An Introduction to Quantum Computing for Computer Scientists, with SAT

Software Engineering II Research Project

Politecnico di Milano

April 29, 2020

Author:
Francesco Piro

Professors:
Elisabetta Di Nitto
Mehrnoosh Askarpour

Outline

Quantum Computing

Quantum Computing Introduction

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**
4. **No-cloning principle**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**
4. **No-cloning principle**
5. **Uniform superposition of the initial state**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**
4. **No-cloning principle**
5. **Uniform superposition of the initial state**
6. **Can we solve NP-complete problems?**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**
4. **No-cloning principle**
5. **Uniform superposition of the initial state**
6. **Can we solve NP-complete problems?**
7. **Can we solve NP-hard problems?**

During the study of the state of the art of quantum computing we were able to define 8 important conclusions that we used in order to set the basis for the study of the SAT problem

1. **Exponential growth of the state space**
2. **Universal quantum computer is Turing complete**
3. **State collapses after measurement**
4. **No-cloning principle**
5. **Uniform superposition of the initial state**
6. **Can we solve NP-complete problems?**
7. **Can we solve NP-hard problems?**
8. **Query complexity**

Exponential growth of the state space

The dimension of the state space of quantum registers grows exponentially in the number of qubits, whereas the dimension of the state space of classical registers grows linearly in the number of bits.

- **Classical Register**
  Sequence of bits with values in $\{0, 1\}$
  The state of a classical machine with one register and n bits is a binary string in $\{0, 1\}^n$
  Thus a n-dimensional space

Exponential growth of the state space

The dimension of the state space of quantum registers grows exponentially in the number of qubits, whereas the dimension of the state space of classical registers grows linearly in the number of bits.

- **Classical Register**
  Sequence of bits with values in $\{0, 1\}$
  The state of a classical machine with one register and n bits is a binary string in $\{0, 1\}^n$
  Thus a n-dimensional space
- **Quantum Register**
  Sequence of qubits with values in $[0, 1]$
  The state of a quantum machine with one register and n qubits is obtained by the tensor product of n $\mathbb{C}^2$ vectors
  Thus a 2^n dimensional space

A universal quantum computer is Turing-complete.

A quantum register of n-qubits represents a quantum state $|\psi\rangle \in \mathbb{C}^{2^n}$.
A quantum operation is a matrix U such that:

$$U \in \mathbb{C}^{2^n \times 2^n}$$

The application of U onto the state $|\psi\rangle$ is the unit vector:

$$U|\psi\rangle \in \mathbb{C}^{2^n}$$

The evolution of a state into another shows two important features of quantum gates:

1. Quantum operations are **linear**
2. Quantum operations are **reversible**

It can be proved that a quantum machine with operations like U is Turing complete

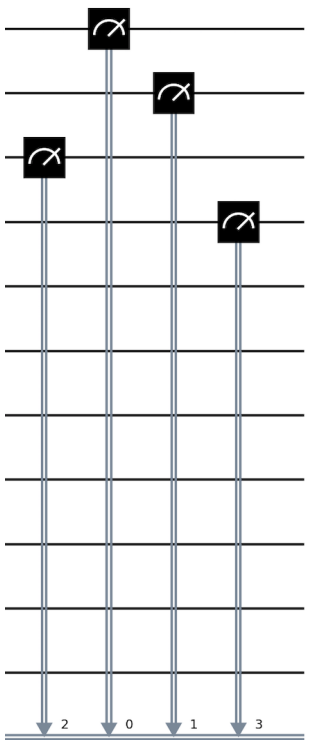
State collapses after measurement

The state of the quantum system after a measurement collapses to a linear combination of only those basis states that are consistent with the outcome of the measurement. The original quantum state is no longer recoverable.

The **measurement** operation allows to bring on a classical bit the value of a qubit at the moment it is performed. Measuring a qubit is based on the famous quantum principles:

- Superposition
- Entanglement

It is typically used to measure the final state of a quantum machine to check if the expected result has been obtained



No-cloning principle

It is impossible to clone quantum states.

Measuring a state makes it collapse to a boring sequence of 1s and 0s
A measured qubit is no longer in superposition, it is either 1 or 0

Is it possible to copy a state before measuring it ?

The answer unfortunately is **NO!**
It may be useful to split the execution of the quantum algorithm continuing with different operations on the same state

Whenever we run a circuit that produces an output quantum state, in general we can reproduce the output quantum state by only repeating all the steps of the algorithm.

Uniform superposition of the initial state

Applying operations on a quantum device whose state is in a uniform superposition allows to apply them simultaneously to all possible binary strings thanks to linearity.

To achieve the power of this conclusion we typically initialize the state of the quantum machine in a uniform superposition

The **Hadamard** gate on all the n-qubits of the register brings the initial state in the uniform superposition

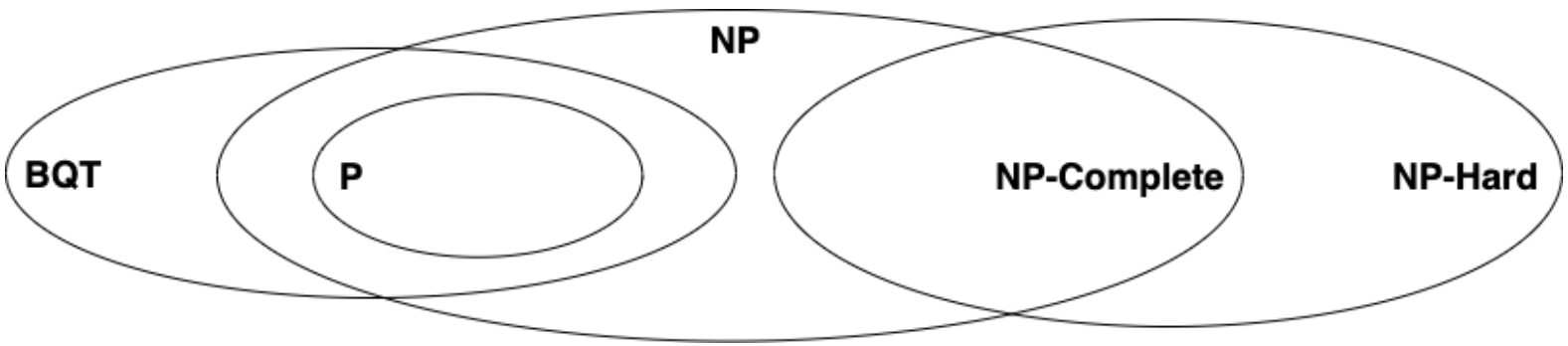
We will see how to use it in the first step of **Grover's search algorithm**

The formal definition of an n-Hadamard gate is:

$$\bigotimes^{n} \mathcal{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} \bigotimes^{n-1} \mathcal{H} & \bigotimes^{n-1} \mathcal{H} \\ \bigotimes^{n-1} \mathcal{H} & -\bigotimes^{n-1} \mathcal{H} \end{pmatrix}$$

Conclusion 6:
Even if we can easily create a uniform superposition of all basis states, the rules of measurement imply that using just this easily-obtained superposition does not allow us satisfactorily solve NP-complete problems, such as, for example, SAT.

Conclusion 7:
In general solving NP-Hard problems in polynomial time with quantum computers is not believed to be possible.



Query complexity

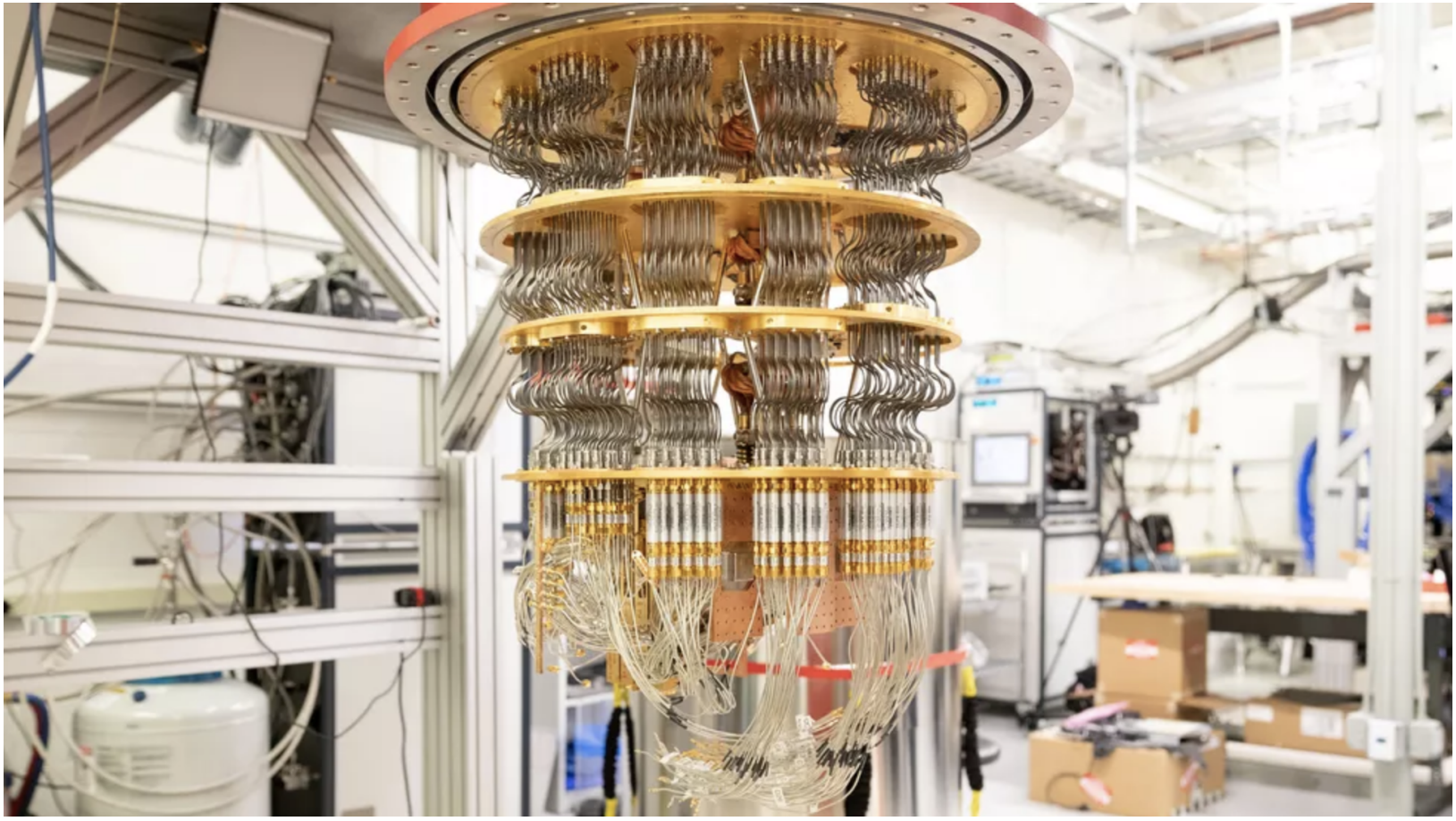
The complexity of a quantum algorithm that belongs to the search class is determined only in terms of the number of the calls to the function f.

Grover's algorithm is a search algorithm based on the call of a general function f

In quantum algorithms of this type we compute the **query complexity** rather than the number of elementary instructions executed by the program

Thanks to this different definition we will be able to describe through the 3 steps of Grover's search how a **quadratic** speedup is achieved w.r.t. a classical search algorithm

These conclusions allow to formally define the *quantum machine paradigm* to be used in order to implement quantum algorithms



A quantum computer is not that different from how we consider a classical one in a general point of view

The model of computation we considered to formalize this paradigm is the **quantum circuit model**, which works as follows:

- The quantum computer has a **state** that is stored in a quantum register, initialized in a certain way at the beginning of the computation

A quantum computer is not that different from how we consider a classical one in a general point of view

The model of computation we considered to formalize this paradigm is the **quantum circuit model**, which works as follows:

- The quantum computer has a **state** that is stored in a quantum register, initialized in a certain way at the beginning of the computation
- **Quantum operations** applied on a state allow the quantum computer to evolve from a state to another

A quantum computer is not that different from how we consider a classical one in a general point of view

The model of computation we considered to formalize this paradigm is the **quantum circuit model**, which works as follows:

- The quantum computer has a **state** that is stored in a quantum register, initialized in a certain way at the beginning of the computation
- **Quantum operations** applied on a state allow the quantum computer to evolve from a state to another
- At the end of the computation the information stored in the quantum register, thus the final state, contains the **result**

Grover's Algorithm

Grover's algorithm is a search algorithm studied for database lookups based on the quantum principle called **Amplitude Amplification**

Basic Idea: *Start with the uniform superposition of all basis states, and iteratively increase the coefficients of basis states that correspond to binary strings for which an unknown function gives output 1.*

The algorithm requires $q = n + 1$ qubits but we will need more to implement it in the generalized solver for the **exactly-1 k-SAT**

Outline: *Obtained the uniform superposition of the quantum state the following steps are iteratively repeated:*

- Flip the sign of the vectors for which U_f gives output 1
- Invert all the coefficients of the quantum state around the average coefficient

SAT

The SATisfiability problem

Problem Definition

Let $X \equiv \{x_1, x_2, ..., x_n\}$ be a set

Then x_k and its negations $\overline{x}_k$ are called literals and the set of all literals is denoted by $X^{'} = \{x_1, \overline{x}_1, ..., x_n, \overline{x}_n\}$

The set of all subsets of $X^{'}$ is denoted by $\mathcal{F}(X^{'})$ and an element $C \in \mathcal{F}(X^{'})$ is called a clause

SAT problem: *Given a set* $X = \{x_1, x_2, ..., x_n\}$ *and a set of clauses* $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ *of clauses, determine whether* $\mathcal{C}$ *is satisfiable or not.*

In the literature we find the problem typically identified as k-SAT where we need to consider:

1. The number of variables on which the problem is defined: **n**
2. The number of clauses on which the problem is defined: **m**
3. The maximal length of the clauses in the problem: **k**

SAT example

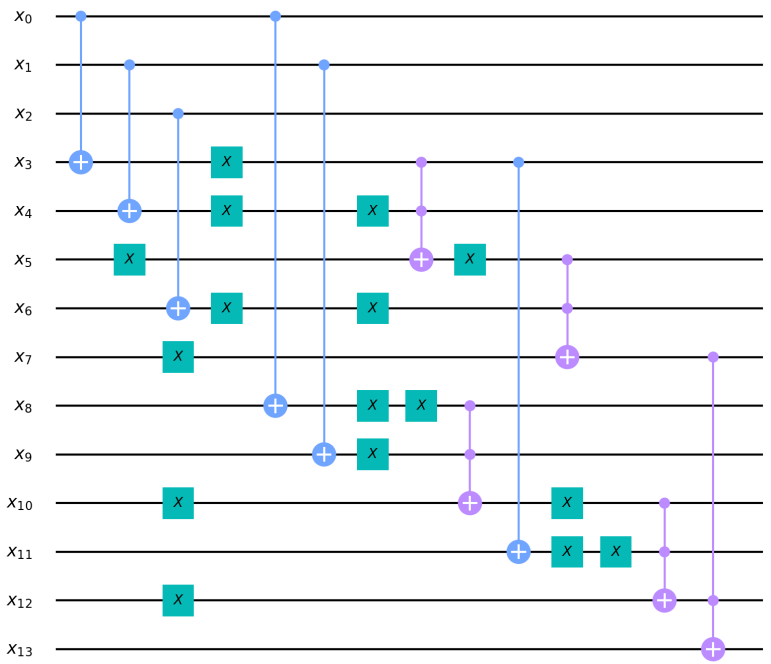
Consider the following two clauses defining a 3-SAT problem over the variables x_1, x_2, x_3:

$$C_1 = \{\overline{x}_1, x_2, x_3\} \qquad C_2 = \{x_1, \overline{x}_2, x_3\}$$

Corresponding to the *Conjunctive Normal Form*:

$$CNF = (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3)$$

That we represented in a quantum circuit, by exploiting the relations between the quantum and classical operations, as represented in the picture on the right



Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ [**Det**]

Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ **[Det]**
- [Makino, Tamaki and Yamamoto, 2013] $\mathcal{O}(1.3303^n)$ **[Der]**

Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ **[Det]**
- [Makino, Tamaki and Yamamoto, 2013] $\mathcal{O}(1.3303^n)$ **[Der]**
- [Iwama, Seto Takai and Tamaki, 2010] $\mathcal{O}(1.32113^n)$ **[Rand]**

Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ [**Det**]
- [Makino, Tamaki and Yamamoto, 2013] $\mathcal{O}(1.3303^n)$ [**Der**]
- [Iwama, Seto Takai and Tamaki, 2010] $\mathcal{O}(1.32113^n)$ [**Rand**]
- [Hertli, Moser and Scheder, 2010] $\mathcal{O}(1.32065^n)$ [**Rand**]

Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ **[Det]**
- [Makino, Tamaki and Yamamoto, 2013] $\mathcal{O}(1.3303^n)$ **[Der]**
- [Iwama, Seto Takai and Tamaki, 2010] $\mathcal{O}(1.32113^n)$ **[Rand]**
- [Hertli, Moser and Scheder, 2010] $\mathcal{O}(1.32065^n)$ **[Rand]**
- [Hertli, 2011] $\mathcal{O}(1.30704^n)$ **[Rand]**

Classical Solver

Classical solvers for the SAT problem are widely studied for their importance in applications, in particular **Artificial Intelligence**

Deterministic and *Nondeterministic* algorithms are used to solve SAT instances, with a focus in particular for the 3-SAT problem:

- [Kutzkov and Scheder, 2010] $\mathcal{O}(1.439^n)$ **[Det]**
- [Makino, Tamaki and Yamamoto, 2013] $\mathcal{O}(1.3303^n)$ **[Der]**
- [Iwama, Seto Takai and Tamaki, 2010] $\mathcal{O}(1.32113^n)$ **[Rand]**
- [Hertli, Moser and Scheder, 2010] $\mathcal{O}(1.32065^n)$ **[Rand]**
- [Hertli, 2011] $\mathcal{O}(1.30704^n)$ **[Rand]**

In our study we considered an efficient classical search algorithm based on backtracking, able to solve the general **k-SAT** algorithm

To face the problem of solving the SAT in the quantum world we realized **3** implementations:

1. **Qiskit's general solver** (Jupyter Notebook)
2. **Decision Version**
3. **Generalized exactly-1 k-SAT solver**

We now focus on describing how the general solver for the **Exactly-1 k-SAT** problem works, considering an example to go through the 3 steps of **Grover's algorithm**

Then a short **demo** on how to run the solver on the same example will prove the consistency of what has been described

Exactly-1 k-SAT solver

Definition: *Given a k-SAT problem we want to determine a satisfying assignment containing one true literal per clause.*

We realized the generalized version provided in [Nannicini, 2020] now able to solve the **exactly-1 k-SAT** problem with an arbitrary number of variables, clauses and in particular k

We implemented the solver using **Grover's search** algorithm to find the satisfiable solution. We now show how the three steps are generalized for the case of a 4-SAT problem over $n = 4$ variables and $m = 4$ clauses

Problem 4

$X = \{x_1, x_2, x_3, x_4\}$

$C = \{C_1, C_2, C_3, C_4\}$

Such that:

$C_1 = \{x_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\}$

$C_2 = \{\overline{x}_1, x_2, \overline{x}_4\}$

$C_3 = \{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4\}$

$C_4 = \{\overline{x}_2, x_3, \overline{x}_4\}$

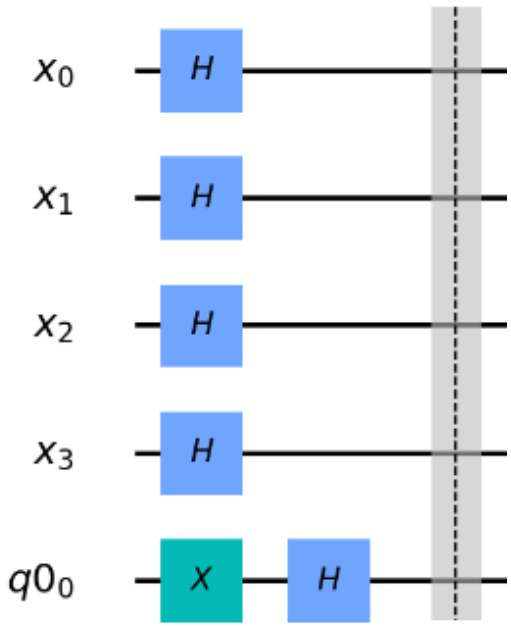
Initialization

Grover's algorithm applies to a function with an n-qubit input and a single qubit output

In this case we have **n=4** inputs and always a single output, where:

- `f_in` is the input register of the encoded function U_f, of size 4
- `f_out` is the output register of U_f, of size 1

The initialization of the algorithm consists of bringing the state in the **uniform superposition**. This is achieved by applying an Hadamard gate on all the 4 input qubits and an X gate followed by another Hadamard for the output.



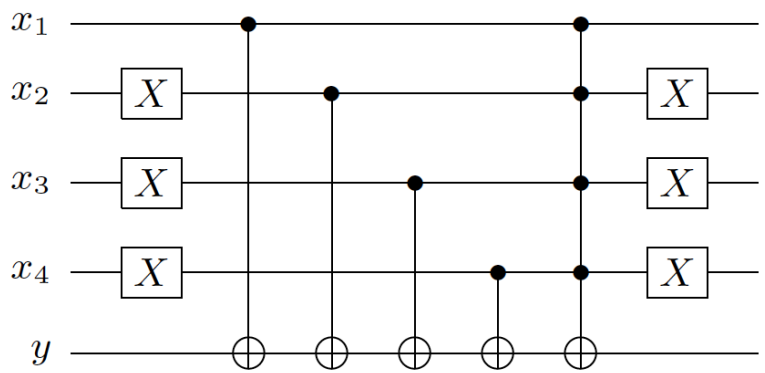
Black-box function U_f

Implementing U_f is the most difficult step of the algorithm

The problem of computing U_f is decomposed for each clause by introducing **m** auxiliary qubits

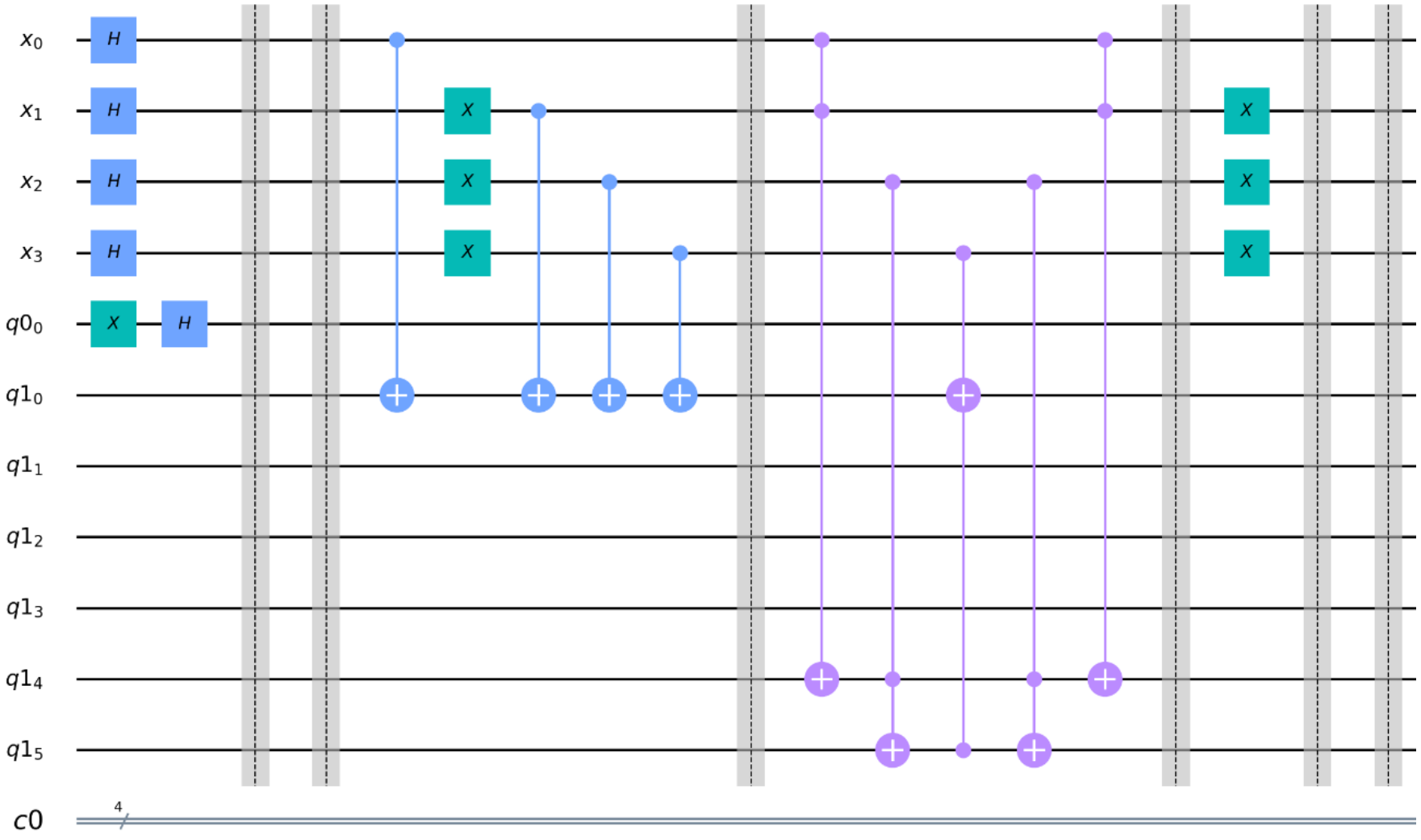
For each **clause** we build a circuit that bit-flips the output register of U_f if and only if all the m auxiliary qubits are equal to 1

Considering the clause C_1 of our example we want to realize the following circuit:



Unfortunately Qiskit provides only doubly-controlled NOT gates...

Qiskit solution



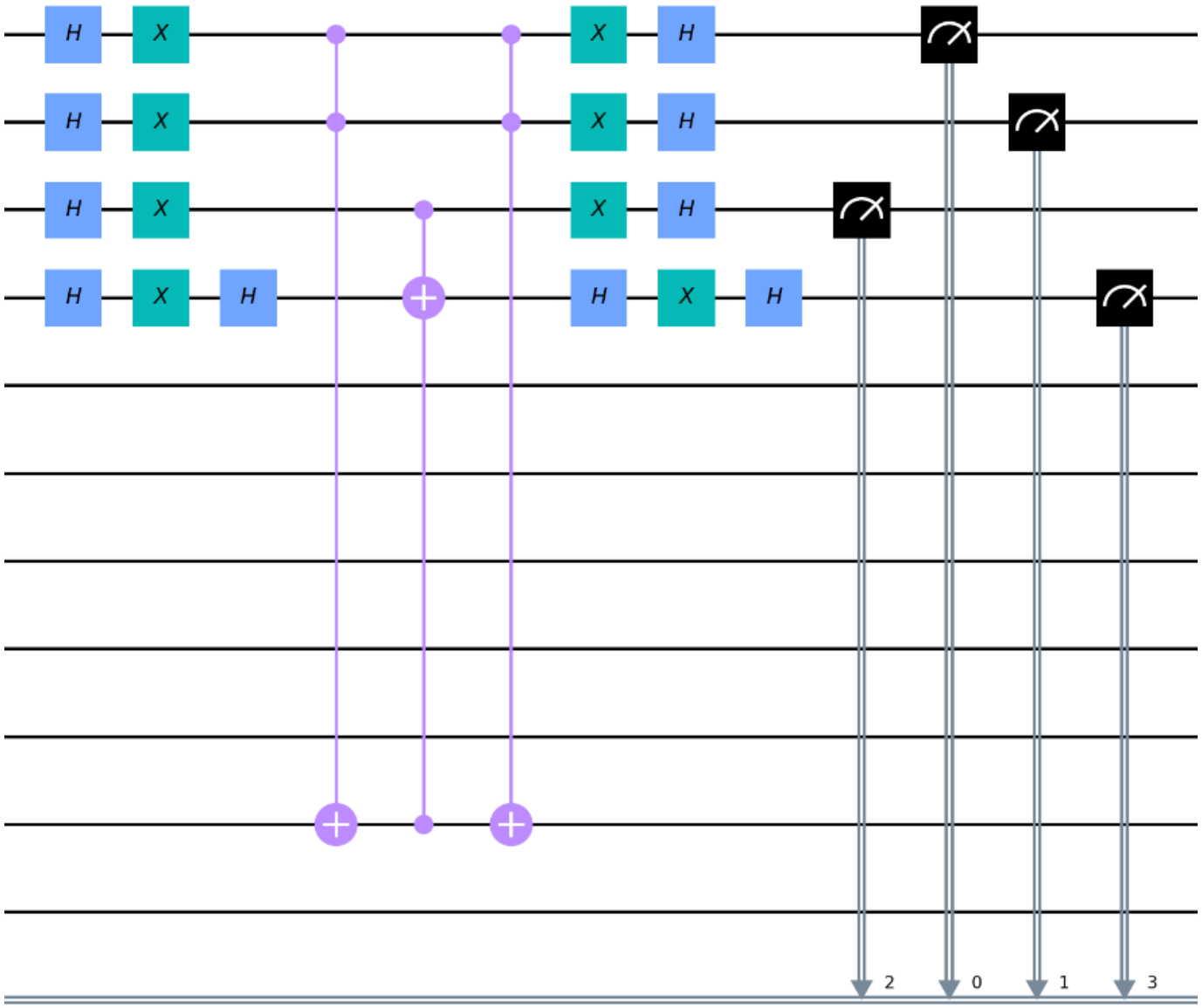
The last step of **Grover's algorithm**, updating once more the coefficients of the quantum state storing the solution, can be described with a unitary matrix called W

The matrix W is performed over the quantum state and is defined in general as:

$$W = (- \otimes^n \mathcal{H}) D (\otimes^n \mathcal{H})$$

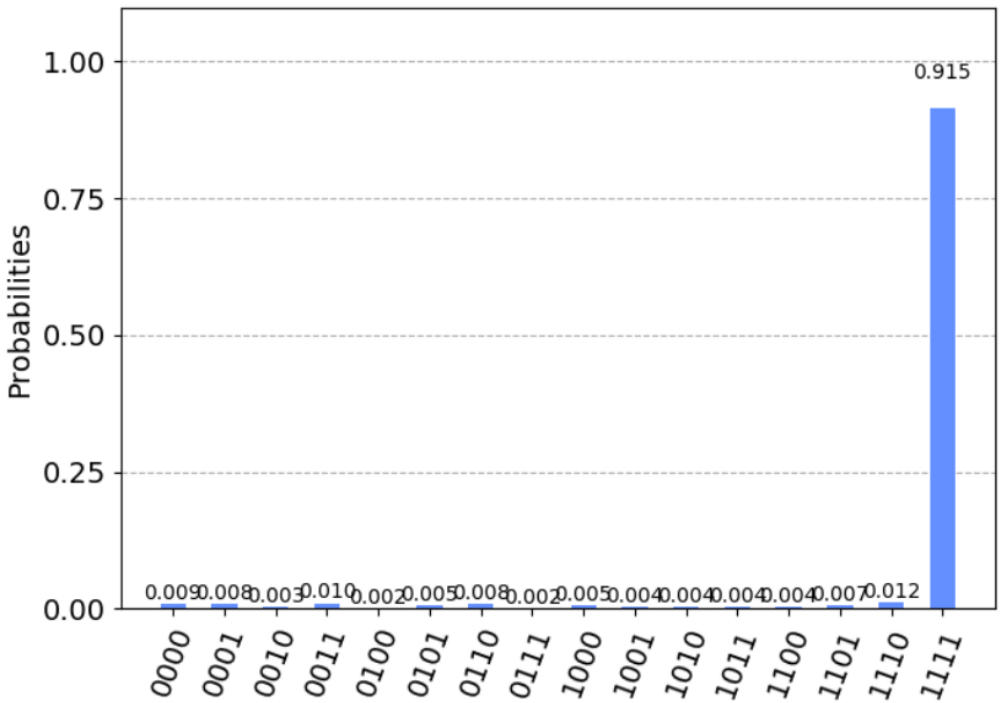
Also in this case we have the problem of realizing a **multiply-controlled NOT** gate, for the diagonal matrix D; thus a $C^{n-1}Z$ gate. In our case, with $n = 4$, we will need to consider an additional ancillary qubit to implement the C^3Z gate.

Qiskit Solution



Problem 4 Result

By executing the solver for the **exactly-1 Problem 4** we obtain the following solution:



Let's now do the demo to show the solver in action for the problem we considered

- **Complexity**
  - **Space Complexity**
    Gates and depth explosion in Qiskit solver
    Exponential growth of qubits to represent the quantum state
    No-cloning principle
  - **Time Complexity**
    QFT approach may lead to the number of elementary instructions
    Grover's search provides a ***quadratic speedup***
- **Implementation:**
  Classical and quantum algorithms do not have an exact correspondence
  We have a ***zoo*** *of quantum algorithms to be reused in order to solve our problem*
  The ***programming paradigm*** *is completely changed*

The algorithm we introduced for the classical solvers run in time

$$\mathcal{O}(T(n)poly(n))$$

where $T(n)$ is an exponentially growing function of n. These algorithms are based on a **polynomial-time** algorithm that outputs the satisfying assignments, if one exists, with probability at least $1/T$.

To make a SAT problem satisfiable we will run the algorithm $\mathcal{O}(T(n))$ times

With Grover and in particular **Amplitude Amplification** we can decide if an algorithm accepts with probability 0 or $1/T$ running it only the $\mathcal{O}(\sqrt{T(n)})$ of the times.

In conclusion applying Grover to a k-SAT solver brings to the **quadratic speedup** we promised:

$$\mathcal{O}(\sqrt{T(n)}poly(n))$$

1. **Gap** between the quantum implementation and its real deploy on a device

1. **Gap** between the quantum implementation and its real deploy on a device
2. **Exponential growth** of the quantum state

Conclusions

1. **Gap** between the quantum implementation and its real deploy on a device
2. **Exponential growth** of the quantum state
3. **Theoretical** time speedups

Conclusions

1. **Gap** between the quantum implementation and its real deploy on a device
2. **Exponential growth** of the quantum state
3. **Theoretical** time speedups
4. **Zoo** of quantum algorithms to solve our problems

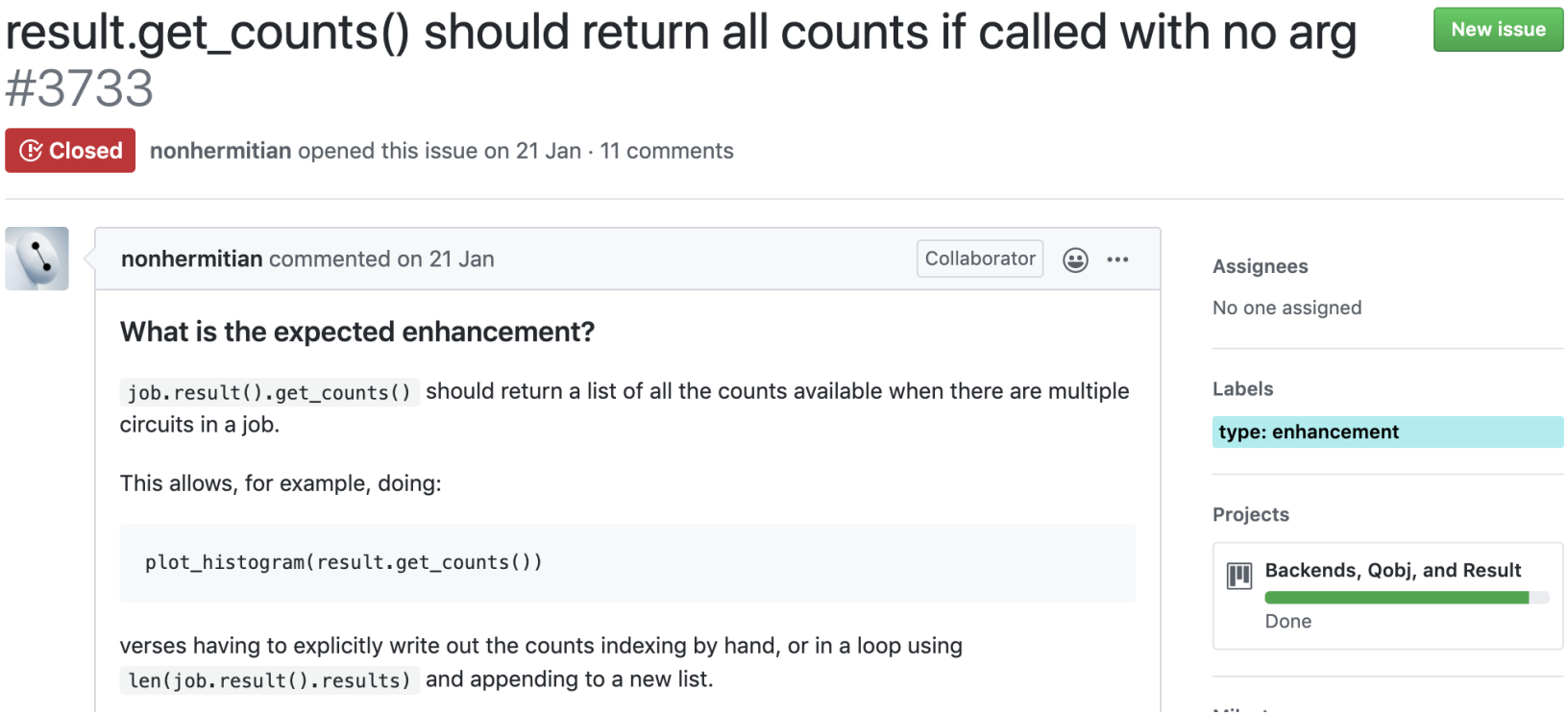
Conclusions

1. **Gap** between the quantum implementation and its real deploy on a device
2. **Exponential growth** of the quantum state
3. **Theoretical** time speedups
4. **Zoo** of quantum algorithms to solve our problems
5. The **role** of the computer scientist

Conclusions

1. **Gap** between the quantum implementation and its real deploy on a device
2. **Exponential growth** of the quantum state
3. **Theoretical** time speedups
4. **Zoo** of quantum algorithms to solve our problems
5. The **role** of the computer scientist
6. New **programming paradigm**

I became contributor of the **Qiskit library** by solving the following issue



result.get_counts() should return all counts if called with no arg #3733

New issue

Closed nonhermitian opened this issue on 21 Jan · 11 comments

nonhermitian commented on 21 Jan — Collaborator

What is the expected enhancement?

`job.result().get_counts()` should return a list of all the counts available when there are multiple circuits in a job.

This allows, for example, doing:

```
plot_histogram(result.get_counts())
```

verses having to explicitly write out the counts indexing by hand, or in a loop using `len(job.result().results)` and appending to a new list.

Assignees

No one assigned

Labels

type: enhancement

Projects

Backends, Qobj, and Result

Done

Link to the **Pull Request**

Thanks for your Attention