Kakuro: A constraint satisfaction problem

The list of constraints is:

1. $x0 \neq x1$
2. $x2 + 2 \neq x3$
3. $x3 \neq x4$
4. $x1 \neq x3$
5. $x3 \neq x5$
6. $x5 \neq x6$
7. $x0 \neq x2$
8. $x1 \neq x5$
9. $x4 \neq x6$
10. $x3 == 2$
11. $x2 + x4 + x3 == 3$

There are 5 different types of constraints here –

- Single qubit register inequality
- Multi-qubit register inequality
- Addition of constant
- Sum of quantum registers
- Qubit register equality

Each of these constraints are addressed separately to build the oracle.

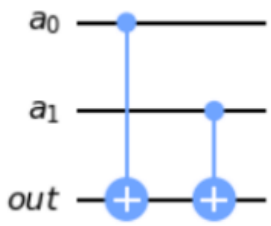
There are a total of 8 qubits required to hold the variables from x0 to x6. (Since x3 is a two qubit register and all other variables are single qubit registers.)

1 additional qubit initialized in |-> state is needed to apply a phase flip.

Qubit count: 9

1. Single qubit register inequality

   To implement an inequality constraint between two single qubit registers, it is required to ensure that both of them are distinct. This can be achieved by an **XOR** operation between them. A quantum circuit that does this, is shown below.



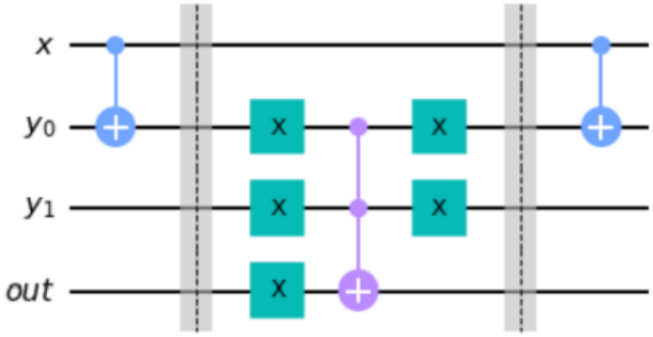
   The register – out is 1 if and only if $a0 \neq a1$. The constraints numbered 1, 6, 7, 8 and 9 are implemented this way. Since each of these requires 2 CX gates and 1 extra qubit to temporarily hold the value of the inequality check, the qubit and CX gate count stands at ***CX gate count: 5 x 2 = 10, Qubit count: 9 + 5 = 14.***

2. Multi-qubit register inequality

   To implement an inequality constraint between two registers of different sizes, it is sufficient to check the OR of inequality in each bit position. So, if a and b are two registers of sizes m and n $(m < n)$, then $a \neq b$ if

$$(a_0 \neq b_0)\ OR\ \ldots (a_{m-1} \neq b_{m-1})\ OR\ (b_m \neq 0)\ OR\ \ldots (b_{n-1} \neq 0)$$

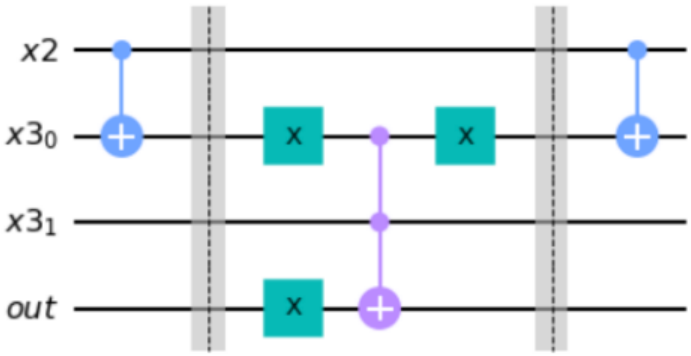
   Each individual inequality check may be implemented as previously done for the single qubit case. To note that, $XOR(y, 0) = y$. A Quantum Circuit implementing this check is shown below for two registers of sizes 1 and 2.



In the above circuit, the first CX gate implements the $XOR\ of\ x\ and\ y_0$. The result is stored in y_0. The last CX gate restores the value of y_0 by undoing the XOR operation. The part of the circuit between the two barriers implements the OR operation. The register – out is 1 if and only if $(x \neq y_0)\ OR\ (y_1 \neq 0)$. The constraints numbered 3, 4 and 5 require this implementation. Hence, the number of CX gates, CCX gates and Qubit count stands at ***CX gate count: 10 + 2*3 = 16, CCX gate count: 3 Qubit count: 14 + 1*3 = 17.***

3. Addition of constant

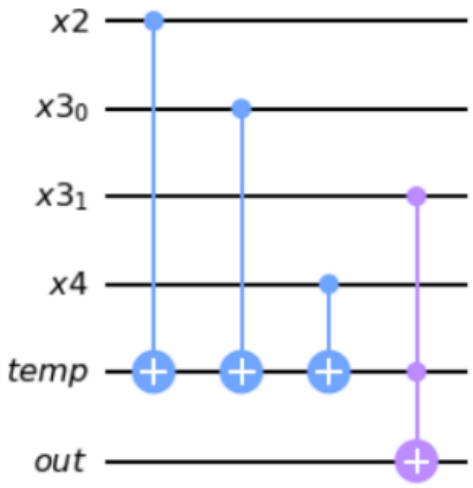
To implement the second constraint, $x2 + 2 \neq x3$, it is required that $(\boldsymbol{x2 + 0 \neq x3_0})\ \boldsymbol{OR}\ (\boldsymbol{x3_1 \neq 1})$. This is obtained by simply writing 2 in binary and applying the inequality over the individual bits (using the fact that $x2$ is a single qubit register and x3 is a two-qubit register). The circuit that implements this is shown below.



The total gate and qubit counts stand at ***CX gate count: 16 + 2 = 18, CCX gate count: 3+1 = 4, Qubit count: 17 + 1 = 18.***

4. Sum of quantum registers

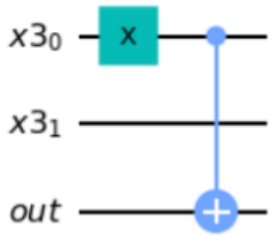
To implement the constraint $x2 + x4 + x3 == 3$, it can be broken down into two parts - $(\boldsymbol{x2 + x4 + x3_0 == 1})$ $\boldsymbol{AND}$ $(\boldsymbol{x3_1 == 1})$. This is accomplished with the below circuit.



The register – temp holds the value of $x2 + x4 + x3_0$, and the register – out holds the value of the AND operation. The total gate and qubit count stand at ***CX gate count: 18 + 3 = 21, CCX gate count: 4 + 1 = 5, Qubit count: 18 + 2 = 20.***

5. Qubit register equality

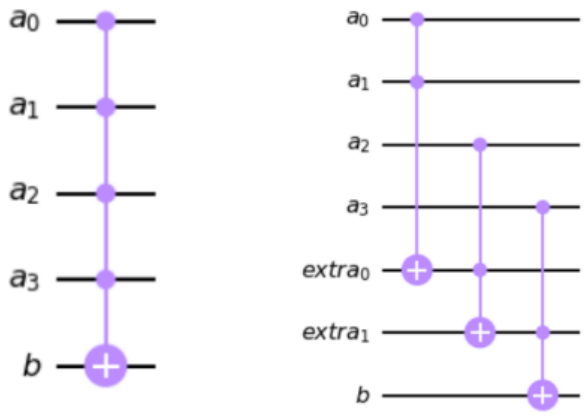
The constraint $x3 == 2$ is written as $(\boldsymbol{x3_0 == 0})$ $\boldsymbol{AND}$ $(\boldsymbol{x3_1 == 1})$. The second part of this, $x3_1 == 1$ has already been implemented above. Therefore, to implement $x3_0 == 0$, the following circuit is used.



The register – out is 1 if and only if $x3_0 == 0$. The gate and qubit count stand at ***CX gate count: 21 + 1 = 22, CCX gate count: 5, Qubit count: 20 + 1 = 21.***

Consolidating all of these constraints, it is now required to AND them together and apply a phase flip to the solution states. The straightforward way is to apply a multi-controlled (specifically, 11 qubits controlled) Toffoli gate with the target qubit being initialized in the state $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. The phase kickback mechanism marks the solution state in such a system.

However, multi-controlled Toffoli operations are expensive and one of the ways to go around this is to use extra qubits and sequentially apply a series of CCX gates.



The above two circuits are equivalent in terms of operation, but the one on the right is far less expensive than the one on the left. Therefore, to AND 11 constraints, 9 extra qubits and 10 CCX gates are required.

So, the total count is ***CX gate count: 22, CCX gate count: 5 + 10 = 15, Qubit count: 21 + 9 = 30.***

Further optimizations:

- If the constraints are re-ordered, so that 7, 8 and 9 are pushed to the end, the inequality checks can be implemented with just one CX gate than two of these gates. The registers are not used later, and so can be over-written temporarily. The over-written registers are uncomputed after the applying the phase flip.
- Each CCX gate requires 6 CX gates to be implemented. But there is a variant – the Margolus gate that implements the Toffoli gate up to a relative phase and requires only 3 CX gates, which is the minimal amount possible. It has been shown that this simplified Toffoli gate can be used in place of the CCX gate where it is uncomputed again.
- The qubit register equality check may be directly implemented without the need for an additional qubit or CX gate.

Doing these optimizations, the ***CX count: 22 – 4 = 18, CCX count: 15, Qubit count: 30 – 4 = 26.***

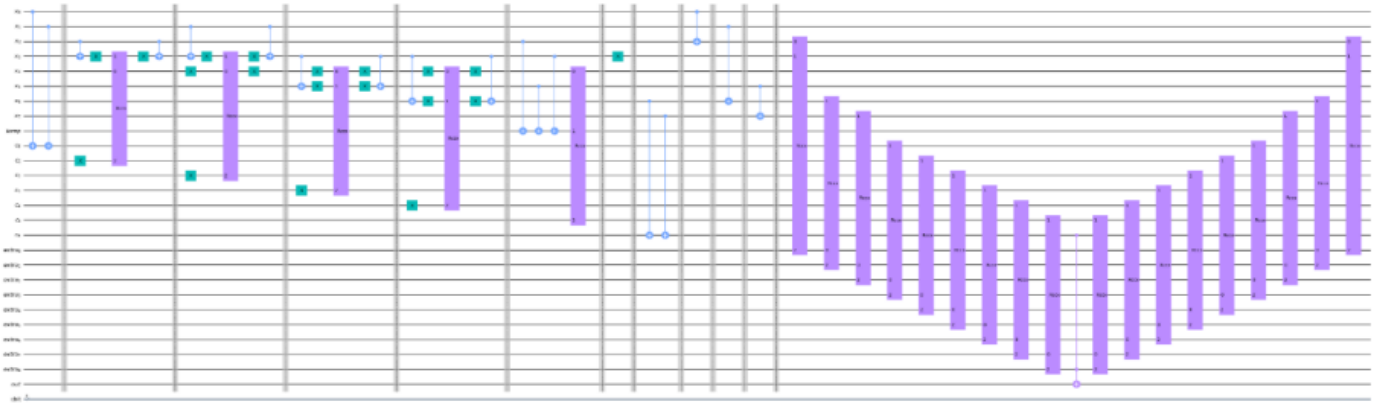
Doing uncomputation makes the total count increase by a factor of 2. ***CX count: 18 + 18 = 36, CCX count: 14 + 1 + 14 = 29, Qubit count: 26.***

Therefore, in total -

CX gates: 36 + 28*3 + 1*6 = 126 (Since one CCX gate cannot be replaced by the Margolus gate)

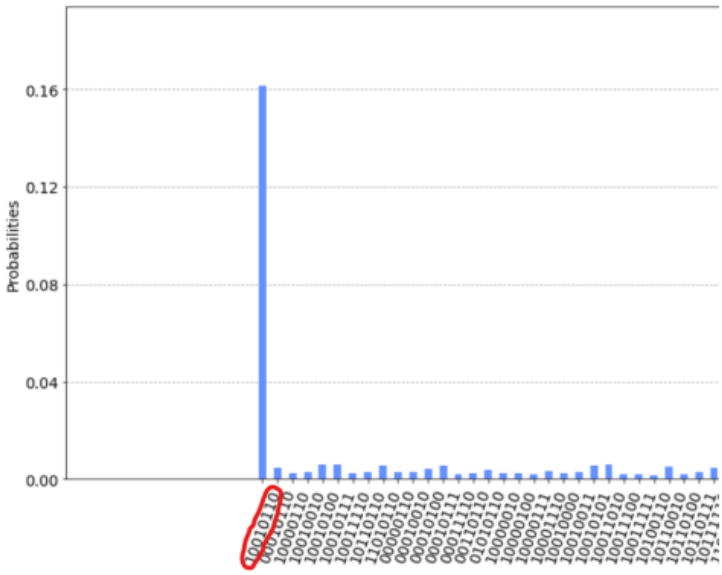
Qubits: 26

The oracle without uncomputation is shown below –



Each of the constraints are separated by a barrier. The variables are represented by $x_0 - x_7$ with x_3 *and* x_4 representing the LSB and MSB of $x3$ respectively, and subsequent variables are ordered in the standard order.

A snapshot of the probability histogram is also shown below –



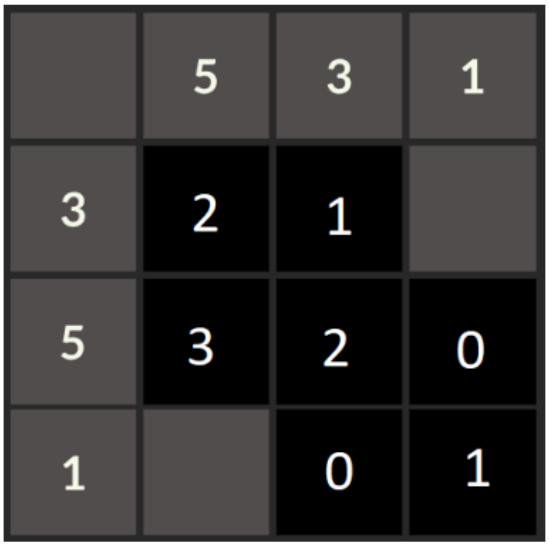
The ordering in qiskit is reversed. So, the solution is read as –

x0 = 0, x1 = 1, x2=1, $x3_0$ = 0, $x3_1$ = 1, x4 = 0, x5 = 0, x6 = 1.

Since the true_value of x0 and x2 is qubit_value + 2, the final solution is –

x0 = 2, x1 = 1, x2 = 3, x3 = 2, x4 = 0, x5 = 0, x6 = 1.

As seen below, this is the valid solution to the given problem –



Grover's algorithm requires approximately $\sqrt{N}$ iterations (calls to the oracle) to mark the solution state. But since this problem has constraints, the search space is much reduced. Experiments have shown that using 4 or 5 Grover operators produces the solution state with much higher probability than the other states.

--- End of explanation ---