# KAKURO: A CONSTRAINT SATISFACTION PROBLEM

*Marcelin Gallezot - Tinubu Square*

## 1. INTRODUCTION

This problem deals with the solving of a Kakuro (a Japanese logic puzzle often referred to as the mathematical transliteration of the crossword) with a quantum computer. This puzzle translates to a constraint satisfaction problem that will be solved using Grover's quantum search algorithm [1]. The challenge consists in designing a Grover oracle with a minimum number of CX gates (only single-qubit gates and CX gates are allowed).

## 2. GROVER'S ALGORITHM

### 2.1. Quantum Search Problem

Suppose you have to find one or multiple solutions to a particular problem within N elements. A classical computer would have to go through each single one of these elements to find the solutions, taking $O(N)$ operations. A quantum computer on the other hand would require only $O(\sqrt{N})$ operations thanks to Grover's algorithm.

### 2.2. The Algorithm

#### *2.2.1. Grover's Oracle*

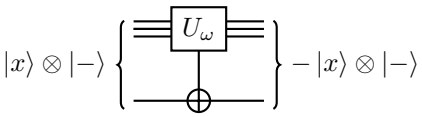
Grover's algorithm works with a so-called oracle, a blackbox that will flip the amplitude sign of the solution state $\omega$. For an input state $|x\rangle$, the action of the oracle can be interpreted as a unitary $U_\omega$ :

$$U_\omega |x\rangle = \begin{cases} |x\rangle & \text{if } x \neq \omega \\ -|x\rangle & \text{if } x = \omega \end{cases}$$

In a context of constraint satisfaction, to verify a solution of a search problem, we have to check that a certain amount of constraints are satisfied. To do so, we can create a function $f$ that will return 0 if $x \neq \omega$ and 1 if $x = \omega$. Our oracle can then be rewritten:

$$U_\omega |x\rangle = (-1)^{f(x)} |x\rangle$$

The following circuit implements a Grover oracle. A second register is added to the circuit and initialized in state $|-\rangle$. The sign of the global amplitude will be switched if and only if all the constraints of the problem are verified (ie $x = \omega$).




#### *2.2.2. Amplitude Amplification*

Now that we have a quantum circuit that is able to "recognize" and isolate a solution, we need to retrieve this solution. A Hadamard gate is applied to each qubit so the system falls in a balanced superposition of all possible states. Among all theses states, the oracle flips the sign of the solution. Then, thanks to a reflection around the mean amplitude, the solution is amplified compared to the other states. This operation is achieved by a unitary called diffuser. The oracle/diffuser cycle must be repeated $O(\sqrt{N})$ times in order for the solution to emerge.

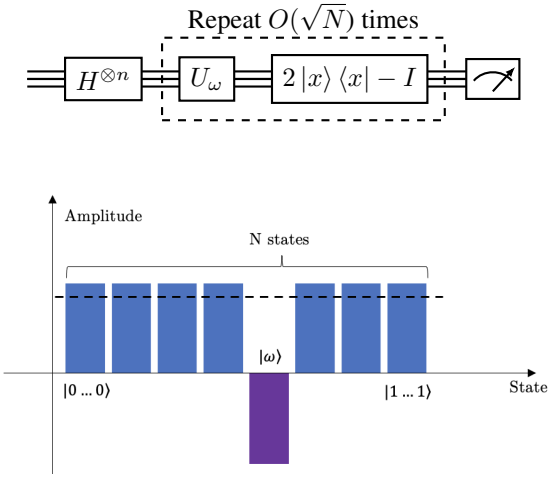



**Fig. 1**. Amplitude of solution state is flipped by the oracle

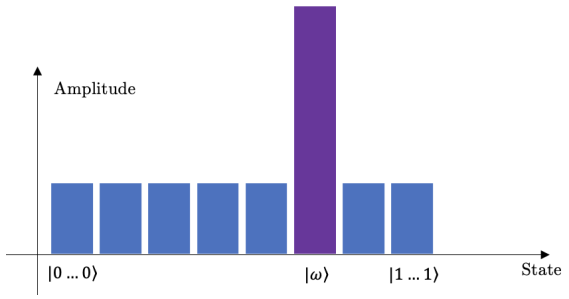



**Fig. 2**. The diffuser induces a reflection around the means, bringing forward the solution state

### 2.3. Implementation for the Kakuro Problem

To do so, we use 3 registers of qubits :

i. $|x_i\rangle_{i\in 1...n}$ : contains the variables of the problem (i.e. the values of the Kakuro cells).

ii. $|a_j\rangle_{j\in 1...m}$ : serve as ancillary to verify the constraints (the verification of these constraints is stored in a subset $|c_k\rangle_{k\in 1...p}$). This register is initialized in state $|0^{\otimes m}\rangle$.

iii. $|t\rangle$ : only one qubit initialized in state $|-\rangle$. If all the constraints are verified, a multi-controlled Toffoli gate will add a global -1 amplitude, achieving the oracle's purpose.

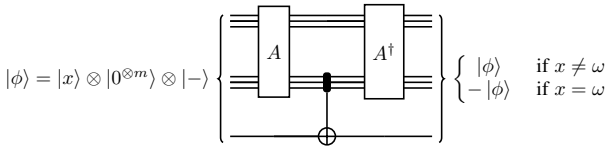



**Fig. 3**. Circuit for a constraint satisfaction problem

Because the oracle must be applied successively $O(\sqrt{N})$ times, it must not change the state of the different qubits (except for a potential global phase). Our circuit can thus be divided into three parts:

1. Encoding (matrix $A$): where each constraint of the system is verified through arithmetic operations on the $|x_i\rangle$ and $|a_j\rangle$ and stored in the $|c_k\rangle$.
2. A Multi-Control Toffoli gate that will verify if all the constraints $|c_k\rangle$ are simultaneously verified.
3. Decoding (matrix $A^\dagger$): which brings back each qubit at its original $|0\rangle$ state.

## 3. ORACLE DESIGN

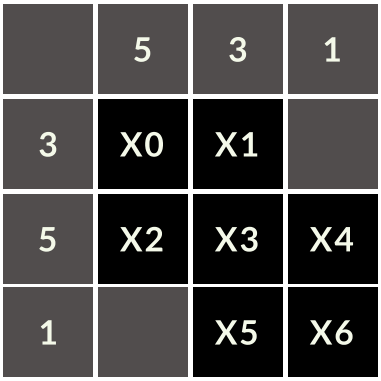



**Fig. 4**. The considered Kakuro problem.

### 3.1. Values encoding

The original constraints of the problem were optimized and reduced to a list of 11. Each variable $x_i$ is supposed to be between 0 and 3 and should therefore require two qubits for encoding (for a total of 14 qubits). However, the constraints of the problem allow us to deduct that only $x_3$ needs two qubits since $x_1$, $x_4$, $x_5$, $x_6 \in \{0, 1\}$ and $x_0$, $x_2 \in \{2, 3\}$. Because $x_0$ and $x_2$ will be encoded in one qubit, 2 will have to be added to the qubit value in order to obtain the true value. This allows us to reduce the number of qubits in the first register to 8 : $|x\rangle = |x_0x_1x_2x_{3,0}x_{3,1}x_4x_5x_6\rangle$

### 3.2. Constraints

To build the oracle, we will use elementary blocks : each constraint will be checked independently and then aggregated with a multi-controlled Toffoli gate. The various constraints of the problem can be divided into subcategories:

A. Inequalities between 1-qubit values :

   a. $x_0 \neq x_1$
   b. $x_0 \neq x_2$
   c. $x_1 \neq x_5$
   d. $x_4 \neq x_6$
   e. $x_5 \neq x_6$

B. (In)equalities involving 2-qubits values :

   a. $x_3 = 2$

b. $x_3 \neq x_1$

c. $x_3 \neq x_4$

d. $x_3 \neq x_5$

C. Sums :

a. $x_2 + 2 \neq x_3$

b. $x_2 + x_4 + x_3 = 3$

To each of these 11 constraints, we assign a given number of ancillary qubits that will be used to verify them and a register $|c_k\rangle_{k\in 1...11}$ to store the boolean associated with the verification of each constraint.

### *3.2.1. Inequalities between 1-qubit values*

The circuit used to check these constraints is simply the quantum XOR gate which uses 2 CX gates and an ancillary qubit that will be used to store the result. A simpler circuit uses only 1 CX as the result is stored in one of the two information qubits. Since one of the two qubit is altered, we have to make sure it is not used later in the oracle.

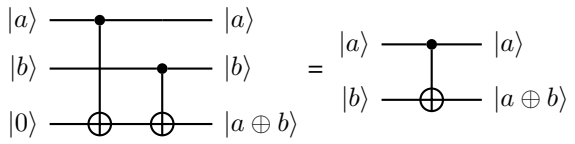
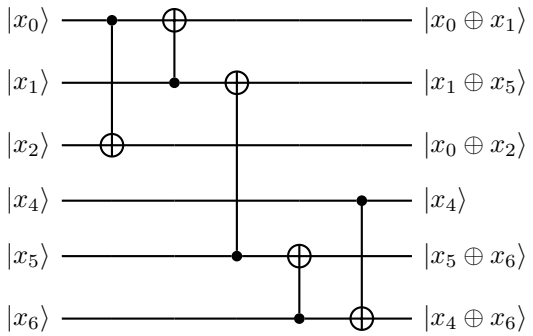



**Fig. 5**. Quantum XOR gate (left) and simplified (right).

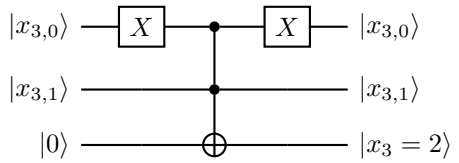
In order to check inequalities A.a. to A.e., we can use the following circuit that requires 5 CX gates and leaves 5 conditions to be checked.




### *3.2.2. (In)equalities involving 2-qubits values*

#### *3.2.2.1. Constraint B.a*

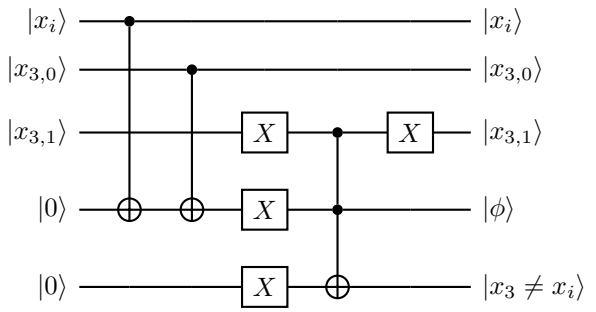
To check the constraint $x_3 = 2$, we must check that $x_{3,0} = 0$ (LSB) and $x_{3,1} = 1$ (MSB). This is achieved with the following circuit:



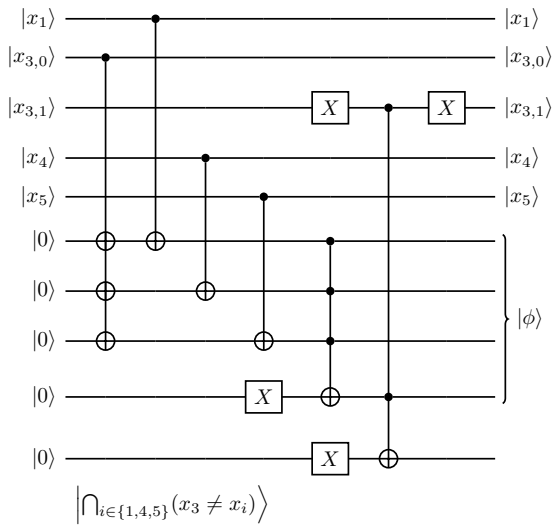

This circuit requires 1 Toffoli gate (= 6 CX gates) and one condition to check.

#### *3.2.2.2. Constraints B.b, B.c and B.d*

Here, we are dealing with inequalities of the form $x_3 \neq x_i$ (with $i \in \{1, 4, 5\}$). Because $x_3$ can take any value between 0 and 3 and $x_i$ only 0 and 1, the inequality is equivalent to $(x_{3,1} = 1)$ or $(x_{3,0} \neq x_i)$.




The circuit above implements this test with 2 CX gates and 1 Toffoli (so a total of 8 CX). If we multiply by 3 for $i \in \{1, 4, 5\}$, that makes a total of 24 CX and 3 conditions to check. However, this might be redundant since we would verify 3 times that $x_{3,1} = 1$. Because $\bigcap_{i\in\{1,4,5\}}(x_{3,1} = 1) \vee (x_{3,0} \neq x_i)$ is equivalent to $(x_{3,1} = 1) \vee \bigcap_{i\in\{1,4,5\}}(x_{3,0} \neq x_i)$, the circuit can be simplified :

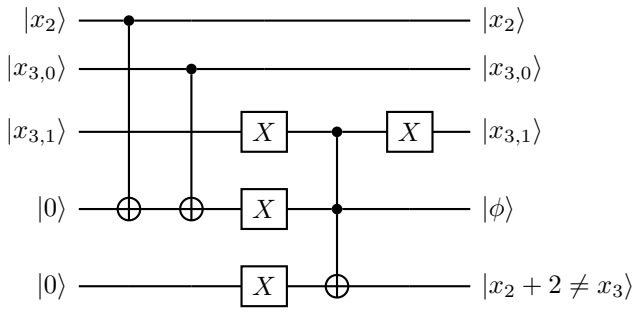



This new circuit uses the same technique for compacting XOR gates as in 3.2.1. It uses 6 CX gates, one Toffoli gate and one 3-fold Toffoli. This circuit may seem more expensive than the one before but he leaves only one condition to check and we will see later than it can be implemented with only 15 CX gates (using relative phase Toffolis).

### *3.2.3. Sums*

#### *3.2.3.1. Constraint C.a*

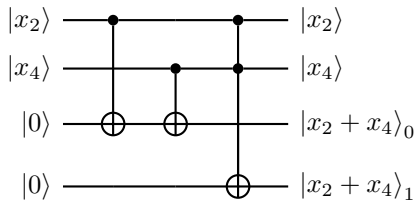
$x_2 + 2 \neq x_3$ translates to $(x_{3,1} \neq 1)$ or $(x_{3,0} \neq x_2)$ which can be implemented with the following circuit:



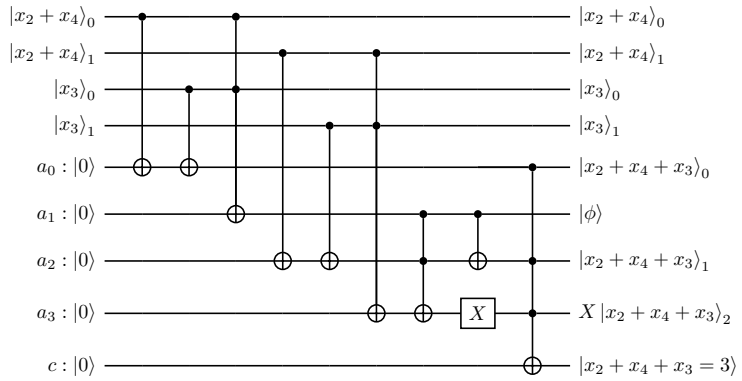

This circuit uses $2 + 6 = 8$ CX gates and one condition qubit.

#### *3.2.3.2. Constraint C.b*

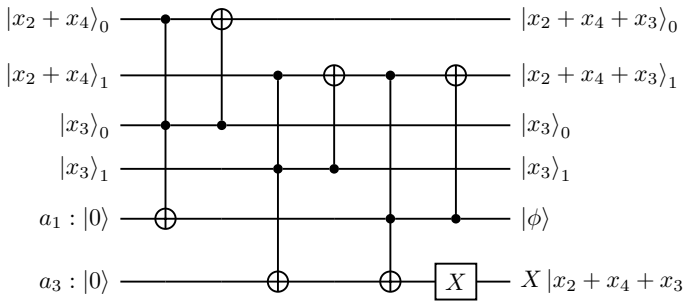
To verify the constraint $x_2 + x_4 + x_3 = 3$, we will first compute the sum $x_2 + x_4$. An arithmetical sum of 2 qubits is rather easy to achieve (the result is stored in two qubits as the sum can go up to the value of 2).




Once $x_2 + x_4$ computed, adding $x_3$ is more complicated because the result can go up to 5 (and thus needs 3 qubits to be stored). There are also more carries to take into account. In the circuit below, ancillary $a_0$ stores the least significant bit of the total sum, $a_1$ serves as carry for the second bit, $a_2$ will store the value of the second bit and $a_3$ the value of the most significant bit. Condition qubit $c$ verifies that the total sum is equal to 3.




Because the sum $x_2 + x_4$ will not be used for further computation, we can re-use the two qubits to compute the sum $x_2 + x_4 + x_3$ like we simplified the XOR gates for constraints A. This allows us to save two CX gates and two ancillary qubits. Instead of computing the test $x_2 + x_4 + x_3 = 3$ straight away, we can leave the values in the ancillary qubits and check it in the final multi-controlled Toffoli. We will see that this saves one ancillary qubit and does not increase the total number of CX gates.



The circuit requires 3 CX gates and 3 Toffolis. We must add the two CX and the Toffoli of the circuit used to compute $x_2+x_4$. That makes a total of $3+3*6+2+6 = 29$ CX gates and 3 conditions to check.

### 3.3. Encoding circuit

| | Condition | CX count | Conditions | Ancillae |
|---|---|---|---|---|
| **A** | **A.a**: $x_0 \neq x_1$<br>**A.b**: $x_0 \neq x_2$<br>**A.c**: $x_1 \neq x_5$<br>**A.d**: $x_4 \neq x_6$<br>**A.e**: $x_5 \neq x_6$ | 5 | 5 | 0 |
| **B** | **B.a**: $x_3 = 2$ | 6 | 1 | 1 |
| | **B.b**: $x_3 \neq x_1$<br>**B.c**: $x_3 \neq x_4$<br>**B.d**: $x_3 \neq x_5$ | 24 | 1 | 5 |
| **C** | **C.a**: $x_2 + 2 \neq x_3$ | 8 | 1 | 2 |
| | **C.b**: $x_2 + x_4 + x_3 = 3$ | 29 | 3 | 4 |
| **Total** | | **72** | **11** | **12** |

**Table 1**. Encoding circuit summary table.

By assembling the various building blocks designed above, we can build our encoding circuit. This encoding circuits required 12 additional ancillae and a total of 72 CX gates, 11 conditions will have to be checked by the multi-controlled Toffoli. That makes a total of 72*2 = 144 for encoding and decoding plus the 11-fold Toffoli gate. How can we reduce this number ?

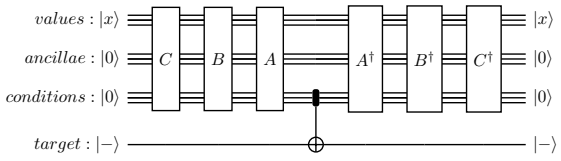



**Fig. 6**. Schematic diagram of the oracle with elementary blocks A, B and C.

## 4. OPTIMIZATION

### 4.1. XOR compacting

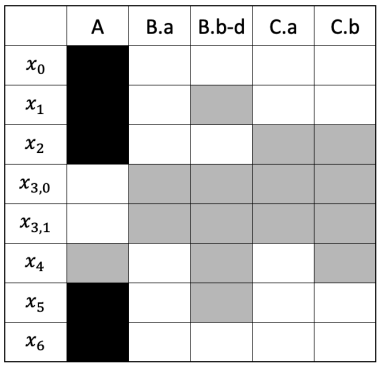



**Fig. 7**. Qubit manipulations in the elementary blocks. A black cell means that the qubit state is altered during the process and a grey cell that the qubit serves as control.

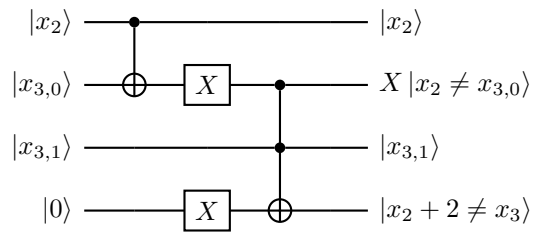
The first approach we can follow to reduce the number of CX gates in the circuit is the one we used for constraints A.a to A.e (i.e. to directly store the result of an operation in an existing value qubit instead of introducing ancillary qubits). However, we must be careful not to alter the state of a qubit that would be used for further computations.

Fig.7 shows which qubits are manipulated in the different building blocks of the oracle. The first obvious point is that the blocks altering qubit states must be placed after the one using those qubits in the encoding circuit. Then, we see that there are still 3 qubits ($x_{3,0}$, $x_{3,1}$ and $x_4$) that are not altered in any step of the circuit and could be used to reduce the CX gates count.

- Because block A alters 5 qubits, it will have to be placed last in the encoding circuit. This blocks the possibility of altering $x_4$ before in the circuit.

- Qubit $x_{3,1}$ is not involved in XOR operations so we won't be able to use it in order to reduce the CX count either.

- We can however transform circuit C.a to save a CX gate and an ancillary qubit with the use of $x_{3,0}$:




### 4.2. Relative phase Toffolis

The simplified Toffoli [2] was introduced by N. Margolus in 1994. It realizes the same unitary map as the Toffoli gate except for the state $|101\rangle$ that is mapped to $-|101\rangle$ instead of $|101\rangle$ .

$$
\begin{aligned}
|00\rangle \otimes |\phi\rangle &\mapsto |00\rangle \otimes |\phi\rangle \\
|01\rangle \otimes |\phi\rangle &\mapsto |01\rangle \otimes |\phi\rangle \\
|10\rangle \otimes |\phi\rangle &\mapsto |10\rangle \otimes Z\,|\phi\rangle \\
|11\rangle \otimes |\phi\rangle &\mapsto |11\rangle \otimes X\,|\phi\rangle
\end{aligned}
$$

The main advantage of the Margolus gate is that it uses only 3 CX gates when the best Toffoli gate implementation uses 6 CX gates. This number of 3 CX gates is optimal [3]. Since phase is important in our oracle, we have to pay attention to the negative phases it might introduce. Nevertheless, because the gates used in the encoding circuit are uncomputed during the decoding part, the phases will cancel out. We can use a similar simplified gate that exists for the 3-fold Toffoli.

Table.2 shows that the use of simplified Toffolis (RCCX and RCCCX) allows us to save 28 CX gates in the encoding circuit (that makes a total of 56 CX gates saved when adding the decoding circuit).

### 4.3. Constraints verification with a Multi-Control Toffoli gate

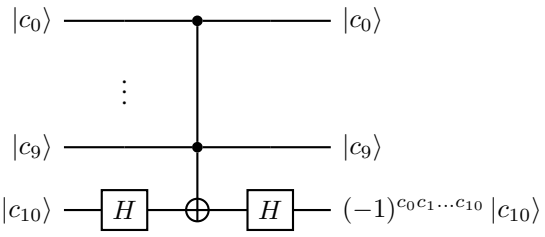
So far, we have reduced the number of CX gates in the encoding circuit but we did not mention how many CX gates were necessary to perform the 11-fold Multi-Control Toffoli (MCT) that will flip the sign of the amplitude if and only if all 11 constraints are satisfied. Yet, the number of CX in such a gate is material. For example, the qiskit implementation of the 11-fold MCT with no additional ancillary qubits uses up to 6140 CX gates. Such a number would make the few optimizations we made above useless. Fortunately, there are other ways to implement such a gate.

| | Condition | CX count | |
|---|---|---|---|
| | | **Toffolis** | **Simplified** |
| **A** | **A.a**: $x_0 \neq x_1$<br>**A.b**: $x_0 \neq x_2$<br>**A.c**: $x_1 \neq x_5$<br>**A.d**: $x_4 \neq x_6$<br>**A.e**: $x_5 \neq x_6$ | 5 | 5 |
| **B** | **B.a**: $x_3 = 2$ | 6 | 3 |
| | **B.b**: $x_3 \neq x_1$<br>**B.c**: $x_3 \neq x_4$<br>**B.c**: $x_3 \neq x_5$ | 24 | 15 |
| **C** | **C.a**: $x_2 + 2 \neq x_3$ | 7 | 4 |
| | **C.b**: $x_2 + x_4 + x_3 = 3$ | 29 | 17 |
| | **Total** | **72** | **44** |

**Table 2**. Comparison of CX gates count when using Toffolis or Margolus gates

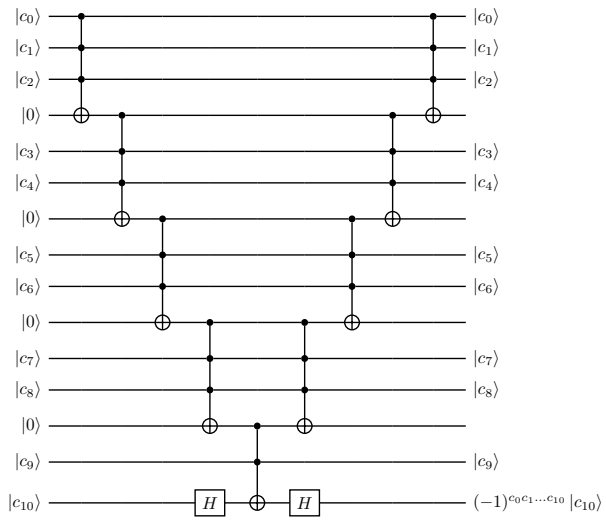
#### *4.3.1. Removing the target qubit*

We can use a more economic version [4] of Grover's algorithm by removing the target qubit and using the last condition qubit $c_{10}$ as target (and using a Multi-Control Z gate instead). By doing that, we save one qubit and find ourselves with a 10-fold MCT instead of 11.




#### *4.3.2. Implementation of 10-fold MCT*

There are several implementations of Multi-Control Toffoli gates available in the literature. We will chose one that uses the simplified Toffolis [5] introduced before. This implementation uses 8 RCCX gates and one regular Toffoli gate for a total of 54 CX gates but it needs an additional 4 clean ancillary qubits.




## 5. CONCLUSION

Now that we have an optimized encoding circuit and 10-fold MCT, we can assemble our oracle by placing successively the encoding circuit, the 10-fold MCT and the decoding circuit (which is the inverse of the encoding circuit). Our Grover oracle requires 15 additional qubits (that makes a total of 23 with the 8 qubits that store the values $x_i$). Out of these 15 qubits, 11 are used to check the constraints of the system and the 4 others are for the 10-fold Multi-Control Toffoli gate. Our oracle totals a number of $54 + 44 * 2 = 142$ CX gates. The only 8-qubit state whose amplitude is flipped by the oracle is $|x_6x_5x_4x_{3,1}x_{3,0}x_2x_1x_0\rangle = |10010110\rangle$. This state corresponds to :

$$
\begin{aligned}
|x_0\rangle = |0\rangle &\mapsto x_0 = 2 \\
|x_1\rangle = |1\rangle &\mapsto x_1 = 1 \\
|x_2\rangle = |1\rangle &\mapsto x_2 = 3 \\
|x_3\rangle = |10\rangle &\mapsto x_3 = 2 \\
|x_4\rangle = |0\rangle &\mapsto x_4 = 0 \\
|x_5\rangle = |0\rangle &\mapsto x_5 = 0 \\
|x_6\rangle = |1\rangle &\mapsto x_6 = 1
\end{aligned}
$$

We obtain the sole solution of the given puzzle.

# References


[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996.

[2] N. Margolus, *Simple quantum gates*. Unpublished manuscript, 1994.

[3] G. Song and A. Klappenecker, "The simplified toffoli gate implementation by margolus is optimal," 2003.

[4] R. Portugal, "Basic quantum algorithms," 2022.

[5] D. Maslov, "Advantages of using relative-phase toffoli gates with an application to multiple control toffoli optimization," *Physical Review A*, vol. 93, feb 2016.