Q Experience device information
- What is QISKit
- Install QISKit using PIP Install Command
- Single-Qubit Gates
- Two-Qubit Gates
- Three-Qubit Gates
- Hello Quantum World program
- Quantum (not)Smiley program

IBM Quantum Experience devices

- IBM Q devices are named after IBM office locations around the globe.
  - Client: IBM Q 20 Tokyo (20 qubits)
  - Public devices:
    - IBM Q 14 Melbourne (14 qubits)
    - IBM Q 5 Tenerife (5 qubits)
    - IBM Q 5 Yorktown (5 qubits)
  - Simulators: IBM Q QASM 32 Q Simulator (32 qubits).

IBM Quantum Experience devices

- display_name: IBM Q 5 Yorktown
- backend_name: ibmqx2
- description: the connectivity is provided by two coplanar waveguide (CPW) resonators with resonances around:
  - 6.6 GHz (coupling Q2, Q3 and Q4)
  - 7.0 GHz (coupling Q0, Q1 and Q2)
- Each qubit has a dedicated CPW for control and readout (labeled as R).

IBM Quantum Experience devices

- display_name: IBM Q 16 Melbourne
- backend_name: ibmq_16_melbourne
- description: the connectivity on the device is provided by total 22 coplanar waveguide (CPW) bus resonators, each of which connects two qubits.
- Three different resonant frequencies are used for the bus resonators: 6.25 GHz, 6.45 GHz, and 6.65 GHz.

IBM Quantum Experience: QISKIT

- QISKit: software development kit (SDK)
- Python-based software libraries used to:
  - create
  - compile and
  - execute...
  - ...quantum computing programs in the processors backends and simulators available in the IBM cloud.

IBM Quantum Experience: QISKIT

- QISKit software stack is composed of various layers.
- **Terra** contains a set of tools for:
  - composing quantum programs at the level of circuits and pulses
  - optimizing them for the constraints of a particular physical quantum processor
  - managing the batched execution of experiments on remote-access backends.

IBM Quantum Experience: QISKIT

- **Acqua** contains a library of cross-domain quantum algorithms upon which applications for near-term quantum computing can be built.
- Aqua is designed to be extensible, and employs a pluggable framework where quantum algorithms can easily be added.
- It currently allows the user to experiment on chemistry, AI, optimization and finance applications.

Installing QISKIT

- Requirements: python 3.x or over, python3-pip.
- Python matplotlib
- To install QISKIT:
  - `pip3 install qiskit` or
  - `pip install qiskit`

Using QISKIT

Useful packages to run experiments:

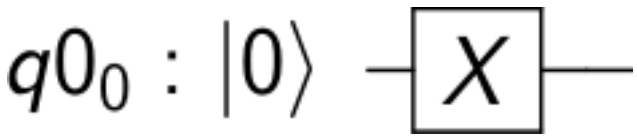
```
1 import numpy as np
2 from qiskit import QuantumCircuit, QuantumRegister
3 from qiskit import execute
4 from qiskit.tools.visualization import circuit_drawer
5 from qiskit import Aer
6 backend = Aer.get_backend('unitary_simulator')
```

Gates

- Single-Qubit Gates:
  - u gates
  - Identity gate
  - Pauli gates
  - Cliffords gates
  - $C3$ gates
  - Standard rotation gates

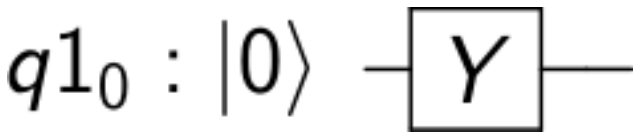
Pauli X

```
# Pauli X : bit-flip gate
q = QuantumRegister(1)
qc = QuantumCircuit(q)
qc.x(q)
im = circuit_drawer(qc)
im.save("GateX.png", "PNG")
job = execute(qc, backend)
print(np.round(job.result().get_data(qc)['unitary'], 3))
```



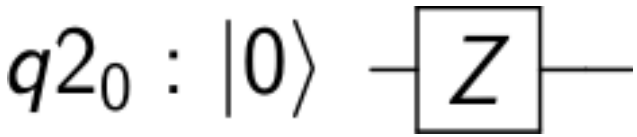
Pauli Y

```
1 # Pauli Y : bit and phase-flip gate
2 q = QuantumRegister(1)
3 qc = QuantumCircuit(q)
4 qc.y(q)
5 im = circuit_drawer(qc)
6 im.save("GateY.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```



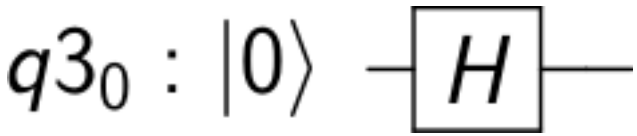
Pauli Z

```
1 # Pauli Z : phase-flip gate
2 Q = QuantumRegister(1)
3 qc = QuantumCircuit(q)
4 qc.z(q)
5 im = circuit_drawer(qc)
6 im.save("GateZ.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```



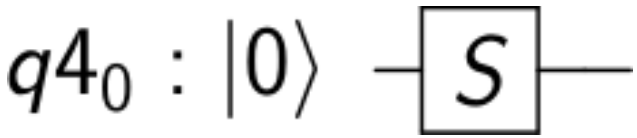
Hadamard

```
1 # Clifford Hadamard gate
2 q = QuantumRegister(1)
3 qc = QuantumCircuit(q)
4 qc.h(q)
5 im = circuit_drawer(qc)
6 im.save("GateH.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))"
```



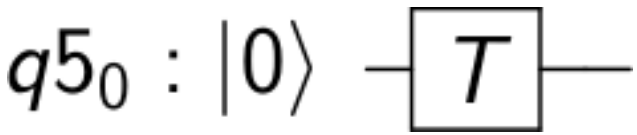
Clifford S

```
1 # Clifford S gate,
2 q = QuantumRegister(1),
3 qc = QuantumCircuit(q),
4 qc.s(q),
5 im = circuit_drawer(qc),
6 im.save("GateS.png", "PNG"),
7 job = execute(qc, backend),
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```



Clifford T

```
# Clifford T gate,
q = QuantumRegister(1),
qc = QuantumCircuit(q),
qc.t(q),
im = circuit_drawer(qc),
im.save("GateT.png", "PNG"),
job = execute(qc, backend),
print(np.round(job.result().get_data(qc)['unitary'], 3))"
```

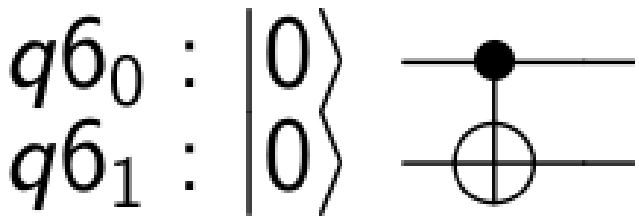


Gates

- Two-Qubit Gates:
  - controlled Pauli gates
  - controlled Hadamard gate
  - controlled rotation gates
  - controlled phase gate
  - controlled u3 gate
  - swap gate

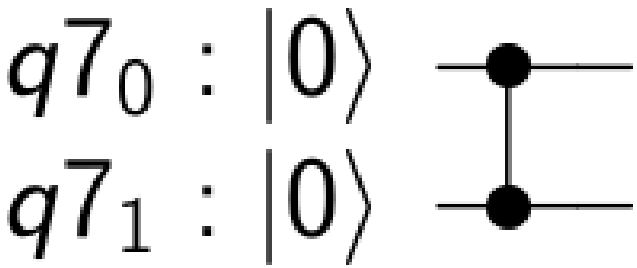
Pauli Controlled-X (or, controlled-NOT)

```
1 # Pauli Controlled-X (or, controlled-NOT) gate
2 q = QuantumRegister(2)
3 qc = QuantumCircuit(q)
4 qc.cx(q)
5 im = circuit_drawer(qc)
6 im.save("GateCX.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```



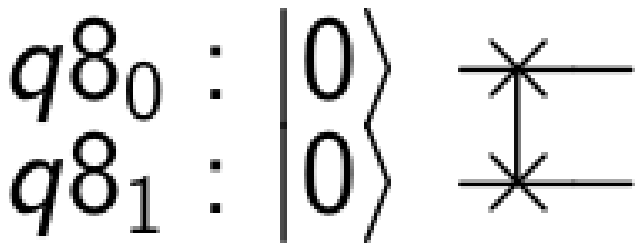
Pauli Controlled-Z

```
1 # Pauli Controlled Z (or, controlled Phase) gate
2 q = QuantumRegister(2)
3 qc = QuantumCircuit(q)
4 qc.cz(q)
5 im = circuit_drawer(qc)
6 im.save("GateCZ.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```



Swap

```
1 # SWAP gate
2 q = QuantumRegister(2)
3 qc = QuantumCircuit(q)
4 qc.swap(q)
5 im = circuit_drawer(qc)
6 im.save("GateSWAP.png", "PNG")
7 job = execute(qc, backend)
8 print(np.round(job.result().get_data(qc)['unitary'], 3))
```

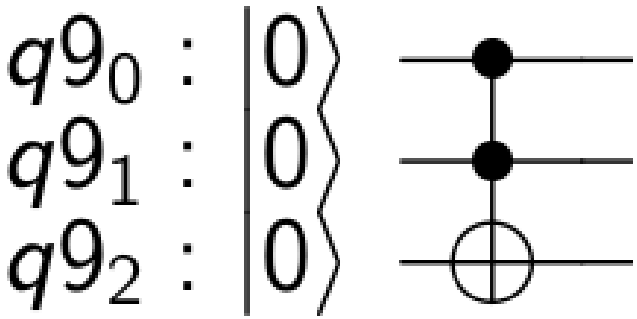


Gates

- Three-Qubit Gates:
  - Toffoli gate
  - Fredkin gate

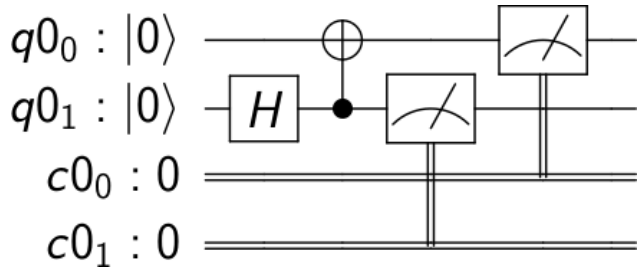
Toffoli gate (ccx gate)

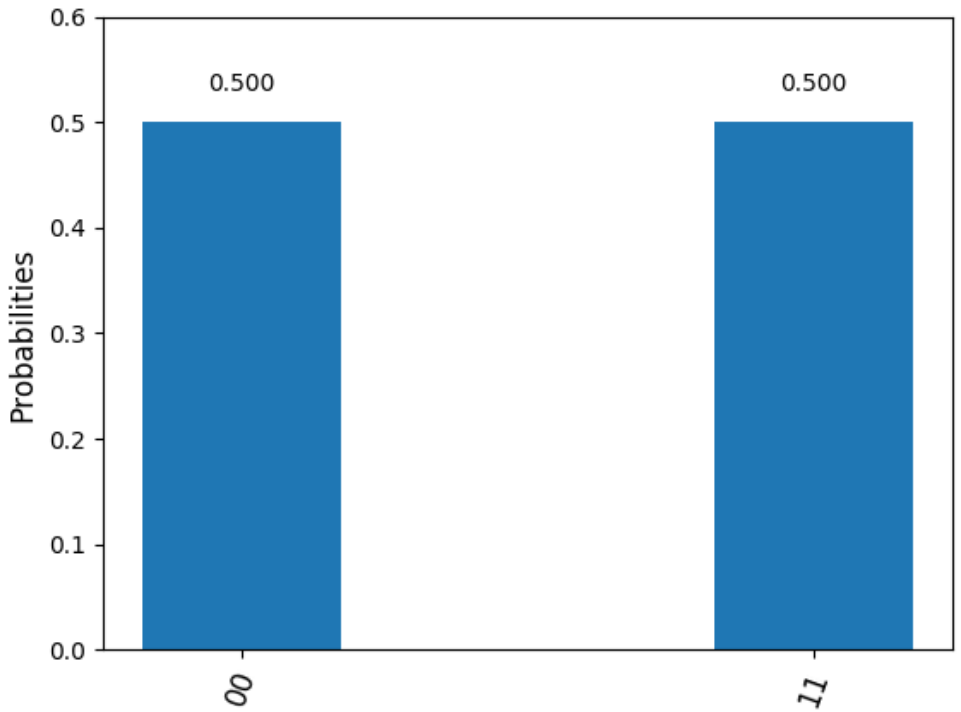
```
# Toffoli gate (ccx gate),
q = QuantumRegister(3),
qc = QuantumCircuit(q),
qc.ccx(q),
im = circuit_drawer(qc),
im.save("GateCCX.png", "PNG"),
job = execute(qc, backend),
print(np.round(job.result().get_data(qc)['unitary'], 3))
```



Writing your first program :)

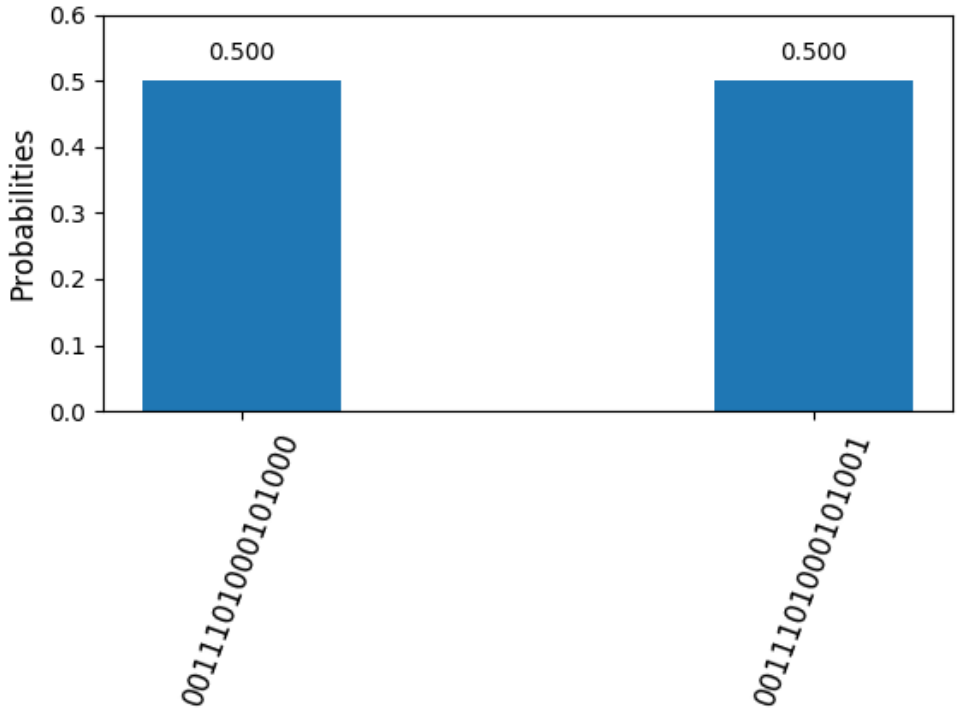
- Quantum "Hello World!"





Smiley :)(

- (not)Smiley"



More examples: mapping classical to quantum

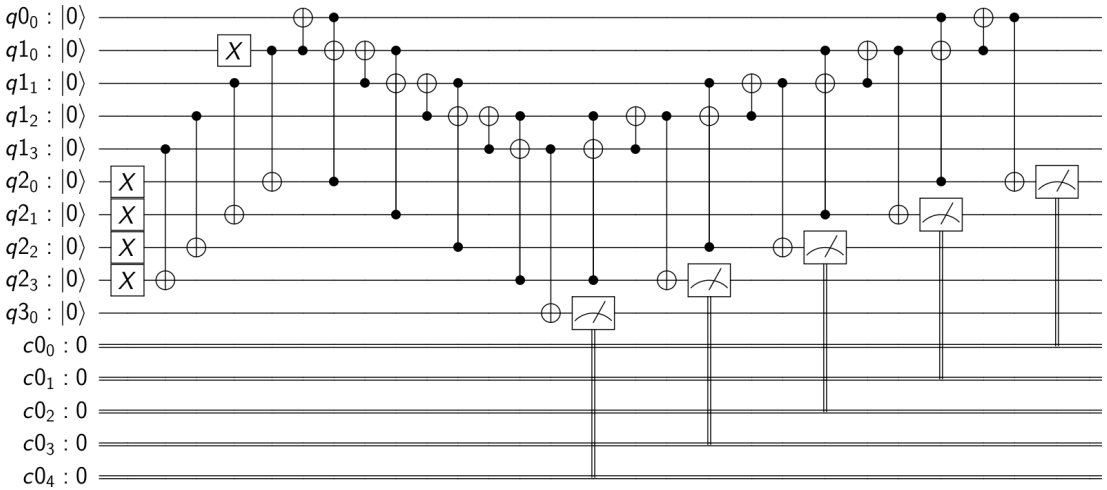
- Assignment
- Arithmetic
  - Sum (Ripple-carry adder)
  - Sum (Quantum Fourier Transform)
  - Average

Assignment

- Quantum Assignment program
- Registers are qubits, therefore we cannot directly assign values from the bit strings to a quantum register.
- As such, when we create a new quantum register, all of its qubits start in the $|0\rangle$ state. By using the quantum X gate, we can flip the qubits that give us the $|1\rangle$ state.

Sum (Ripple-carry adder)

- Quantum adder (1)
- This implementation prepares a $= 1$ and b $= 15$ and computes the sum into b with an output carry cout[0] by using a quantum ripple-carry adder approach.
- Of course, your final result should be 16!
- Here it is the resulting circuit:



Sum (Quantum Fourier Transform)

- Quantum adder (2)
- calculate sum on a quantum computer using the quantum Fourier Transform by applying repeated controlled-U gates.
- Unfortunately, the resulting circuit is too large, but you can try clicking here to see it.
- Contrast this quantum version of the adder with the classical one.

Average

- In a two-qubit system we represent the kets as vectors, the Pauli as matrices and perform the tensor products in order to implement the average.

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We calculate the tensor product, such as:

$$|01\rangle = |0\rangle \otimes |1\rangle \equiv \begin{pmatrix} 1 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

where $\langle 01|$ is the Hermitian conjugate of

$$\langle 01| = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$$

Average

- We use the same reasoning for the operators:

$$X_1X_2 = \sigma_1 \otimes \sigma_2 = \begin{pmatrix} 0 \times \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 1 \times \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ 1 \times \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 0 \times \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

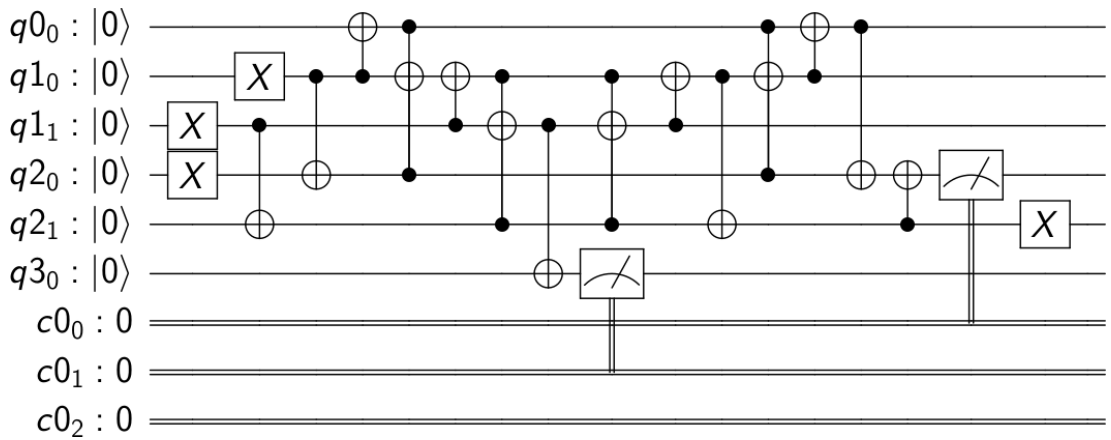
Average

- Finally, we multiply!

$$\langle 01| X_1 X_2 |01\rangle = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} = 0$$

Average

- Average between 2 numbers
- Here it is the resulting circuit:



Search: Grover algorithm

- Grover's Search
- This algorith is mainly used to search databases.
- As mentioned before, the time it runs improves quadratically compared to the classical version.

Search: Grover algorithm

- Given an array of N elements, the goal is to locate a specific and unique element.
- How does it work to improve complexity?
  - The Oracle:
    - encode a list using a function f, where $f(x) = 0$ for all unwanted positions and $f(u) = 1$ for the element u we want to locate.
    - define the Oracle to act on the state $|x\rangle$ by applying the oracle matrix U_f: $U_f |x\rangle = (-1)^{f(x)} |x\rangle$.
    - This works because the state of the qbit only changes when we find $f(u) = 1$ resulting in $U_f |u\rangle = -|u\rangle$ while the rest remains the same.

Search: Grover algorithm

- Amplitude amplification
  1. Start a uniform superposition $|s\rangle$ by doing the following $|s\rangle = H \otimes n\,|0\rangle\, n$.
  2.
  3. We then have to apply the oracle function $U_f\,||\psi t\rangle = |\psi t'\rangle$. This results on the amplitude of the element we are searching for turning negative.
  4. We now apply an additional reflection U_s on the state $|s\rangle$.
  5. This process is repeated $O(\sqrt{N})$.

Using Grover to implement exactly-1 solution 3-SAT

- The code implements a deterministic solution for a SAT problem with 3 clauses and 3 literals that has exacly one true output, the formula has to be on conjunctive normal form(CNF) in order for it to work.
- This works the following way, imagine all the results the expression could give are in an array, since the expression only has one output that is true, being that the one we want to find, it is possible to apply the algorithm to this problem.

Shor Algortihm

- Any integer number has a unique decomposition into a product of prime numbers.
- In 1995, Peter Shor proposed a polynomial-time quantum algorithm for the factoring problem.
- Runs partially on quantum computer: Fourier Transform and Period-finding algorithm.
- Pre- and post-processing on a classical computer:
  - greatest common divisor (quadratic in number of digits of a, b)
  - test of primality (polynomial) prime power test ($O(\log n)$)
  - continued fraction expansion (polynomial)
- Worst Case in Quantum: polynomial in $\log N$
- Worst Case in Classical: polynomial in N

Shor Algortihm

- Period Finding
  - Given integers N and a find the smallest positive integer r such that $a^r - 1$ is a multiple of N, where r is the period of $a \mod N$ (smallest positive integer r such that $a^r = 1 \mod N$)
  - Example: $N = 15$ and $a = 7$
  - find the smallest positive integer r such that $a^r = 1 \mod N \Leftrightarrow 7^r = 1 \mod 15$

Shor Algortihm

- From factoring to period finding
  - Assume that N has only two distinct prime factors: $N = p1 * p2$
    1. pick an integer a between $[2, N-1]$ and compute greatest common divisor $\gcd(N, a)$ using Euclid's Algorithm
    2. If $\gcd(N, a) = p1$ or $p2$ we are done.
    3. Else repeat the above steps with different random choices of a until r is even.
    4. If r is even and is the smallest integer such that $a^r - 1$ is a multiple of N: $a^r - 1 = (a^{r/2} - 1) * (a^{r/2} + 1)$, where neither $(a^{r/2} - 1)$ nor $(a^{r/2} + 1)$ is a multiple of N, but their product is
    5. We can find $p1$ and $p2$ by computing $\gcd(N, (a^{r/2} - 1))$ and $\gcd(N, (a^{r/2} + 1))$
    6. If $(a^{r/2} + 1)$ is multiple of N we try with a different a.

Shor Algortihm

- At the heart of Shor's Algorithm is the Quantum Fourier Transform (QFT), a linear transformation on quantum bits, and the quantum analogue of the discrete Fourier transform.
- Formally, the QFT is expected to be useful in determining the period r of a function i.e. when $f(x) = f(x + r)$ for all x.
- The input register for QFT contains n-qubit basis state $|x\rangle$ which is rewritten as the tensor product of the individual qubits in its binary expansion.

Shor Algortihm

- Each of those single-qubit operations can be implemented efficiently using a Hadamard gate and controlled phase gates.
- The first term requires one Hadamard gate and $(n-1)$ controlled phase gates, the next one requires a Hadamard gate and $(n-2)$ controlled phase gate, and each following term requires one fewer controlled phase gate.
- More info at: `http://demonstrations.wolfram.com/QuantumFourierTransformCircuit/`
- Code: `https://github.com/Qiskit/openqasm/blob/master/examples/generic/qft.qasm`

Shor Algortihm

- Shor's algorithm exploits interference to measure periodicity of arithmetic objects.
- Suppose a, N are co-primes, $gcd(a, N) = 1$.
- Our goal is to compute the smallest positive integer r such that $a^r = 1 \mod N$
  1. Determine if n is even, prime or a prime power. If so, exit.
  2. Pick a random integer $x < n$ and calculate $\gcd(x, n)$. If this is not 1, then we have obtained a factor of n.
  3. Quantum algorithm:
     - Pick q as the smallest power of 2 with $n^2 \leq q < 2 * n^2$.
     - Find period r of $x^a \mod n$.
     - Measurement gives us a variable c which has the property $c \approx d$ where $d \in| N$.
  4. Determine d, r via continued fraction expansion algorithm. d, r only determined if $\gcd(d, r) = 1$ (reduced fraction).
  5. If r is odd, go back to 2.
  6. If $x^(r/2) = -1 \mod n$ go back to 2.
  7. Otherwise the factors $p, q = \gcd(x^(r/2) + / - 1, n)$.

Challenges

- Improve algorithms for better coupling map.
- Better understanding of how to map classical problems to quantum platforms.
- Higher level programming language.
- More qubits.

Challenges: coupling map using tabu search

- Regarding coupling maps we have contributed with a tabu search implementation for the generation of alternative couplings.
- The algorithm main steps can be briefly described by:
  1. generating the neighbor solutions (one alternative coupling map)
  2. evaluating each neighbor
  3. getting the neighbor with maximum evaluation.
  4. The algorithm stops at any iteration where there are no feasible moves into the local neighborhood of the current solution.

Challenges: coupling map using tabu search

Result: best solution, evaluation, best iteration

```
initialize quantum population Q(t)
while True do
    if (iteration − bestiteration) > maxquantity then
        if bestsolution[0] ≠ bestsolution[1] then
            apply CNOT gate
        else
            apply H gate
        end
    end
    new neighborhood generation and evaluation
    if neighborsevaluation > bestevaluation then
        keep tabu move
        new better solution s^b
        update Q(t) best evaluation e^b
        increment best iteration i^b
    end
end
```

Algorithm 1: General QTS algorithm

Final Remarks

- Check Quantum Fourier Transform, Grover's and Shor's algorithms.
- Check 'from qiskit import available_backends', useful to know which real backends are available.
- Check 'from qiskit.tools.qi.qi import state_fidelity', tool to verify if two states are same or not.
- Check 'from qiskit.tools.visualization import...', tools for exceptional visualization charts!
- For any help check the IBM Q Graphical composer :)
  - Compose quantum circuits using drag and drop interface.
  - Save circuits online or as QASM text, and import later.
  - Run circuits on real hardware and simulator.

Happy Quantum Coding!

Bibliography

1. Kaye, P., Laflamme, R. & Mosca, M. (2007). An Introduction to Quantum Computing, Oxford University Press.
2. Nielsen, M., & Chuang, I. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511976667.
3. IBM Research and the IBM QX team (2017). User guide. IBM Q Experience Documentation https://quantumexperience.ng.bluemix.net/qx/tutorial
4. Cuccaro, S., Draper, T., Kutin, S. & Moulton, D. (2004). A new quantum ripple-carry addition circuit. arXiv:quant-ph/0410184.
5. Cross, A. W., Bishop, L. S., Smolin, J. A. & Gambetta, J. M. (2017). Open quantum assembly language. arXiv:1707.03429.
6. Nielsen Page, M. A. & Chuang, I. L. Quantum Computation and Quantum Information. (2010). Cambridge University Press.

Bibliography

1. 5-qubit backend: IBM Q team, "IBM Q 5 Yorktown backend specification V1.1.0," (2018). Retrieved from https://ibm.biz/qiskit-yorktown.
2. 14-qubit backend: IBM Q team, "IBM Q 14 Melbourne backend specification V1.x.x," (2018). Retrieved from https://ibm.biz/qiskit-melbourne.¡br¿
3. Silva, C., Dutra, I., & Dahlem M.S. (2018). Driven tabu search: a quantum inherent optimisation. arXiv preprint arXiv:1808.08429.
4. Ruiz-Perez L. & Garcia-Escartin, J. C. (2014). Quantum arithmetic with the Quantum Fourier Transform. arXiv preprint arXiv:1411.5949.
5. Ferrari, D. & Amoretti, M. (2018). Demonstration of Envariance and Parity Learning on the IBM 16 Qubit Processor. arXiv:1801.02363v2 [quant-ph] 7 Feb 2018.
6. Zulehner, A., Paler, A. & Wille, R. (2017). An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. arXiv:1712.04722v3 [quant-ph] 7 Jun 2018.