# EMERGING TRENDS

## Quantum Sudoku Solver

### Main Research Question
Can I create a performant sudoku solver using Qiskit and Grover's algorithm
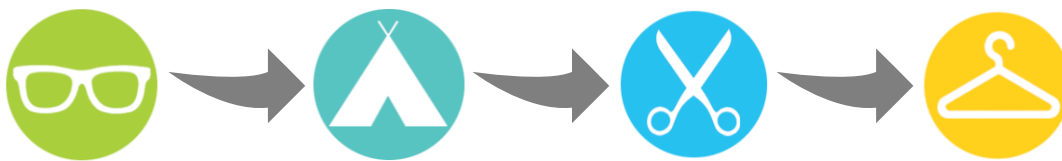
Luka Spaninks

## Preface

In order to pass this semester of school I have to pick one of several emerging trends and write a research report about it. I chose quantum computing because it was the subject I know the least about. I hope this research helps me get into quantum computing and gives me an advantage in the future. This document partially contains theoretical research and partially applied research.

## Research Strategy

I use the DOT framework in order to conduct research on the questions. I start off by doing some library research on the basics of quantum computing. Then I do some documentation research and look at the product I am interested in (Qiskit). Afterwards I start prototyping and eventually apply a benchmark test.

# Table of contents

# 1. What are Qubits? 🤓

After having done some shallow research on how to build a sudoku solver a lot of papers mentioned qubits quite a few time. Knowing what qubits are is essential for starting with quantum computing. That's why I have chosen to first learn about qubits.

A qubit is short for quantum bit and is comparable to a bit in normal computing. A major difference however is that a qubit can hold a 0, 1 or even both. The reason that classical binary digits do not work within quantum computing is that they are not stable. This has to do with the quantum realm having different physics. Qubits can use a light particle (photon) or a spin in a magnetic field as a state to store information. As long as a qubit is unobserved, it is stateless. Scientists call this superposition. This means that it is not possible to predict the exact value of the qubit until it is measured. The advantage of this is that a series of qubits can contain a lot of configurations at once. As you can see below a qubit state is generally visually represented in a 3d sphere (Bloch sphere). The angle represents the probability of a qubit being either close to 0 or 1.
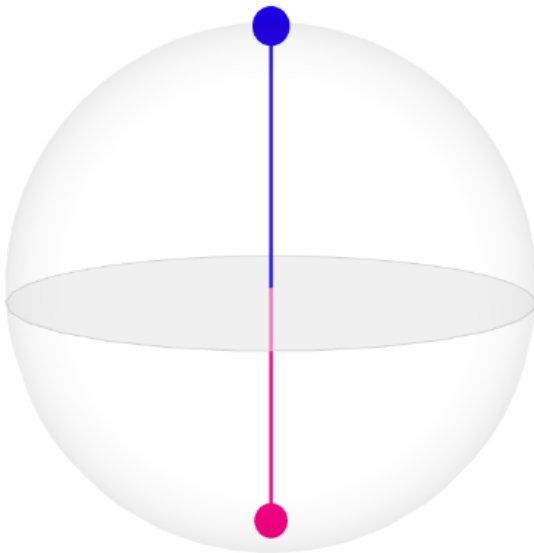


*Qubit state represented as a 3d sphere (up is 0, down is 1)*

In quantum computing, multiple qubits can display quantum entanglement. This means that these qubits have a high correlation with another. It is possible for two particles to link together in a certain way no matter how far apart they are in space. Their state remains the same.

Because of the way qubits work quantum is only more efficient in a handful of applications. Examples of these applications are simulation, AI, Drug development & Cybersecurity. This is because the advantage of quantum computing is parallel processing, this can theoretically make it exponentially faster than a normal processing unit.

## 2. What is Grover's algorithm? 🤓

Within quantum computing, several algorithms can be used for different applications. Shor's algorithm can for example be used for integer factorization. Using this algorithm should be significantly faster than using any classic algorithm.

Grover's algorithm was devised by Lov Grover in 1996 and is also commonly referred to as the quantum search algorithm. In order to build a sudoku solver we need a fast algorithm that can try a lot of possibilities at once. This is similar to brute forcing. I believe Grover's algorithm can help out.

One of the main uses case of Grover's algorithm is searching a database. The way I want to use it is to solve a NP-complete problem. A problem is NP-complete when the solution can be verified quickly and can be found by a brute-force search. Also, the problem can be used to simulate every other problem for which we can verify quickly that a solution is correct. Both of these requirements are met by a classic sudoku problem.

Using a classical algorithm an unstructured search could be solved in the following amount of time: *O(N)*. The O describes how time scales with the variables and N describes the number of entries. It takes on average *n*/2 values to find it. However, Grover's algorithm can speed this up by : $O(\sqrt{n})$. This means that Grover's algorithm makes it possible for linear algorithms to be sped up quadratically. This should result in a performance boost in comparison to classical computing. Even though it is not an exponential speed up, it will still be very noticeable in bigger datasets.

Similar to other quantum algorithms, Grover's algorithm is probabilistic. This means that there is a high probability it will give the correct outcome. The chance of failure can be decreased by repeating the algorithm.

## 3. What are applications of Grover's algorithm? 🤓

As already mentioned, Grover's algorithm is generally used for searching a database. But it also has several other use cases in cryptography. It essentially is able to do a function inversion which makes it useful for finding an unknown value.

Grover's algorithm can easily be used in a brute-force attack. A 128-bit symmetric cryptographic key can be brute-forced in 264 iterations and a 256-bit key in 2128. That's why security experts often recommend that symmetric key lengths should be doubled to protect against future quantum attacks.

Grover's algorithm can also break a cryptographic hash function by performing a collision attack and a preimage attack. A collision attack tries to find two inputs producing the same hash value. This is different from a preimage attack where a specific target hash value is defined.
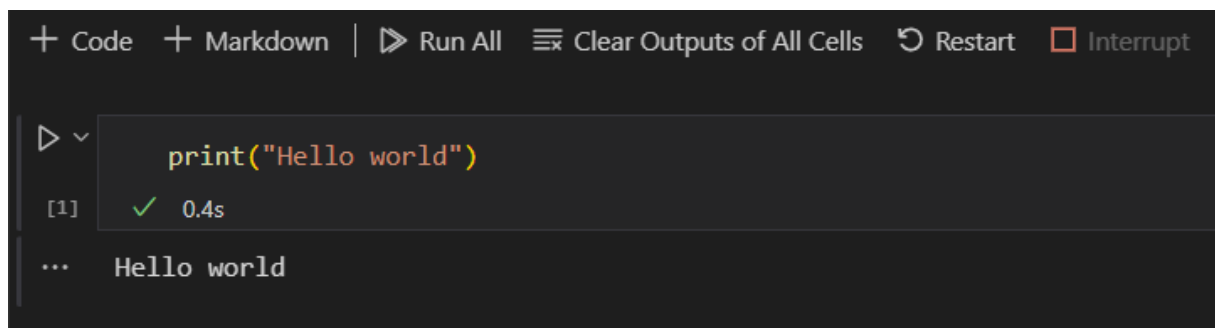
As already mentioned, Grover's algorithm is also generally used to speed up algorithms. This is called the amplitude amplification trick. This trick stretches out the amplitude (the distance between a particle wave's center and its top**)** of the marked item, which shrinks the other items' amplitude, so that measuring the final state will have a much higher chance of returning the right item.

# 4. How do I install and use Qiskit on my machine? ⏶

In order to start programming a quantum sudoku solver I need to install Qiskit on my machine. I will be using the official documentation throughout the process. I have the option to start on the cloud or locally. I will be installing it locally because I am already familiar with the environment.

First of all I make sure I have python 3.7 (or later) installed on my machine. It is also recommended that I install Anaconda which comes with Jupyter notebooks.
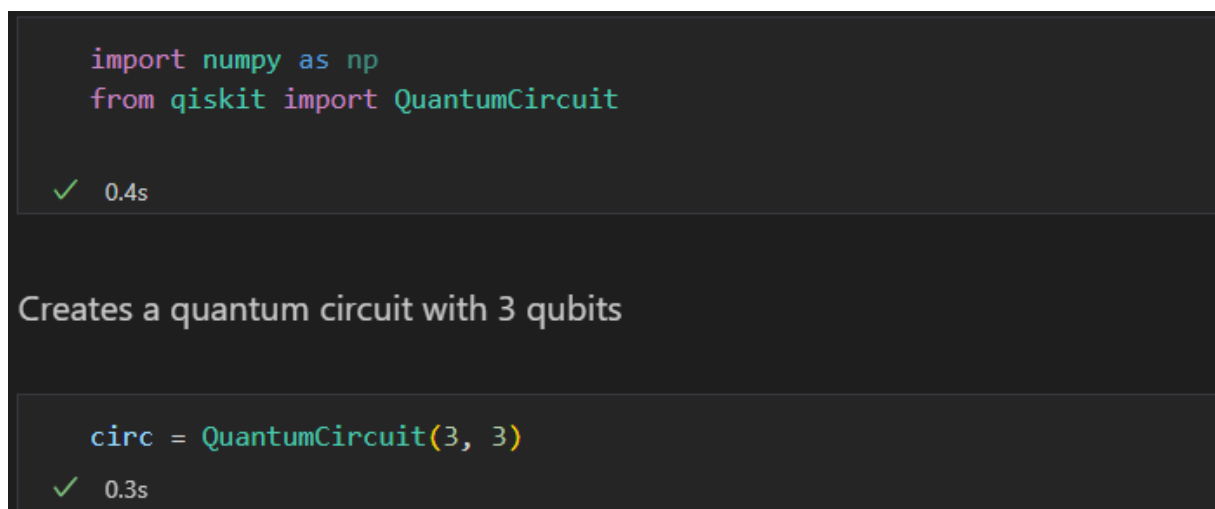
Next I create a new anaconda environment and install Qiskit with pip package manager. Now I can create a Jupyter notebook and open it in visual code. Now that the document has been created am I able to run python code in cells.

```
+ Code  + Markdown  |  ▷ Run All  ≡x Clear Outputs of All Cells  ↻ Restart  ☐ Interrupt

▷ ∨        print("Hello world")
[1]    ✓  0.4s

...     Hello world
```

In order to get a better understanding of Quantum programming I am going to follow a tutorial on Quantum circuits. This can also be found on the Qiskit website.

Next I create 1 cell doing the imports and another to create a quantum circuit containing 3 qubits.

```
    import numpy as np
    from qiskit import QuantumCircuit


✓  0.4s
```

Creates a quantum circuit with 3 qubits

```
    circ = QuantumCircuit(3, 3)
✓  0.3s
```

Now I apply the Hadamard gate to the first qubit. The Hadamard puts a qubit in superposition meaning that the value will be either 0 or 1 until it's observed. Next I apply two not gates to the second and third qubit. This entangles all qubits together in a bell state. This means that the probability of the qubits sharing the same value is at its highest.

```
circ.h(0)
circ.cx(0, 1)
circ.cx(0, 2)
✓ 0.4s
```

Next I draw the circuit displaying it.

```
circ.draw('mpl')
✓ 0.1s
```
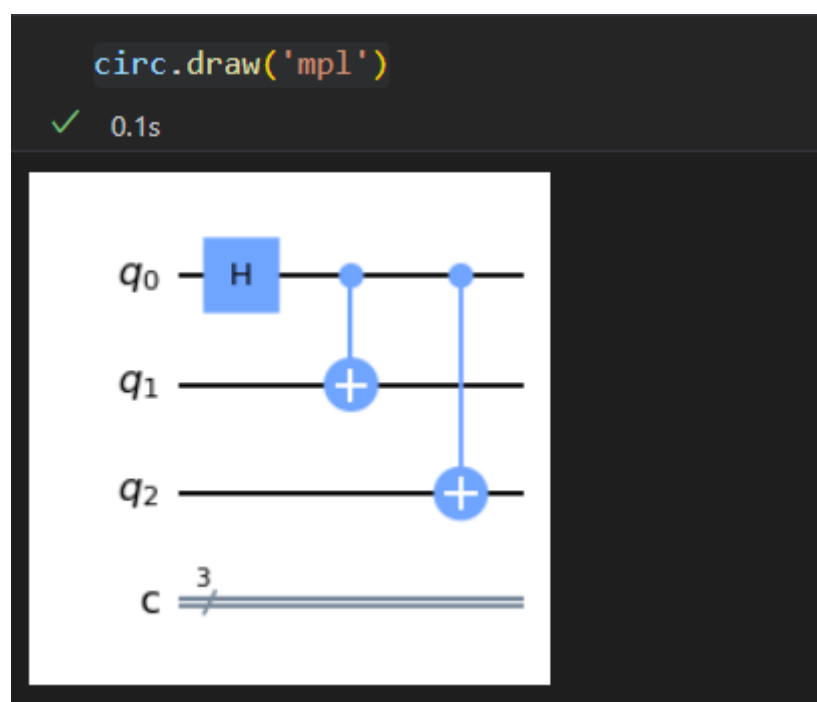
Next I add code which creates a state vector being the size of two to the power of three. Two being the number of possible values per qubit and 3 being the number of qubits, so 8 total possible values. After that I add the circuit to the state vector with the evolve function and visualize it. The sphere shows us that the result can be either 000 or 111 because of the entanglement.

```python
from qiskit.quantum_info import Statevector
# Set the intial state of the simulator to the ground state using from_int
state = Statevector.from_int(0, 2**3)
# Evolve the state by the quantum circuit
state = state.evolve(circ)
state.draw('qsphere')
```

✓ 0.6s
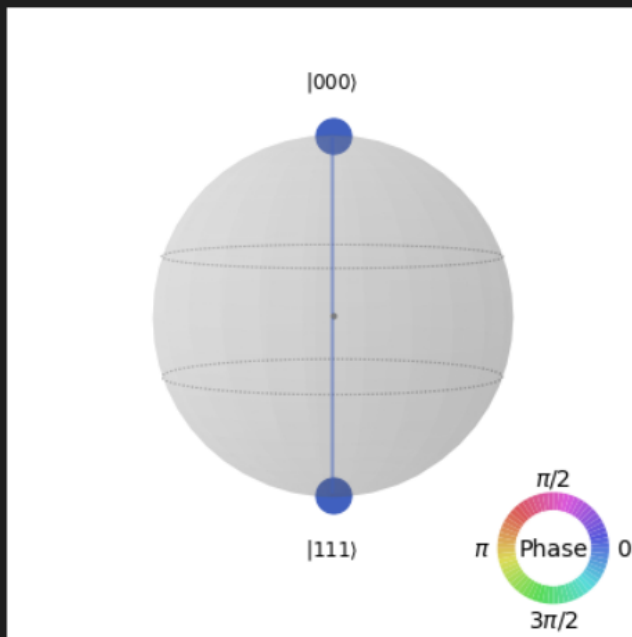
Next I prepare a quantum simulator and indicate I want to run the circuit an x number of times. Furthermore the circuit gets transpiled in order to use it within the simulator. Now I run the circuit and plot the result in a histogram. It shows that there is about a 50% chance for every bit to be either 0 or 1.

```python
backend = Aer.get_backend('aer_simulator')
shots = 100000

transpiled_circ = transpile(circ, backend)
result = backend.run(circ, shots=shots).result()
counts = result.get_counts(circ)
plot_histogram(counts, title='Bell-State counts')
```

✓ 0.3s
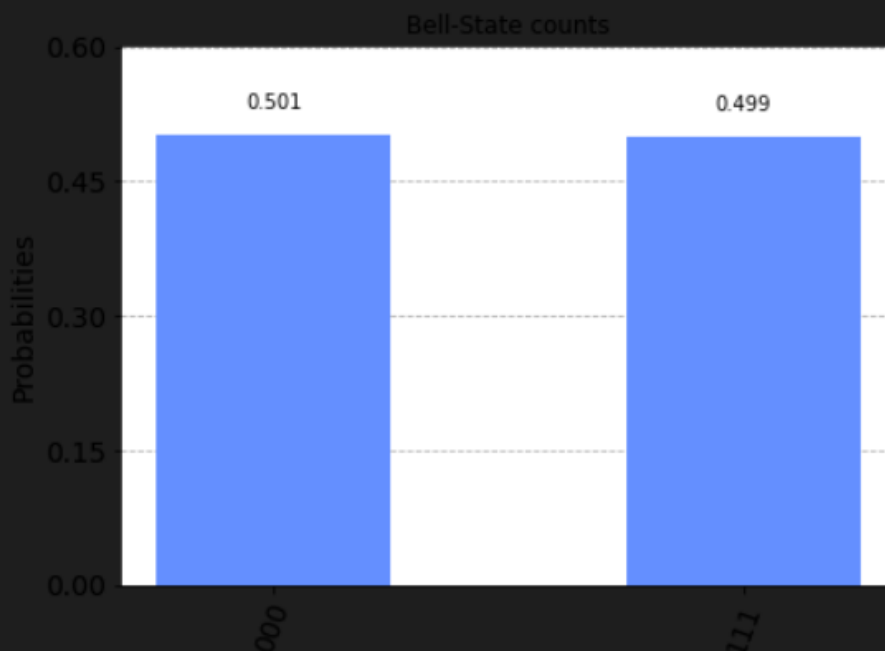
# 5. How do I program a Sudoku Solver using Qiskit? ✂️

In order to keep things doable I will be creating a 2x2 quantum sudoku solver with Grover's algorithm. The sudoku puzzle will be represented as a 2d array containing 4 values in total. My first action is to define clauses according to the rules. The idea is that the numbers in the array are the values which cannot be the same. So the first entry shows that value 0 and value 1 cannot be equal.

```
clause_list = [[0,1],
               [0,2],
               [1,3],
               [2,3]]
```

| Value 0 | Value 1 |
|---------|---------|
| Value 2 | Value 3 |

Next I define a XOR gate function by applying not gates (entanglement).

```
def XOR(qc, a, b, output):
    qc.cx(a, output)
    qc.cx(b, output)
```

Iterating over the clauses and assigning the NOT gates to the different check qubits gives us the circuit below. The out combines all checks into one result being either 0 (invalid) or 1 (valid).
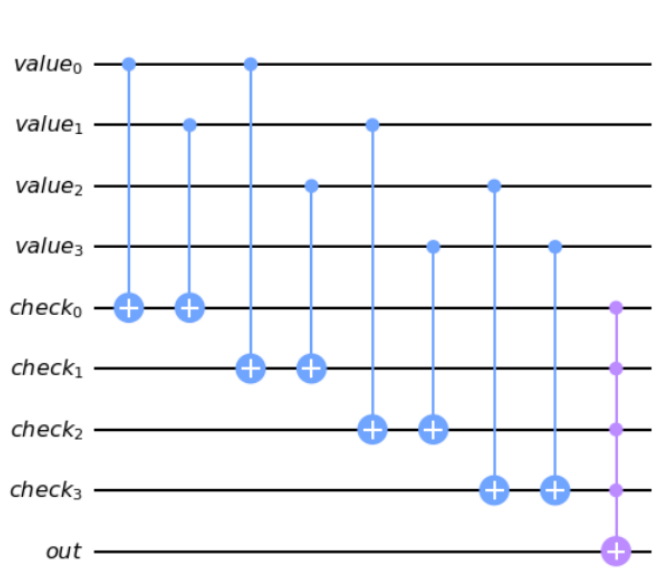
Now I define a sudoku oracle function and a diffuser function. The sudoku oracle function is responsible for checking the clauses and resetting the check qubits. The diffuser function is a general function responsible for applying the amplitude amplification trick.

It's time to combine all pieces. First I define variable qubits, clause qubits, output qubits and classic registers (cbits). The cbits are used to be able to capture the measurements done at the end of the circuit. These qubits are combined into one quantum circuit.

```
var_qubits = QuantumRegister(4, name='v')
clause_qubits = QuantumRegister(4, name='c')
output_qubit = QuantumRegister(1, name='out')
cbits = ClassicalRegister(4, name='cbits')
qc = QuantumCircuit(var_qubits, clause_qubits, output_qubit, cbits)
```

Next I initialize the out state by putting it in a complex position. After that I put the variable qubits into a superposition with the Hadamard gate and add a barrier for separation.

```
# Initialize 'out0' in state |->
qc.initialize([1, -1]/np.sqrt(2), output_qubit)

# Initialize qubits in state |s>
qc.h(var_qubits)
qc.barrier()  # for visual separation
```

Now I add the sudoku oracle to the circuit, another barrier for separation and finally apply the diffuser as a gate to the variable qubits. I do this for 2 iterations as I have noticed it yields the best results. This can be calculated with the formula stated below.

```
#iterations
iterations = 2
for i in range(iterations):
    sudoku_oracle(qc, clause_list, clause_qubits)
    qc.barrier()  # for visual separation
    # Apply our diffuser
    qc.append(diffuser(4), [0,1,2,3])
```

$N_{\text{optimal}} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} \right\rfloor$ is the optimal number of iterations that maximizes the likelihood of obtaining the correct item

*Optimal number of iterations (source: Microsoft)*
*M = different valid inputs*
*N=2n eligible items for the search task*

Having calculated N, the result roughly amounts to 1.7. Rounding this equates to 2 iterations.

Finally the variable qubits are measured and the circuit is drawn to an output cell. This way I can more easily verify nothing went wrong.

```
# Measure the variable qubits
qc.measure(var_qubits, cbits)

qc.draw('mpl',fold=-1)
```
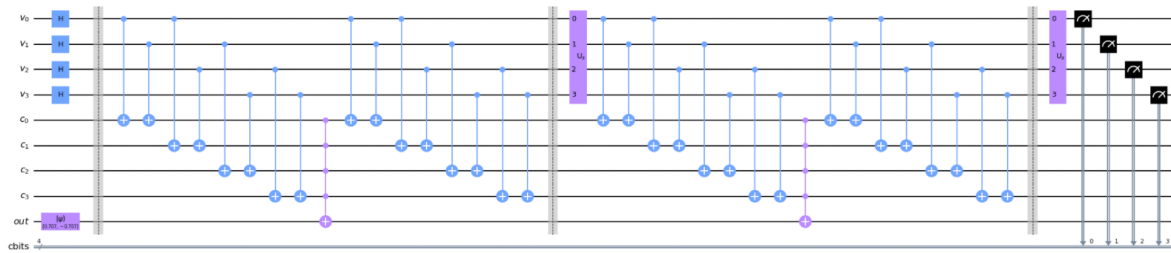


Now all that's left is to run the circuit and display the results.

```
shots=1000
aer_simulator = Aer.get_backend('aer_simulator')
transpiled_qc = transpile(qc, aer_simulator)
qobj = assemble(transpiled_qc)
result = aer_simulator.run(qobj, shots=shots).result()
plot_histogram(result.get_counts())
0.2s
```
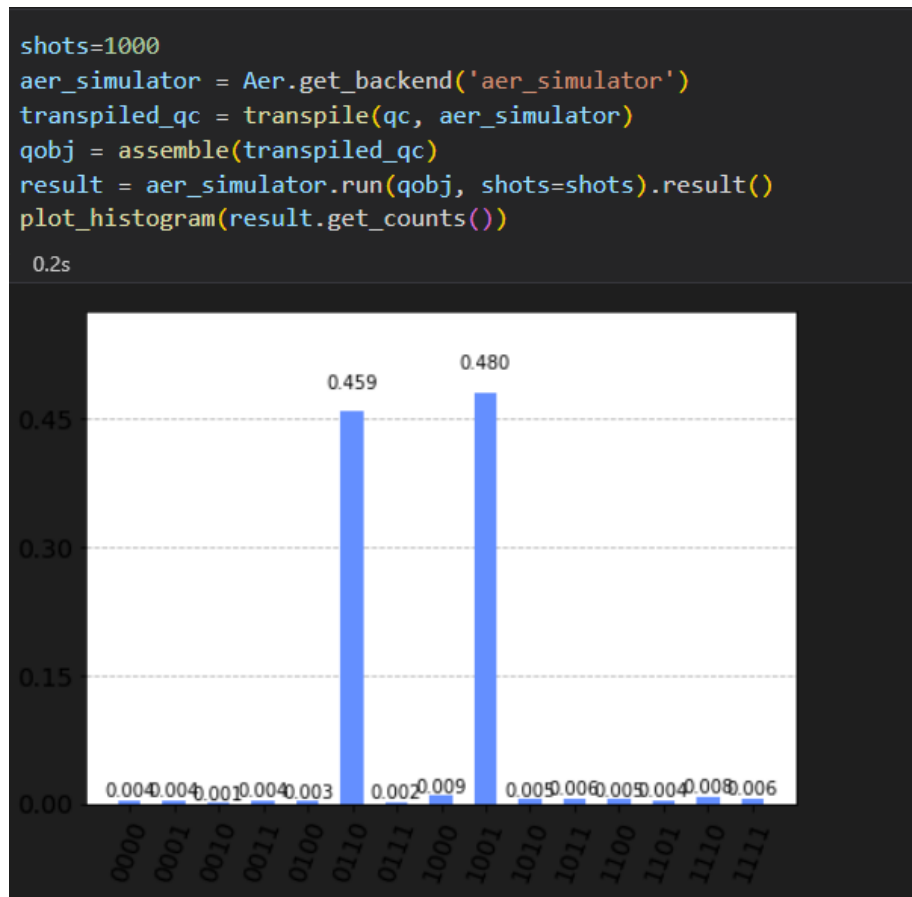


| 0 | 1 |
|---|---|
| 1 | 0 |

0.459%

| 1 | 0 |
|---|---|
| 0 | 1 |

0.480%

Having ran the simulator with 1000 shots, nearly a 100% of the time the algorithm produces 1 of the 2 correct answers.

The reason that I chose a 2x2 sudoku problem was that it requires a relatively small number of qubits. While a regular 9x9 sudoku is theoretically possible, it will probably use up to 486 qubits. The largest IBM computer only contains 127 qubits at this moment, which isn't even publicly available. My 2x2 solution uses 9 qubits, which should be able to run on some public quantum computers.

# 6. How does the performance of a quantum Sudoku Solver compare to other non-quantum computing sudoku solvers? 👔

In order to compare the quantum algorithm to a classic algorithm I wanted to first execute my circuit on a real-time public quantum computer. I started out by signing up for the IBM public quantum computers. This process was fairly straightforward and I managed to run a test circuit in a matter of a 15 minutes. However, the problem was that I couldn't use more than 5 qubits on any of their backends. I couldn't find a way to get access to higher count public quantum computers. This option unfortunately turned out to be not viable.

Next I tried to get access to the Rigetti aspen backends. Setting this up was a lot harder because it came with extra steps. I had to run a local qvm docker container, install a new package (qiskit_rigetti) and sign-up for an account. Unfortunately, I wasn't able to get access.

As of right now I can conclude that doing a practical performance test on a quantum circuit bigger than 5 qubits is not worth the effort as of right now. This problem will most likely be solved in the future as quantum computers will become even more available to the public.

Luckily, the question was already partially answered in a previous chapter. I mentioned that by using Grover's algorithm a classic algorithm can be sped up quadratically. This means that in optimal conditions a classic sudoku solver that would normally take 64 steps to solve, will be solved in 8 steps. If 1 step on a classical computer can be done in the same time as on a quantum computer, the resulting benchmark would look as following:

| Example 1 | Steps | Seconds |
|---|---|---|
| **Classical** | 64 | 2 |
| **Quantum** | 8 | 0,25 |

| Example 2 | Steps | Seconds |
|---|---|---|
| **Classical** | 10,000 | 312.5 |
| **Quantum** | 100 | 3.125 |

*\* Assuming a classical computer and quantum computer have the same processing speed*
*\*\*  Assuming 1 step = 0.03125 seconds (arbitrary)*

As you can see in example 2, when the steps increase to a higher number the difference becomes really significant.

## Conclusion

Was I able to create a performant sudoku solver? Yes and no, because of the maturity of the technology and therefor the low level of abstraction it was quite a challenge to program something as simple as a 2x2 sudoku solver. Before I started on the research I was hoping I could build a fully-fledged 9x9 sudoku solver in a matter of 2 weeks. Unfortunately, my expectations did not come entirely true. I also did not manage to apply benchmark testing to the product. So the only thing I can conclude is that it should be theoretically more performant than a regular sudoku solver.

# Reflection

It is general knowledge that understanding quantum concepts is quite difficult. I also noticed that my math and physics knowledge was not up to the standard. This contributed to me finding it difficult to grasp a lot of the theory. I do still feel like I learned a lot and managed to learn a lot within a short time span.

What I would do differently next time is choose a smaller research scope. During my research I ran into a lot of questions and the rabbit hole seemed to only go deeper. Also, if it would have been possible within the time constraints, I would have studied some algebra in advance.

Overall I am happy with the result. I think this newfound knowledge could potentially help me in the future. I learned what Qubits are, the use of Grover's algorithm and how to write basic programs on a Quantum Circuit using Python and Qiskit.

# References

***Getting started — Qiskit 0.36.1 documentation*. (2022). Getting Started - Qiskit.**

Retrieved 10 May 2022, from https://qiskit.org/documentation/getting_started.html

**Team, T. Q. (2022, April 28).** *Grover's Algorithm*. **Grover's Algorithm.**

Retrieved 11 May 2022,

from https://qiskit.org/textbook/ch-algorithms/grover.html

**Team, T. Q. (2022b, April 28).** *Single Qubit Gates*. **Single Qubit Gates.**

Retrieved 11 May 2022,

from https://qiskit.org/textbook/ch-states/single-qubit-gates.html

**Wikipedia contributors. (2022, May 10).** *Grover's algorithm*.  **Wikipedia.**

Retrieved 11 May 2022,

from https://en.wikipedia.org/wiki/Grover%27s_algorithm

**Vanguard, D. (2020).** *What is a qubit?*  **Quantum Inspire.**

Retrieved 10 May 2022,

from https://www.quantum-inspire.com/kbase/what-is-a-qubit/

*Composer*. **(2016). IBM Quantum.**

Retrieved 10 May 2022,

from https://quantum-computing.ibm.com/composer/files/new

*A beginner's guide to quantum computing | Shohini Ghose*. **(2019, February 1). [Video].**

**YouTube.**

https://www.youtube.com/watch?v=QuR969uMICM

*Logic Gates Rotate Qubits*. **(2021, August 20). [Video]. YouTube.**

https://www.youtube.com/watch?v=ZBaXPY_0TNI

*IBMQBackend — Qiskit 0.36.1 documentation*. (2022). IBMQBackend.

Retrieved 17 May 2022, from

https://qiskit.org/documentation/stubs/qiskit.providers.ibmq.IBMQBackend.html

R. (2021). *GitHub - rigetti/qiskit-rigetti: Qiskit provider serving Rigetti hardware &*

*simulator backends.* GitHub.

Retrieved 17 May 2022,

from https://github.com/rigetti/qiskit-rigetti

*The DOT Framework - ICT research methods*. (2021). The DOT Framework.

Retrieved 10 May 2022,

from https://ictresearchmethods.nl/The_DOT_Framework

*Anaconda | The World's Most Popular Data Science Platform*. (2022). Anaconda.

Retrieved 10 May 2022,

from https://www.anaconda.com/

*Warke,* C. (2021, December 9). *Introduction to quantum computing part -1 Representation*

*of qubit using Bloch sphere*. Medium.

Retrieved 11 May 2022,

from https://medium.com/random-techpark/introduction-to-quantum-computing-

part-1-representation-of-qubit-using-bloch-sphere-c6830d6ac240

*Tutorials — Qiskit 0.36.1 documentation*. (2022). Tutorials.

Retrieved 11 May 2022,

from https://qiskit.org/documentation/tutorials.html

B. (2022, April 1). *Theory of Grover's search algorithm - Azure Quantum*. Microsoft Docs.

Retrieved 11 May 2022,

from https://docs.microsoft.com/en-us/azure/quantum/concepts-grovers