Qudoku: solving Sudoku using Grover's algorithm

Quantum Computing
Computer Science Master Degree
University of Parma

Federico Serafini
`federico.serafini@studenti.unipr.it`

08/02/2023

Contents

1 Introduction

Many interesting problems that computer science tries to solve are *search problems* over *structured* and *unstructured* databases. The best algorithm for searching a winning element ω in a structured database, a sorted list of N elements for example, is the *binary search*: it allows to find ω in $\mathcal{O}(\log N)$ tries exploiting data ordering. Things get more difficult when the search is done over a list of N randomly-placed elements. Using a classical algorithm there is no way of taking advantage of the data structure to speed up the search: in the worst case, where the searched element is at the end of the list, a scan over all the N elements of the list is required. In the average case, the number of tries to find ω is $N/2$. So, the overall time complexity is $\mathcal{O}(N)$ and this means that time cost for classical unstructured search algorithms grows linearly with the size of the input.

In this report, we are going to discuss a new search algorithm based on quantum computation: Grover's algorithm. When searching any database (structured or unstructured), Grover's algorithm time complexity is $\mathcal{O}(\sqrt{N})$, which for unstructured search is a quadratic improvement over the best classical algorithm. It can serve as a general trick or subroutine to obtain quadratic run time improvements for a variety of other algorithms through *amplitude amplification trick*. In particular, we are going to see a simple example of conversion from a classical search problem as solving a binary Sudoku, into oracles for Grover's algorithm.

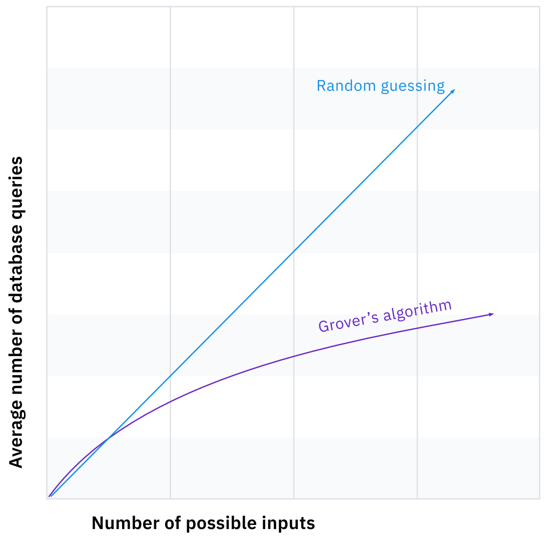

Figure 1: Time complexity.

2 Basic concepts

2.1 Definitions

Decision problem In computability theory and computational complexity theory, a *decision problem* is a computational problem that can be posed as a yes/no question of the input values. It is traditional to define the decision problem as the set of possible inputs together with the set of inputs for which the answer is yes. In our case, it will be useful to formulate the Sudoku as a decision problem, where the input values are all the n^{n^2} ways of filling the board and the output is the set of valid solutions.

Oracle Many quantum algorithms are based around the analysis of some *oracle* function f: a "black box" which we can give an input x and receive the corresponding output $f(x)$. Our objective is to determine some properties of the oracle function using the minimum number of queries.

2.2 Phase kickback

2.2.1 Rotations and eigenvalues

Given quantum gate U, and it's eigenstate $|x\rangle$, then:

$$U|x\rangle = \lambda |x\rangle = e^{2\pi i\theta} |x\rangle \,,$$

meaning that if a gate rotates (and only rotates) all the amplitudes of a state vector by the same amount, then that state is an eigenstate of that gate and U acting on $|x\rangle$ will add a global phase θ. For example, performing an X-gate on a $|-\rangle$ qubit gives it the phase -1:

$$\begin{aligned} X|-\rangle &= \tfrac{1}{\sqrt{2}}\big(|0\rangle\langle 1| + |1\rangle\langle 0|\big)\big(|0\rangle - |1\rangle\big) \\ &= \tfrac{1}{\sqrt{2}}\big(|0\rangle\langle 1| + |1\rangle\langle 0|\big)\big(|0\rangle - |1\rangle\big) \\ &= \tfrac{1}{\sqrt{2}}\big(|0\rangle\langle 1|0\rangle - |0\rangle\langle 1|1\rangle + |1\rangle\langle 0|0\rangle - |1\rangle\langle 0|1\rangle\big) \\ &= \tfrac{1}{\sqrt{2}}\big(-|0\rangle + |1\rangle\big) \\ &= -\tfrac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big) = -|-\rangle \,. \end{aligned}$$

2.2.2 Kick back

Phase kickback is where the eigenvalue added by a gate to a qubit is "kicked back" into a different qubit via a controlled operation. When the control qubit is in either $|0\rangle$ or $|1\rangle$, this phase affects the whole state, however it is a global phase and has no observable effects:

$$\begin{aligned} CX|{-0}\rangle &= |-\rangle \otimes |0\rangle \\ &= |{-0}\rangle \,, \end{aligned}$$

$$\begin{aligned} CX\left|-1\right\rangle &= X\left|-\right\rangle \otimes \left|1\right\rangle \\ &= -\left|-1\right\rangle . \end{aligned}$$

Things get interesting when the control qubit is in a superposition of $|0\rangle$ and $|1\rangle$. The component of the control qubit that lies in the direction of $|1\rangle$ applies this phase factor to the corresponding target qubit. This applied phase factor in turn introduces a relative phase into the control qubit:

$$CU\left|x\right\rangle \big(\alpha\left|0\right\rangle + \beta\left|1\right\rangle\big) = \left|x\right\rangle \big(\alpha\left|0\right\rangle + \beta e^{2i\pi\theta}\left|1\right\rangle\big).$$

For example:

$$\begin{aligned} CX\left|-+\right\rangle &= \tfrac{1}{\sqrt{2}}\big(X\left|-0\right\rangle + X\left|-1\right\rangle\big) \\ &= \tfrac{1}{\sqrt{2}}\big(\left|-0\right\rangle + X\left|-1\right\rangle\big) \\ &= \tfrac{1}{\sqrt{2}}\big(\left|-0\right\rangle - \left|-1\right\rangle\big) \\ &= \left|-\right\rangle \otimes \tfrac{1}{\sqrt{2}}\big(\left|0\right\rangle - \left|1\right\rangle\big) = \left|--\right\rangle . \end{aligned}$$

This interesting quantum effect is a building block in many famous quantum algorithms, including Grover's search algorithm.

2.2.3 Oracles

Classical oracle On classical algorithms, oracles can be seen as general functions:

$$f : \{\,0,1\,\}^n \mapsto \{\,0,1\,\}^m,$$

where $n, m \in \mathbb{N}$ are the input and output length.

Boolean oracle In quantum computation, one of the main forms that oracles take is that of *boolean oracles*, described by the following unitary evolution:

$$U_f\big(\left|x\right\rangle \otimes \left|y\right\rangle\big) = \left|f(x) \oplus y\right\rangle ,$$

where:

- x is the input register (n qubits, $n \geq 1$);
- y is the output register (m qubits, $m \geq 1$);
- $\oplus$ is an exclusive or.

Note that the result of applying U_f depends on:

1. U_f definition;
2. initial contents of both input and output registers, in particular, if $|y\rangle = |0^m\rangle$ then $|f(x) \oplus y\rangle = |f(x)\rangle$.

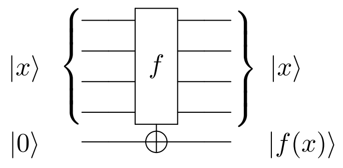

Figure 2: Boolean oracle.

Phase oracle Another form of oracle used in quantum computation is the *phase oracle*, defined as:

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

A phase oracle can be realized using boolean oracle and the phase kickback mechanism:

$$U_f\big(|x\rangle \otimes |y\rangle\big) = U_f\big(|x\rangle \otimes |-\rangle\big) = \big(P_f \otimes I\big)\big(|x\rangle \otimes |-\rangle\big) = P_f |x\rangle \otimes |-\rangle = P_f |x\rangle .$$

Input x controls an X rotation targeting the output qubit y in state $|-\rangle$: an X rotation is applied to $|-\rangle$ if $f(x) = 1$, resulting in a kickback of the phase -1 on the state $|x\rangle$. Output qubit y (whose state is left unchanged by the whole process and can safely be ignored) it is called *ancilla* qubit.

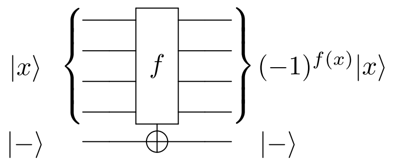

Figure 3: Phase oracle.

3 Grover's algorithm

Now we are going to see how the Grover's algorithm works when searching for a winning item ω among N items. Before diving into the details it can be useful to visualize the shape of the circuit that implements it:

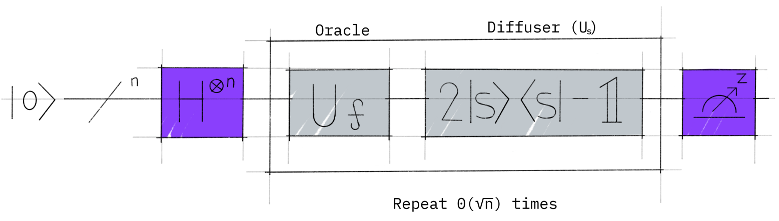

Figure 4: Grover's circuit.

3.1 Preparation

Before looking at the list of items, we have no idea where the winning item is. Therefore, any guess of its location is as good as any other, which can be expressed in a equal superposition of every possible input:

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

We can create this superposition by applying a H-gate to each input qubit.

3.2 Grover oracle

Grover's oracle U_ω it is a phase oracle, so its application on an input state $|x\rangle$ can be described as:

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

First thing we need to define is the *indicator function* f. A strong point of this algorithm is that it is *general*: definition of f can vary depending on the search problem we are addressing. In our case, the indicator function f is defined as:

$$f|x\rangle = \begin{cases} 1, & \text{if } x = \omega; \\ 0, & \text{otherwise.} \end{cases}$$

In general, there are many computational problems in which it's difficult to find a solution, but relatively easy to verify a solution; let W be the set of all valid solutions of a problem, then f will be the function that tell us if a candidate solution x it is a valid solution or not.

$$f|x\rangle = \begin{cases} 1, & \text{if } x \in W; \\ 0, & \text{otherwise.} \end{cases}$$

Given f, we can observe that Grove's oracle U_ω adds a negative phase only to the winning element:

$$U_\omega|x\rangle = \begin{cases} -|x\rangle, & \text{if } x = \omega; \\ |x\rangle, & \text{otherwise.} \end{cases}$$

Another way of representing U_ω is taking an identity matrix I and adding a -1 phase to the winning element:

$$U_\omega = \begin{bmatrix} (-1)^{f(0)} & 0 & \cdots & 0 \\ 0 & (-1)^{f(1)} & \cdots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{f(2^n-1)} \end{bmatrix}$$

3.3 Amplitude amplification

Finally it is time to apply the *amplitude amplification* trick. A procedure that amplifies the amplitude of the winning element, which shrinks the other items' amplitude, so that measuring the final state will return the right item with high probability. There a nice geometrical interpretation of the amplitude amplification in terms of two reflections, which generate a rotation in a two-dimensional plane.

Step 1 Consider the vectors corresponding to the winner $|\omega\rangle$ and the uniform superposition $|s\rangle$. These two vectors span a two-dimensional plane. They are not perpendicular because $|\omega\rangle$ occurs in the superposition $|s\rangle$ with amplitude $\frac{1}{\sqrt{2^n}}$ as well. We can, however, introduce an additional state $|s'\rangle$, that is in the span of these two vectors and is perpendicular to $|\omega\rangle$ and (obtained from $|s\rangle$ by removing $|\omega\rangle$):

$$|s'\rangle = \frac{1}{\sqrt{2^{n-1}}} \sum_{k \neq \omega} |k\rangle \,.$$

Then $|s\rangle$ is a linear combination of $|\omega\rangle$ and $|s'\rangle$:

$$|s\rangle = \frac{1}{\sqrt{2^n}} |\omega\rangle + \sqrt{\frac{2^n-1}{2^n}} |s'\rangle = \sin\theta\, |\omega\rangle + \cos\theta\, |s'\rangle \,,$$

where $\theta = \arcsin \langle s|\omega\rangle = \frac{1}{\sqrt{2^n}} = \frac{1}{\sqrt{N}}$ is the amplitude af all the different states.

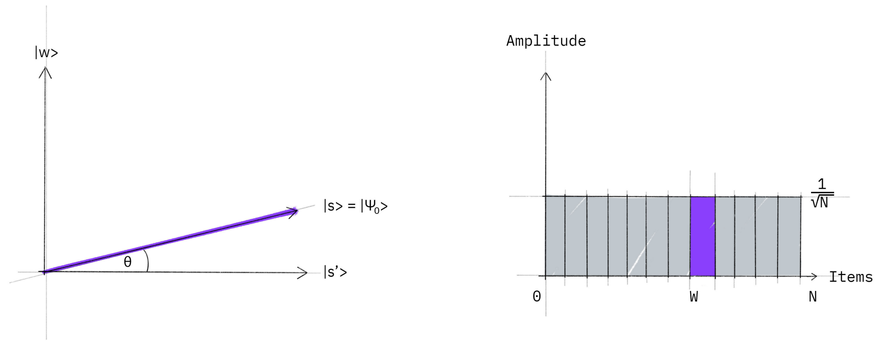

Figure 5: $|s\rangle = H^{\otimes^n} |x\rangle$.

Step 2 Now we apply the oracle U_ω to the superposition state $|s\rangle$. Geometrically, this corresponds to a reflection of the state $|s\rangle$ about $|s'\rangle$: the amplitude in front of the $|\omega\rangle$ becomes negative, which in turn means that the average amplitude has been lowered.

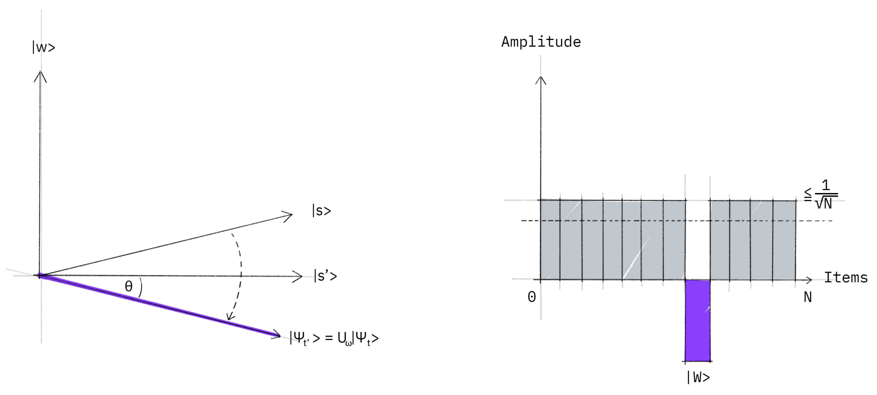

Figure 6: $U_\omega |s\rangle$.

Step 3 Apply the *diffuser* $U_s = 2|s\rangle\langle s| - I$ to the state $|s\rangle$. Since this is a reflection about the state $|s\rangle$, we want to add a negative phase to every state orthogonal to $|s\rangle$. One way to do this is:

1. transform $|s\rangle \rightarrow |0^{\otimes n}\rangle$ applying Hadamard gates;
2. apply a negative phase to the states orthogonal to $|0^{\otimes n}\rangle$ using $U_0 = X^{\otimes n}(MCZ)X^{\otimes n}$;
3. transform $|0^{\otimes n}\rangle \rightarrow |s\rangle$ applying H-gate again.

Putting all together $U_s = H^{\otimes n}X^{\otimes n}(MCZ)X^{\otimes n}H^{\otimes n} = H^{\otimes n}U_0H^{\otimes n}$.

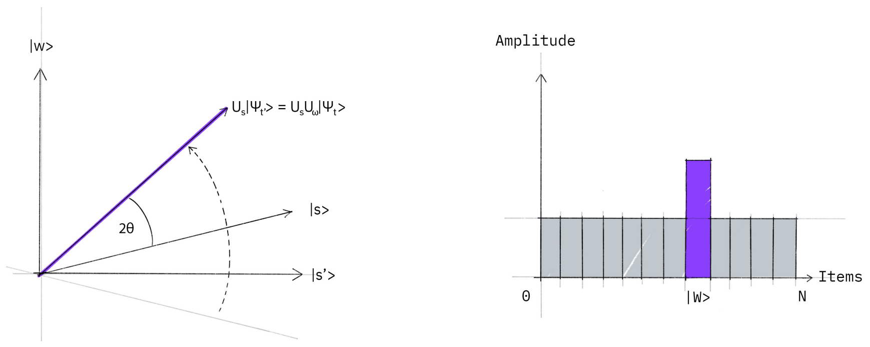

Figure 7: $U_s U_\omega \left|s\right\rangle$

The transformation $U_s U_\omega \left|s\right\rangle$ rotates $\left|s\right\rangle$ closer towards the winner $\left|\omega\right\rangle$. The action of the reflection in the amplitude bar diagram can be understood as a reflection about the average amplitude. Since the average amplitude has been lowered by the first reflection, this transformation boosts the negative amplitude of $\left|\omega\right\rangle$ to roughly three times its original value, while it decreases the other amplitudes. We then go to step 2 to repeat the application.

3.4 Repetitions

3.4.1 Single element

Each application of steps 2 and 3, rotates $\left|s\right\rangle$ closer towards the winner $\left|\omega\right\rangle$ of an angle 2θ. We would like to find the minimum number of repetitions that brings us as close as possible to $\left|w\right\rangle$: a $90°$ rotation taking into account the initial position of $\left|S\right\rangle$.

$$\begin{aligned} &t(2\theta) = \frac{\pi}{2} - \theta, \\ &t = \frac{\pi}{4\theta} - \frac{\theta}{2\theta}, \\ &t = \frac{\pi}{4}\sqrt{N} - \frac{1}{2} = \mathcal{O}(\sqrt{N}). \end{aligned}$$

3.4.2 Multiple element

Let M be the number of winning (or marked) elements. Then the winning state $\left|\omega\right\rangle$ is defined as:

$$\left|\omega\right\rangle = \frac{1}{\sqrt{M}} \sum_{i=1}^{M} \left|\omega_i\right\rangle.$$

Then:

- $|s'\rangle = \frac{1}{\sqrt{N-M}} \sum_{k \notin \{\omega_1 \dots \omega_M\}} |k\rangle$;
- $|s\rangle = \sqrt{\frac{M}{N}} |\omega\rangle + \sqrt{\frac{N-M}{N}} |s'\rangle = \sin\theta\, |\omega\rangle + \cos\theta\, |s'\rangle$.

Angle θ becomes larger:

$$\theta = \arcsin \langle s|w\rangle = \sqrt{\frac{M}{N}},$$

and t reduces:

$$\begin{aligned} t(2\theta) &= \frac{\pi}{2} - \theta, \\ t &= \frac{\pi}{4\theta} - \frac{\theta}{2\theta}, \\ t &= \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} = \mathcal{O}\left(\sqrt{\frac{N}{M}}\right). \end{aligned}$$

4 Solving Sudoku

We will apply the Grover's algorithm to solve a 2×2 binary Sudoku. Considering all the possible ways of filling the board, the search space size is $N = 2^4$, and a candidate solution $x = |v_3, v_2, v_1, v_0\rangle$ is a binary number in the interval $[0 \mathinner{.\,.} N-1]$.

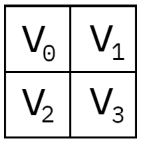

Figure 8: Binary Sudoku.

4.1 Circuit code

4.1.1 Global variables

```
l = 2    # Borad length.
n = l*l  # Borad size.

# Row clauses: 0 != 1, 2 != 3.
# Col clauses: 0 != 2, 1 != 3.
clause_list = [[0, 1], [0, 2], [1, 3], [2, 3]]

######## Registers. ########

var_qubits = QuantumRegister(n, name='v')     # Variables.
clause_qubits = QuantumRegister(n, name='c')  # Clause-checks.
```

```python
out_qubit = QuantumRegister(1, name='out')     # f(x).
cbits = ClassicalRegister(n, name='cbits')     # Classical bits.
```

4.1.2 Indicator function

Let W be the set of all valid solutions (winning elements). A candidate solution x is considered valid ($x \in W$) if the following conditions are met:

1. no row may contain the same value twice ($v_0 \neq v_1, v_2 \neq v_3$);
2. no columns may contain the same value twice ($v_0 \neq v_2, v_1 \neq v_3$).

We can define our function f as the indicator function of W:

$$f(x) = \begin{cases} 1, & \text{if } x \in W; \\ 0, & \text{otherwise.} \end{cases}$$

```python
# Use XOR to check every single clause.
def XOR(qc, q_in1, q_in2, q_out):

  qc.cx(q_in1, q_out)
  qc.cx(q_in2, q_out)

# Indicator function for binary Sudoku.
def f(n):

  qc = QuantumCircuit(var_qubits, clause_qubits)
  # Check every single clause using XOR.
  c = 0
  for clause in clause_list:
    XOR(qc, clause[0], clause[1], clause_qubits[c])
    c += 1

  f = qc.to_gate()
  f.name = "f"
  return f
```

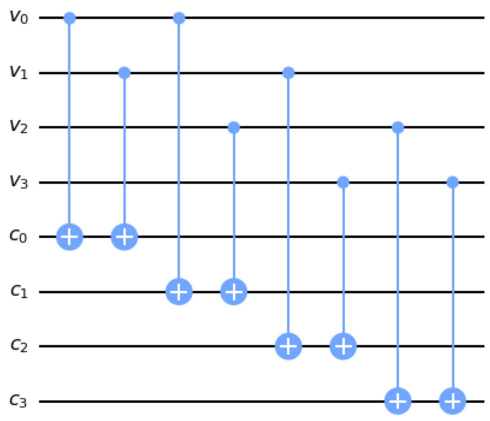

Figure 9: Sudoku indicator function f.

4.1.3 Oracle

```python
def phase_oracle(n):

  qc = QuantumCircuit(var_qubits, clause_qubits, out_qubit)

  # Apply f.
  qc.append(f(n), range(2*n))

  # Flip out qubit if all clauses are satified.
  qc.mct(list(range(4, 2*n)), 2*n)

  # Uncompute applying f a second time (set clause qubits to zero).
  qc.append(f(n), range(2*n))

  # Return oracle as a gate.
  U_f = qc.to_gate()
  U_f.name = "U_f"
  return U_f
```

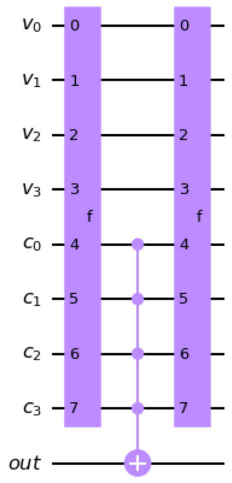

Figure 10: Sudoku oracle U_f.

4.1.4 Diffuser

```python
# Diffuser.
def diffuser(n):

  qc = QuantumCircuit(var_qubits)

  # Apply transformation |s> -> |00..0>.
  for qubit in range(n):
    qc.h(qubit)

  # Apply transformation |00..0> -> |11..1>.
  for qubit in range(n):
    qc.x(qubit)

  # Multi controlled Z-gate.
  qc.h(n-1)
  qc.mct(list(range(n-1)), n-1)
  qc.h(n-1)

  # Apply transformation |11..1> -> |00..0>.
  for qubit in range(n):
    qc.x(qubit)

  # Apply transformation |00..0> -> |s>.
  for qubit in range(n):
    qc.h(qubit)

  # Return the diffuser as a gate.
  U_s = qc.to_gate()
  U_s.name = "U_s"
  return U_s
```

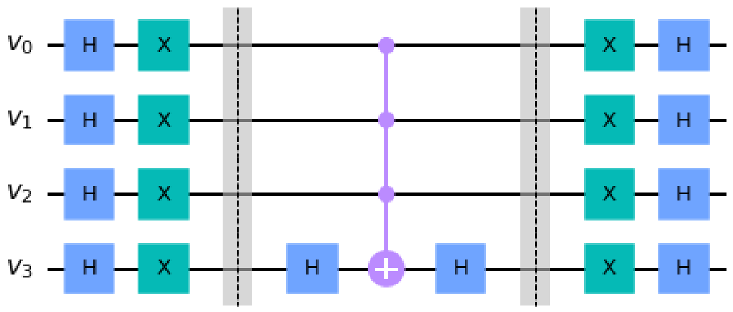

Figure 11: Diffuser U_s.

4.1.5 Amplitude amplification

```
######## Preparation. ########

# Create a quantum circuit.
qc = QuantumCircuit(var_qubits, clause_qubits, out_qubit, cbits)

# Superposition.
for q in var_qubits:
  qc.h(q)

# Initialize 'out' in state |->.
qc.x(out_qubit)
qc.h(out_qubit)
qc.barrier()

######## Amplification. ########

t = 2

for i in range(t):
  qc.append(phase_oracle(n), range(2*n +1))
  qc.append(diffuser(n), range(n))
  qc.barrier()
```

```

######## Measure. ########

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

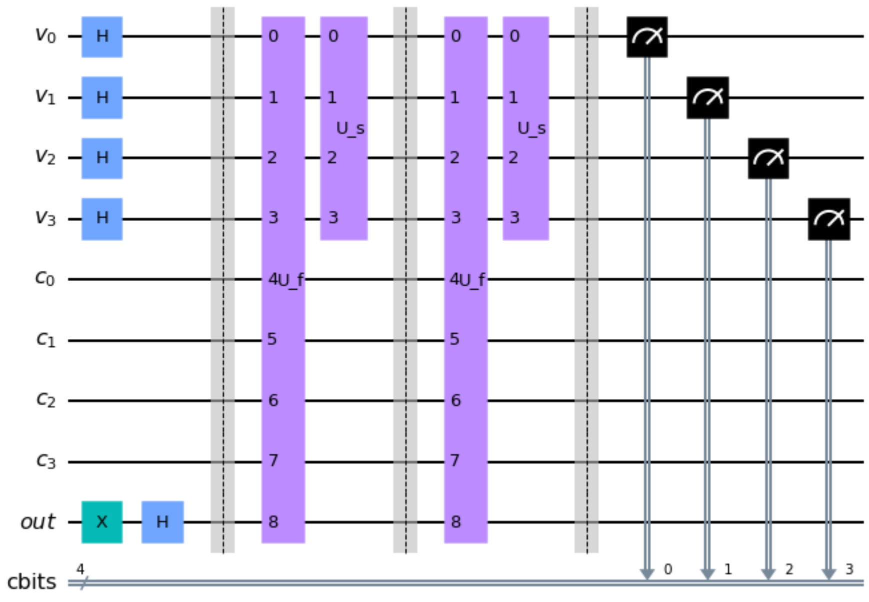

Figure 12: Solving Sudoku circuit.

5 Simulation

5.1 Ideal simulation

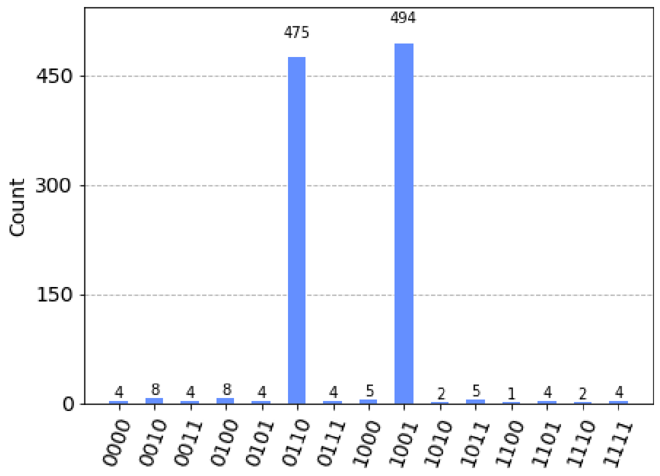

Figure 13: circuit simulation $t = 2$.

From the results of 1024 ideal simulations of our circuit we can observe that the phase amplification trick it is working properly and it is really powerful. In our case, following the theory and knowing in advance that $N = 2^4, M = 2$, we can find that:

$$t = \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} = \frac{\pi}{4}\sqrt{8} - \frac{1}{2} \approx 2.$$

What happen with $t = 1$ or $t = 3$? Result quality reduces:

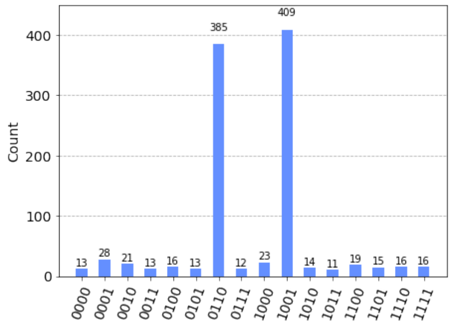

Figure 14: circuit simulation $t = 1$.

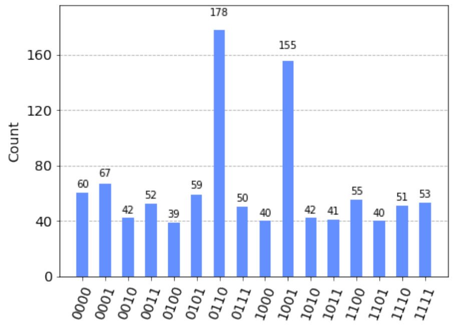

Figure 15: circuit simulation $t = 3$.

5.2 Noisy simulation

and comment the results obtained by running the circuit on the simulator (and possibly on the hardware), comparing the effects of different sources of noise (depolarizing errors on 1 or 2-qubit gates, relaxation, measurement). Try to apply some mitigation techniques to increase the quality of the final results.

6 Conclusion

In general, given a Sudoku, a rough upper bound for the search space size is n^{n^2}, corresponding the number of all the possible ways of filling an $n \times n$ board using n values. Therefore, a rough upper bound for a "classical" board with $n = 9$ is 9^{81}, that is near to the number of atoms in the universe; exploiting the game rules is therefore essential to reduce the search space and find a solution in a timely manner. This is the key idea behind all the different and well-studied search techniques known in our days.