



VQA FOR ORDER-FINDING

THESIS BY KAROLIS SPUKAS

HAEMANTH V
ROHAN SHARMA



THE ORDER FINDING PROBLEM

IT CAN BE FORMALLY SHOWN THAT FACTORIZATION PROBLEM GETS POLYNOMIALLY REDUCED TO THE MODULAR ORDER FINDING PROBLEM (DESCRIBED BELOW).

- FOR A GIVEN SEMIPRIME N :
- LET THE FUNCTION $f(x) = a^x \pmod{N}$:
- {WHERE a IS A COPRIME WITH N AND $x \in \mathbb{Z}_N$ }
- THEN THE ORDER r IS DEFINED AS $f(0) = f(r) = 1$
- INTUITION: THE ORDER IS SOME SORT OF A PERIOD.

TO UNDERSTAND ORDER BETTER.

- *Consider the following example:*
- $x =$ 1 2 3 4 5 6 7 8 9
- $x \pmod{3} =$ 1 2 0 1 2 0 1 2 0
- It is clear from here that there is an associated periodicity with the way numbers appear in the modular arithmetic.
- This very period is what we call order. In the above eg. $r=3$
- In SHOR's factoring algorithm, order finding is the central idea.
- The Problem: For the implementation of Shor's Algorithm, we need 1000's of fault tolerant qubits and Deep circuits, but we live in the NISQ era.

WHAT NOW?

- We are well versed with the fact that VQAs are best suited for NISQ era hardware
- This paper does exactly that.
- It discusses implementation of Variational Quantum Algorithms to tackle the Modular Order finding problem.
- We shall be discussing 2 of those that involve QAOA.

ALGORITHM 1:

- We first have to convert the order finding problem into an optimization problem for it to be solved using QAOA.
- Then introduce a basic cost function $C(x) = (a^x - 1) \bmod N$
- This will always be ≥ 0 as $\min(a^x) = 1$, so we don't have to square it.
- $\min(C(x)) = 0$ at $x \equiv 0 \bmod r$
- Thus our goal becomes to find an $x (\neq 0)$ that is as close to a multiple of r as possible.

GRAPH:

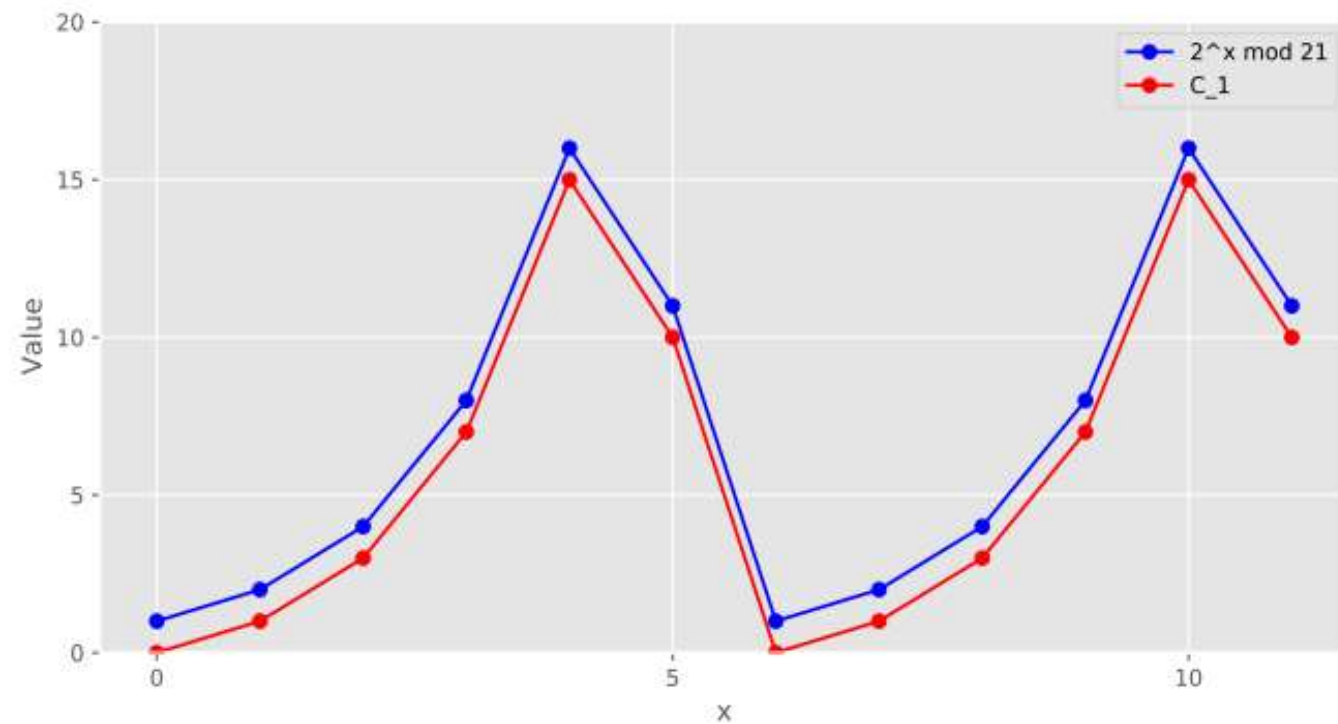


Figure 4.1: Graph for $2^x \bmod 21$ (blue) and C_1 (red). $C_1(x) = 0$ precisely when $x \equiv 0 \bmod r$

CONSTRUCTING THE HAMILTONIAN.

- Once we have the cost function, the next obvious step is to map it to a Hamiltonian.
- In order to do that, we need to rewrite it as a PBF $[\{0,1\}^k \rightarrow R]$ and convert it to a QUBO.
- Therefore there are three issues to be tackled in order to achieve H:
 1. The input $x \in Z_N$ but we want $x \in \{0,1\}^N$
 2. The exponential term a^x needs to be expressed as a polynomial.
 3. The cost function is expressed in terms of modulo N.
- To Summarize: the idea is to have a cost function which is QUBO modulo N. Such QUBO could be converted to a Hamiltonian. Modulo would be applied classically after calculating Hamiltonian expectation value and before adjusting circuit parameters.

TACKLING ISSUE 1:

- We express $x \in Z_N$ in binary, i.e $\vec{x} = (x_{\lfloor \log(N) \rfloor}, \dots, x_1, x_0)$ where $x_i \in \{0,1\}$
- Then $a^x = a^{(2^0 x_0 + 2^1 x_1 + 2^2 x_2 + \dots + 2^{\lfloor \log(N) \rfloor} x_{\lfloor \log(N) \rfloor})}$
- a^x can now be written as $a^{x_0} * a^{2x_1} * a^{4x_2} \dots a^{2^{\lfloor \log(N) \rfloor} x_{\lfloor \log(N) \rfloor}}$
- Then $a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} (a^{2^i})^{x_i}$ in compact notation.
- So now it's become a PBF as the x 's take value from $\{0,1\}$

TACKLING ISSUE 2:


- The advantage of exponentiating with binary variables can be used to convert the function to a polynomial.
- The advantage: $a^{x_i} = 1$ and a , for $x = 0$ and 1 respectively ; $[x_i \in \{0,1\}]$
- So we can write: $a^{x_i} = a * x_i + (1 - x_i)$
- So our original equation becomes: $a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} (a * x_i + (1 - x_i))^{2^i} + 1$

TACKLING ISSUE 3:

- $C(x) = (\prod_{i=0}^{\lfloor \log(N) \rfloor} (a * x_i + (1 - x_i))^{2^i} + 1) - 1 \bmod N$. It is now a PBF.
- It is a HOBO mod N and we have to convert it to a QUBO mod N using Quadrization.
- So we state $C(x) = \min_w C^*(\vec{x}, \vec{w})$ for auxiliary variables \vec{w}
- C is the HOBO version whereas C^* is the QUBO version of the cost function.



BUT.....

- While we address the issue 2, we should note the fact that on expanding the product we will end up with exponentially increasing number of terms (depicting interactions within multiple variables of \vec{x})
 - This would directly harm the complexity of our algorithm.
 - Turns out, there's no way of reducing a^x to a polynomial with polynomially many terms.
 - Taylor series ain't a good idea either. (It approximates in the neighbourhood)
 - And while addressing issue 3, we realise that the traditional quadratization methods (Freedman & Drineas and Ishikawa) do not work with the modulo operation.... So no \mathcal{C}^* .
- 

CONCLUDING LINES:

- Without first converting the PBF to its unique polynomial representation, we can't convert C to a QUBO. This introduces additional overhead.
- C^* (the QUBO version of the cost function) cannot be obtained due to lack of quadratization methods that deal with modular valued stuff.
- So the paper concludes that this algorithm based on QAOA can't be used to solve the order finding problem.

ALGORITHM 2

- Drawbacks of Algorithm 1
 - Uses modulo operation
- Modified cost function
 - $C(x, k) = (a^x - 1 - kN)^2$, $x \in Z_N$ and k is some positive integer
 - Squaring is chosen over absolute value because quadratization methods like those of Freedman & Drineas and Ishikawa do not work with absolute value operation
 - Why it works?
 - Aim to find order r such that $a^r \equiv 1 \pmod{N}$
 - Minimum value of $C(x, k)$ is 0 and is obtained only when $x = 0$ or $a^x - 1 = kN$
 - Should impose an extra constraint so that $x \neq 0$ to ensure we find a multiple of the order

CONVERTING TO PBF

- As discussed before, we convert integers x and k to their binary forms
 - $x = (x_{\lfloor \log(N) \rfloor} \dots x_1 x_0)$ and $k = (k_{\lfloor \log(d) \rfloor} \dots k_1 k_0)$ in binary for some integer d
 - $x = \sum_{i=0}^{\lfloor \log(N) \rfloor} 2^i x_i$ and $k = \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i$
 - $a^x = a^{\sum_{i=0}^{\lfloor \log(N) \rfloor} 2^i x_i} = \prod_{i=0}^{\lfloor \log(N) \rfloor} (a^{2^i})^{x_i}$
 - $a^{x_i} = x_i(a - 1) + 1$, $x_i \in \{0,1\}$ and hence $a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} (x_i(a^{2^i} - 1) + 1)$
- Cost function as PBF
 - $C(x, k) = \left(\prod_{i=0}^{\lfloor \log(N) \rfloor} (x_i(a^{2^i} - 1) + 1) - 1 - N \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i \right)^2$

ANALYZING THE COST FUNCTION

BEFORE SQUARING

- $\sum_{i=1}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$ number of positive terms \rightarrow consider all possible combinations of $x_0, x_1, x_2, \dots, x_{\lfloor \log(N) \rfloor}$ obtained from

$$a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} (x_i (a^{2^i} - 1) + 1)$$

- $\lfloor \log(d) \rfloor + 1$ number of negative terms \rightarrow obtained from

$$-Nk = -N \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i$$

ANALYZING THE COST FUNCTION

AFTER SQUARING

- $\sum_{i=1}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$ number of positive terms \rightarrow from

$$(a^x)^2 = \prod_{i=0}^{\lfloor \log(N) \rfloor} \left(x_i (a^{2^i} - 1) + 1 \right)^2$$

and $2 * (-1) * a^x$ both of which will again contain all possible combinations of $x_0, x_1, x_2, \dots, x_{\lfloor \log(N) \rfloor}$

- $\sum_{i=3}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$ terms have more than 2 variables
- The terms are considered positive though we have the negative term $-2a_x$ because $a^{2^i} - 1 < 2$ only when $a=2, i=0$ (so we ignore this only case for the sake of generality)

ANALYZING THE COST FUNCTION

AFTER SQUARING(CONTD.)

- $\lfloor \log(d) \rfloor + 1 + \binom{\lfloor \log(d) \rfloor + 1}{2}$ number of positive terms \rightarrow from

$$k^2 = \left(2^0 k_0 + 2^1 k_1 + \dots + 2^{\lfloor \log(d) \rfloor} k_{\lfloor \log(d) \rfloor} \right)^2$$

which yields all possible two variable permutations of $k_0, k_1, \dots, k_{\lfloor \log(d) \rfloor}$ and from $2 * (-1) * (-Nk)$

- $\lfloor \log(d) \rfloor + 1$ for all terms involving only one variable from $k_0, k_1, \dots, k_{\lfloor \log(d) \rfloor}$ (as $k_i \in \{0,1\}$ and hence $k_i^2 = k_i$)
- $\binom{\lfloor \log(d) \rfloor + 1}{2}$ for all possible combinations of two distinct variables from $k_0, k_1, \dots, k_{\lfloor \log(d) \rfloor}$
- All terms have atmost 2 variables(quadratized already)

ANALYZING THE COST FUNCTION

AFTER SQUARING(CONTD.)

- $(\lfloor \log(d) \rfloor + 1) \sum_{i=1}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$ number of negative terms \rightarrow from

$$2 * a^x * (-Nk) = -2N * \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i * \prod_{j=0}^{\lfloor \log(N) \rfloor} x_j (a^{2^j} - 1) + 1$$

which yields all possible combinations of $x_0, x_1, \dots, x_{\lfloor \log(N) \rfloor}$ and any 1 variable from $k_0, k_1, \dots, k_{\lfloor \log(d) \rfloor}$

- Of these terms, $(\lfloor \log(d) \rfloor + 1) \sum_{i=2}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$ number of terms have more than 2 variables

NUMBER OF AUXILIARY VARIABLES

- Using Ishikawa's method, quadratizing a positive term of degree l requires $\left\lfloor \frac{(l-1)}{2} \right\rfloor$ variables

- Total number of positive terms of degree more than 2 =

$$\sum_{i=3}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$$

- Thus number of auxiliary variables for positive terms =

$$\sum_{i=3}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i} * \left\lfloor \frac{(i-1)}{2} \right\rfloor$$

NUMBER OF AUXILIARY VARIABLES(CONTD.)

- Using Freedman & Dinneen's method, quadratizing a negative term requires 1 variables

- Total number of negative terms of degree more than 2 =

$$(\lfloor \log(d) \rfloor + 1) \sum_{i=2}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$$

- Thus number of auxiliary variables for negative terms =

$$(\lfloor \log(d) \rfloor + 1) \sum_{i=2}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$$

TOTAL NUMBER OF VARIABLES/QUBITS

- We use 1 qubit for each variable
 - For x , we use $x_0, x_1, \dots, x_{\lfloor \log(N) \rfloor} = \lfloor \log(N) \rfloor + 1$ variables
 - For k , we use $k_0, k_1, \dots, k_{\lfloor \log(d) \rfloor} = \lfloor \log(d) \rfloor + 1$ variables
- Thus total number of qubits required(including auxiliary qubits) are
$$\lfloor \log(N) \rfloor + 1 + \lfloor \log(d) \rfloor + 1 + \sum_{i=3}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i} * \left\lfloor \frac{(i-1)}{2} \right\rfloor +$$
$$(\lfloor \log(d) \rfloor + 1) \sum_{i=2}^{\lfloor \log(N) \rfloor + 1} \binom{\lfloor \log(N) \rfloor + 1}{i}$$
- Note: Adding a constraint to ensure $x \neq 0$ will not introduce new terms and will only affect the scalar coefficients of existing terms

CHALLENGES AND CONCLUSION

- Algorithm works but it introduces exponentially many variables –

$O(\lfloor \log(d) \rfloor * 2^{\lfloor \log(N) \rfloor}) = O(\lfloor \log\left(\frac{a^{N-1}-1}{N}\right) \rfloor * 2^{\lfloor \log(N) \rfloor})$ but our goal is to only find the $\lfloor \log(N) \rfloor + 1$ variables in the binary representation of x . The smallest instance considered in the chapter, uses 8 qubits of which only 2 qubits encode x .

- Hence it is not suitable for practical applications(has scaling limitations)
- Moreover, this algorithm may give us a multiple of the order and not the order itself.