Kakuro game – Submission 2 (of 2)

Adam Glos, Özlem Salehi

June 5, 2022

This is the first submission out of two, which uses 86 CNOTs for an oracle.

Contents

1 The original problem

The goal of the problem was to find solution to the system of equations

$$x_0 \neq x_1 \tag{1}$$
$$x_2 + 2 \neq x_3 \tag{2}$$
$$x_3 \neq x_4 \tag{3}$$
$$x_1 \neq x_3 \tag{4}$$
$$x_3 \neq x_5 \tag{5}$$
$$x_5 \neq x_6 \tag{6}$$
$$x_0 \neq x_2 \tag{7}$$
$$x_1 \neq x_5 \tag{8}$$
$$x_4 \neq x_6 \tag{9}$$
$$x_3 = 2 \tag{10}$$
$$x_2 + x_4 + x_3 = 3 \tag{11}$$

where all $x_i \in \{0, 1\}$ except $x_3 \in \{0, 1, 2, 3\}$ using Grover algorithm.

Instead of starting in the uniform superposition of all possible values, we already encoded some of the constraints into the initial state. Thanks to this we have the following benefits:

1. because we started at the superposition of smaller number of basic states, we can use less number iterations of Grover's search, which would likely give **significant** reduction in number of gate
2. the conditions are moved from the verification in the oracle to the initial state preparation and slightly more advanced Grover mixer, which now makes an inversion around mean only around the states we start in

3. therefore, the constraints which are encoded in the initial state do not need to be verified in the oracle which greatly saves the number of CNOTs,

4. finally let us highlight that **the constraints were not ignored or classically optimized**, but simply we chose **alternative way of encoding them into Grover algorithm**. Thus we believe the rules of the competition are not violated

5. this method is extremely difficult (if not impossible to do it efficiently) for all the constraint, thus it would likely lead to exponentially-complicated quantum circuit, which is usually beyond assumptions made for Grover algorithm.

For our purposes we choose conditions $x_0 \neq x_2$, $x_1 \neq x_0$, $x_5 \neq x_1$, $x_6 \neq x_5$, $x_4 \neq x_6$. Note that if $|x_2\rangle = |0\rangle$, then it means that $|x_0\rangle = |1\rangle$, $|x_1\rangle = |0\rangle$, $|x_5\rangle = |1\rangle$, $|x_6\rangle = |0\rangle$ and $|x_4\rangle = |1\rangle$ based on the conditions. Symmetric scenario would occur if we would start with $|x_2\rangle = |1\rangle$. This means that we should start in the state

$$|x_2, x_0, x_1, x_5, x_6, x_4\rangle \otimes |x_3\rangle = \left(\frac{1}{\sqrt{2}}(|10101\rangle + |01010\rangle\right) \otimes \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) \quad (12)$$

where registers were reordered for readability. The state for the $|x_3\rangle$ is created as usualy through Hadamard gates on both qubits, while for the rest it is done as follows:

1. Hadamard gate is applied on register x_2,

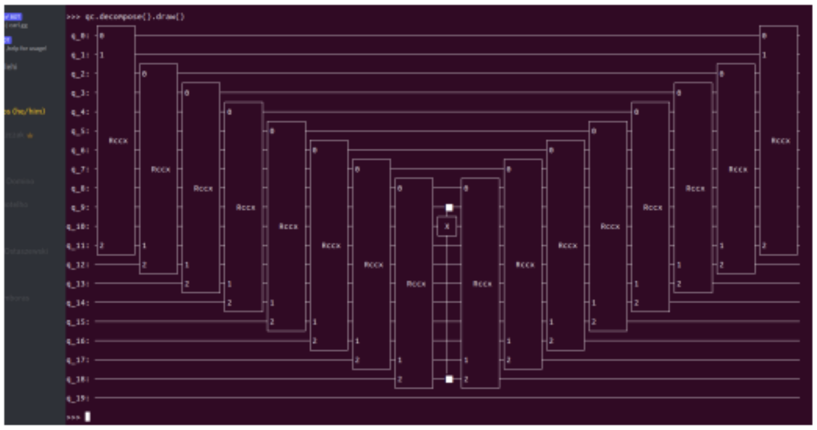
2. CNOT is applied with control on x_2 and target on x_0. Then NOT is applied on x_0. Then we obtain $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$,

3. CNOT is applied with control on x_0 and target on x_1. Then NOT is applied on x_1. Then we obtain $\frac{1}{\sqrt{2}}(|010\rangle + |101\rangle)$, etc.

The Grover mixer is then as follows: we apply inverse of state preparation unitary. Then we apply the traditional part of the Grover diffusion : NOT on all qubits corresponding to register x_i, multi-controlled Z gate on the same qubits, and again NOT on all qubits. Finally a state preparation unitary is applied.

Note that similar technique was used to create Grover Mixer for QAOA, which relies on the very similar effect `https://arxiv.org/abs/2006.00354`.

The oracle was implemented as follows. First we prepared a set of n qubits, which are all supposed to be equal to $|1\rangle$ iff all conditions are met. Different qubit(s) are corresponding to different conditions from above, thus any qubit set to $|0\rangle$ means the corresponding condition is not satisfied. Then we apply the multi-controlled Z operation with controls on first $(n-1)$ qubits and target on the last qubits (note in controlled Z there is actually no difference between control and target qubits). This multi-controlled Z corresponds to changing the $|1\dots1\rangle$ to $-|1\dots1\rangle$ and doing nothing for other states. Such multi-controlled Z operation was implemented by applying Hadamard gate on the target qubit, multi-controlled NOT gate with same controls and target, and again Hadamard gate on the target qubit. We used v-chain implementation of multi-controlled NOT given by qiskit, which consists of appropriately applied many phase-relative Toffoli gates RCCX (which greatly limits the number of CNOTs) and a single true Toffoli gate as below


```
>>> qc.decompose().draw()
```

Applying multi-controlled Z should be interpreted as marking the solution. Then, unitary preparing mentioned n qubits is uncomputed (equivalent to applying its inverse which is simply reverse order of inversion of gates).

The core part is to appropriately prepare n qubits for the remaining conditions. We will dedicate next section to how this conditions were implemented.

2 Implementation of conditions

Some constraints are implemented inplace (no additional bits are used and bits representing the variables become the control of multi controlled Z) and for some new constraint bits are used.

In all cases except the single Toffoli in the multi-controlled Z we replaced the Toffoli gate with relative phase Toffoli (RCCX). This has not effect on the solution as the incorrected phases are uncomputed. Thus whenever we will say Toffoli gate we in fact mean RCCX gate.

We would like to mention that two sequences of bits a, b equal if their XOR is all-zero. This can be simulated by CNOT-gates with controls on bits of a and target on bits of b. This way on b register we will have $|0\dots0\rangle$ iff $a = b$.

2.1 $x_2 + 2 \neq x_3$

We need additional ancilla bit for the constraint bit.

1. we XOR-ed least significant bit of x_3 with x_4, and XOR-ed most significant bit of x_3 with 1 (effectively applied NOT gate). Now x_3 is $|00\rangle$ iff $x_3 == x_2 + 2$,
2. we applied NOT on both bits of x_3. Now x_3 is $|11\rangle$ iff $x_3 == x_4$,
3. we applied Toffoli with controls on x_3 and target on the constraint bit,
4. we applied NOT on the constraint bit to change equality condition to inequality,
5. we undo the first two steps (cleaning part).

2.2 $x_3 = 2$

we store this constraint on additional ancilla qubit. What we did is,

1. we added 2 modulo 4 to the register $|x_3\rangle$ which is equivalent to applying NOT on its most significant bit,
2. we applied Toffoli with controls on x_3 and target on extra constraint bit to set it to $|1\rangle$ if the condition is met
3. we undo adding 2

2.3 $x_3 \neq x_4$, $x_3 \neq x_1$, $x_3 \neq x_5$

Let's consider first scenario, the rest is similar. For each we need single additional constraint qubit. We do the following:

1. we XOR-ed least significant bit of x_3 with x_4. Now x_3 is $|00\rangle$ iff $x_3 == x_4$,
2. we applied NOT on both bits of x_3. Now x_3 is $|11\rangle$ iff $x_3 == x_4$,
3. we applied Toffoli with controls on x_3 and target on the constraint bit,
4. we applied NOT on the constraint bit to change equality condition to inequality,
5. we undo the first two steps (cleaning part).

2.4 $x_2 + x_4 + x_3 = 3$

In this step we store the LHS sum on the register x_3 by making first inplace addition $x_3+=x_2$ and then $x_3+=x_4$. Then the condition is satisfied iff on register x_3 we have $|3\rangle = |11\rangle$, so both bits of x_3 becomes constraint bits.

Note that $0 \leq x_2 + x_4 + x_3 \leq 6$. Therefore to check the constraint it is enough to verify equality to 3 modulo 4, which justifies why we can make inplace addition on register x_3.

The addition $x_3+=x_4$ was done as follows (using so-called half-adder):

1. we applied Toffoli gate with control on x_4 and the least significant bit of x_3 and target on the most significant bit of x_3
2. we applied CNOT gate with control on x_4 and target on the least significant bit of x_3

Normally half-adder requires extra bit for carry-out, and extra for solution bit, see `https://www.qmunity.tech/tutorials/building-a-half-adder-circuit`. However here since we make an inplace addition to x_3 register, and the most significant bit can play a role of carry-out bit, *the above method is a general method of applying bit 0 to two-bit register modulo 4*. In fact the above simplification can always be made for $x+=y \mod 2^k$ on two most significant bits of x if $y \in 0, 1$ and x_3 using k bits.