Variational Quantum Factoring

Piyush Galav Milan John Sahil Prabhudesai Navjot Singh Tushar
Sakthi Jay Mahenthar C Nripendra Majumdar Jyotiprakash Parhi
Shyambabu Pandey Soumyodeep Mukhopadhyay Yeshab Yadav
Ankit Singh Thakur Santoshkumar S Veeramani TS Pujitha
Yugandhara Nangare

Advisor: Sukhsagar Dubey

July 2022

1 Introduction

Factorization has always been topic of interest because of its use in almost every type of encryption decryption models(especially RSA).Integer factorization plays a vital role in the applications of quantum computing. We can use classical computers for this but classical computational power takes exponential time with respect to bit length of number to be factorized.By Shor's algorithm we can solve this even exponentially fasten with quantum computer.

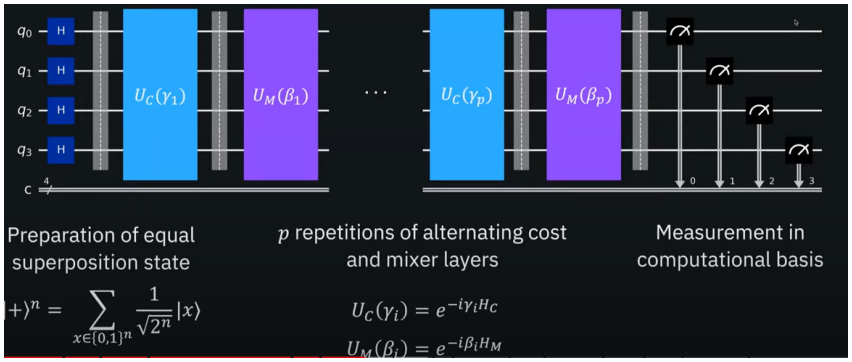
But Shor's algoritism ensures that we can reduce the complexity to Polynomial time (i.e. N^{20}) to make it efficient.Now we just need 4099 Stable qubits, because using these a quantum computer can break RSA-2048 in 10 seconds. We do not have large number of qubits which are stable (because they may be noisy). The biggest quantum computer till now have 72 qubits. we can work on the problem of factoring integers as an optimization problem. So we first convert it and then solve it using QAOA. As we don't have 4000 qubits, we take the help of VQF to solve this. Here we use factoring as inverse of multiplication.

2 QAOA

The QAOA(Quantum approximate optimization algorithm) is the direct extension of Variational Quantum Eigensolver (VQE) and has the same optimization structure as VQE(since both are optimization problems) but one difference between the VQE and QAOA algorithms is that in VQE initially the parametrized state $\psi|\vec{\theta}>$,where $\vec{\theta} = (\theta_1, \theta_2, .., \theta_i, ..)$ is the parameters, is prepared by applying arbitrary rotation gates whose angle is parametrized arbitrarily, whereas in QAOA we use p-layers of parametrized X-rotations which is obtained by exponentiation of the mixer Hamiltonian with parameters $\vec{\beta}$ and p different layers of exponential of problem Hamiltonian with parameter γ. Hence, the trial state in QAOA is $\psi|\vec{\beta}, \vec{\gamma}>$, parametrized by two parameters $(\vec{\beta}, \vec{\gamma})$, which is obtained by applying the exponentials of mixer and problem Hamiltonian to the initial state (which is generally linear superposition of all qubits obtained by applying H-gate to all qubits in problem).

Thus the basic principle of QAOA is quantum adiabatic theorem, in which we evolve the initial

state (which is the eigenstate of mixer Hamiltonian) by p blocks of problem and mixer hamiltonian in a adiabtic manner so that it is also the ground state of problem hamiltonian. but to achieve this we must have limit p tends to ∞. but practically we can apply finite number of gates so we apply only finite p-layers to prepare parameterized state $\psi|\vec{\beta},\vec{\gamma}>$ and then apply hybrid classical-quantum approach to find the optimal values of $\vec{\beta}$ and $\vec{\gamma}$ that is $\vec{\gamma_{op}}$ and $\vec{\beta_{op}}$ and measure $\psi|\vec{\gamma_{op}}\vec{\beta_{op}}>$ to obtain the bitstring that maximizes or minimizes the cost function.



3 VQF algorithm

The general idea here is construct the relevant Hamiltonian for our problem first. Let's start with the basics. So, we have the binary representations of p,q and m. From the equation m=p*q, we can perform binary multiplication and we get a set of equations. One quick mention of an assumption we make for this is that we know the sizes of p and q beforehand.

Binary multiplication is done as follows:-

p					1	p_2	p_1	1
q					1	q_2	q_1	1
					1	p_2	p_1	1
				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
carries	z_{67}	z_{56}	z_{45}	z_{34}	z_{23}	z_{12}		
	z_{57}	z_{46}	z_{35}	z_{24}				

$$
\begin{aligned}
p_1 + q_1 &= 1 + 2z_{12} \\
p_2 + p_1q_1 + q_2 + z_{12} &= 1 + 2z_{23} + 4z_{24} \\
1 + p_2q_1 + p_1q_2 + 1 + z_{23} &= 1 + 2z_{34} + 4z_{35} \\
&\vdots \\
q_2 + p_2 + z_{45} + z_{35} &= 0 + 2z_{56} + 4z_{57} \\
1 + z_{56} + z_{46} &= 0 + 2z_{67} \\
z_{67} + z_{57} &= 1.
\end{aligned}
$$

We can write these set of equations to condense all of these equations into one.

$$0 = \sum_{j=0}^{i} q_j p_{i-j} + \sum_{j=0}^{i} z_{j,i} - m_i - \sum_{j=1}^{n_c} 2^j z_{i,i+j}$$

Now we need to frame this equation as an optimization problem. We can do so by creating clauses of the following form:-

$$C_i = \sum_{j=0}^{i} q_j p_{i-j} + \sum_{j=0}^{i} z_{j,i} - m_i - \sum_{j=1}^{n_c} 2^j z_{i,i+j},$$

$$0 = \sum_{i=0}^{n_c} C_i^2.$$

Apply the following classical pre-processing rules to the Clauses-

$$\begin{aligned} xy - 1 = 0 &\implies x = y = 1, \\ x + y - 1 = 0 &\implies xy = 0, \\ a - bx = 0 &\implies x = 1, \\ \sum_i x_i = 0 &\implies x_i = 0, \\ \sum_{i=1}^{a} x_i - a = 0 &\implies x_i = 1. \end{aligned}$$

Once you get the finalized clauses, we have a very intuitive way to think of the Hamiltonian

$$E = \sum_{i=0}^{n_c} C_i'^2$$

Minimizing the Classical energy function gives us our desired constraint

$$b_k \to 1/2(1 - \sigma_{b,k}^z)$$

Apply the standard conversion to p_i, q_i and z_i in the clauses to get the corresponding Hamiltonian operator H. Set $H_c = H$ for the QAOA to work with. Mixing Hamiltonian is just the sum of all the sigma variables.

$$H_a = \sum_{i=1}^{N} \sigma_i^x$$

The Quantum Variational circuit works on the n qubits with p-layers using alternating cost layer and mixing layer parametrized by vectors beta and gamma.

$$|\beta, \gamma> = \prod_{i=1}^{p} (exp^{i\beta_i H_a} exp^{i\gamma_i H_c}) |+>^{\otimes n}$$

The classical optimizer is fed the expectation value of the Hamiltonian for that parametrized state and optimizes this objective function by finding the optimal parameters.

$$M(\beta, \gamma) = <\beta, \gamma | H_c | \beta, \gamma>$$

4 Implementation: Quantum factorization of 143 using 4 Qubits

VQF is a hybrid of both quantum and classical methods which transforms the factorization problem into an optimization problem. In the below example the integer 143 is factorized into 11 and 13 using only 4 qubits.

(To get the complete source code visit - https://github.com/cozmoguy/VQF_Project.git)

Essential imports - *qiskit, numpy, math, pandas*

4.1 Setting up the optimization problem

The initial process involves converting the integer into binary and writing the unknown digits of factors as variables p and q. The factors can be written as (1 p1 p2 1) and (1 q1 q2 1). These can be used to plot a binary multiplication table and get the following equations.

Table 1: *Multiplication table for* $11 \times 13 = 143$ *in binary.*

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
p					1	p_2	p_1	1
q					1	q_2	q_1	1
					1	p_2	p_1	1
				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
carries	z_{67}	z_{56}	z_{45}	z_{34}	z_{23}	z_{12}		
	z_{57}	z_{46}	z_{35}	z_{24}				
$p \times q = 143$	1	0	0	0	1	1	1	1

$$
\begin{aligned}
p_1 + q_1 &= 1 + 2z_{12} && (1)\\
p_2 + p_1q_1 + q_2 + z_{12} &= 1 + 2z_{23} + 4z_{24} && (2)\\
1 + p_2q_1 + p_1q_2 + 1 + z_{23} &= 1 + 2z_{34} + 4z_{35} && (3)\\
&\vdots\\
q_2 + p_2 + z_{45} + z_{35} &= 0 + 2z_{56} + 4z_{57} && (4)\\
1 + z_{56} + z_{46} &= 0 + 2z_{67} && (5)\\
z_{67} + z_{57} &= 1. && (6)
\end{aligned}
$$

The above equations can be simplified using the rules of binary addition and we get:

$$
\begin{aligned}
p_1 + q_1 - 1 &= 0 && (7)\\
p_2 + q_2 - 1 &= 0 && (8)\\
p_2q_1 + p_1q_2 - 1 &= 0. && (9)
\end{aligned}
$$

Equations 7,8 and 9 are squared and added to get a function which is to be redused to get the values p1, p2, q1 and q2.

$$H\_function = 5 - 3p_1 - p_2 - q_1 + 2p_1q_1 - 3p_2q_1 + 2p_1p_2q_1 - 3q_2 + p_1q_2 + 2p_2q_2 + 2p_2q_1q_2$$

Each variable in the above function can take the values 0 or 1. Since there are 4 variables there would be a total of 16 possible combinations. Giving those inputs to the function yields a list of values that are stored in the list *dig_H*.

Considering the list *dig_H* as the diagonal elements we can define a matrix that acts as the Hamiltonian (H) of the VQF method.

```
H  =    array([[5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
[0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]])
```

4.2 Decomposing the matrix into sum of tensor products of pauli matrices:

"*Expectation value of a Hamiltonian can be written as the linear combination of expectation value of tensor product of pauli matrices*". This principle can be employed to get an equation for the expectation value of the matrix H. For that first we have to decompose the above matrix. Since the matrix H only involve diagonal elements we have to consider only the combinations of I and Z matrix.

A 16*16 Matrix in the pauli matrix basis can be written as:

$$H = \sum_{i,j,k,l} h_{ijkl} \cdot \frac{1}{4} \sigma_i \otimes \sigma_j \otimes \sigma_k \otimes \sigma_l,$$

where σ represents pauli matices and the coefficient h_{ijkl} can be written as:

$$h_{ijkl} = \frac{1}{4} \mathrm{Tr}\big((\sigma_i \otimes \sigma_j \otimes \sigma_k \otimes \sigma_l)^\dagger \cdot H\big) = \frac{1}{4} \mathrm{Tr}\big((\sigma_i \otimes \sigma_j \otimes \sigma_k \otimes \sigma_l) \cdot H\big)$$

The Decomposed Hamiltonian H can be written as:

$H = 2.0 * iiii + 0.5 * iiiz + 0.25 * iizi + 0.25 * iizz + 0.25 * izii + 0.75 * iziz + -0.25 * izzi + -0.25 * izzz + 0.5 * ziii + 0.25 * ziiz + 0.75 * zizi + 0.25 * zzii + -0.25 * zzzi$

4.3 Variational Quantum Circuits

To find the expectation value of individual elements in the pauli matrix basis we have to use variational quantum circuits that takes in certain parameters that act as the ansatz. These ansatz manipulate the qubits which can be visualized as rotations. Variying these parameters using an optimization algorithm can be used to find a suitable expectation value.

For this particular example we have tried a set of 9 variational quantum circuits. These circuits also involve depths that repeats the circuit several times for better results.

4.4 Expectation Function

The expectation value equation can be defined as follows. The function takes in a variational quantum circuit and measures all the expectation values of pauli matrix basis elements. The output is the expectation value of the hamiltonian.

```
def expectation_function(params, num_shots = 10000):
    var_circ = var_form_ryrz_00(params, 2)
    list_bin = ['0000','0001','0010','0011','0100','0101','0110','0111','1000',
                '1001','1010','1011','1100','1101','1110','1111']
    prob_list = []
    for k_ in range(2):
        for l_ in range(2):
            for m_ in range(2):
                for n_ in range(2):
                    if (1/16)*np.trace(np.kron(np.kron(pauli[k_],pauli[l_]),np.
↪kron(pauli[m_],pauli[n_]))*H) != 0:
                        Basis_meas = var_circ.copy()
                        Basis_meas.measure_all()
                        result = execute(Basis_meas, backend = simulator, shots␣
↪= num_shots).result()
                        counts = result.get_counts(Basis_meas)
                        for jj in list_bin:
                            if jj not in counts:
                                counts[jj] = 0
                        total_counts = 0
                        for i_ in list(counts.values()):
                            total_counts += i_
                        one = np.ones(16).astype('int')
                        prob = np.inner(np.matmul(one,␣
↪basis_arr[k_][l_][m_][n_]),list(counts.values()))
                        prob = prob/total_counts
                        prob_list.append(prob)

    cost = 0
    for i_ in range(len(prob_list)):
        cost += (prob_list[i_] * coeff[i_] )
    return cost
```

4.5 Classical Optimization

Our goal is to reduce the value of expectation_function to the minimun. Since the limit of expectation value is from the smallest eigenvalue to the largest eigenvalue, by reducing the expectation_function we get the lowest eigenvalue. This can be done using a classical optimizer. In our example we are using *COBYLA* which is a gradient free optimizer.

4.5.1 Optimization dataset for various circuits and depths

For getting the best result we have tried various variational circuits with different values of depths. The dataset shown below shows the same.

	var_form_ryrz_00	var_form_ryrz_01	var_form_ryrz_02	var_form_ryrz_03	\
0	0.52855	1.57930	1.73725	0.63785	
1	1.66415	1.49250	1.57640	1.65955	
2	2.42090	1.57155	1.88750	0.77340	
3	2.62705	0.78910	1.83635	1.10540	
4	3.03715	1.01920	2.31965	1.41955	
5	0.81615	1.54355	1.48160	1.59395	
6	0.96925	1.99935	2.01455	1.04350	
7	1.97010	2.00450	1.42530	1.18135	
8	1.95505	1.13185	2.18690	0.76060	

	var_form_ryrz_04	var_form_ryrz_05	var_form_ryrz_06	var_form_ryrz_07	\
0	1.54950	1.04645	2.35155	1.01495	
1	1.29845	0.88940	1.65330	1.29020	
2	1.61670	1.99155	2.18220	1.16705	
3	1.60460	1.11085	2.07910	1.35180	
4	1.55775	1.01505	1.72900	1.68780	
5	1.46715	1.71635	1.45160	2.40590	

6	1.85710	1.37120	3.53355	0.49250
7	1.26930	1.47220	1.94115	0.95800
8	1.50955	1.28330	1.77995	2.22960

	var_form_ryrz_08
0	0.70215
1	1.47625
2	1.66750
3	1.59110
4	1.69135
5	1.48820
6	1.39815
7	1.65920
8	1.20890

We get the lowest value as 0.49 the eigenvalue lower than that is 0, hence it is the lowest eigenvalue

4.6 Determining the binary value corresponding to the lowest eigenvalue

After getting the lowest eigenvalue we can get the binary corresponding to the position of lowest eigenvalue

$$0\,1\,1\,0$$
$$1\,0\,0\,1$$

We got two positions in which the lowest eigenvalues are present. We can treat them as seperate cases.

Case 1	Case 2
p1 = 0, p2 = 1, q1 = 1, q2 = 0	p1 = 1, p2 = 0, q1 = 0, q2 = 1

Both the cases point out that the factors of 143 in binary can be written in binary as 1011 and 1101, which are 11 and 13 respectively.

4.7 Conclusion

We Successfully factorized the number 143 into 11 and 13 using only 4 qubits. This was only possible because the factors differ by only 2 digits in binary form. Similarly any integer whose factors differ by only 2 digits can be factorized using 4 qubits. Examples of such integer are 3599, 11663, and 56153. For the integers with factors differing more than 2 digits we have to use more qubits.

References

[1] Anschuetz, Eric & Olson, Jonathan & Aspuru-Guzik, Alán & Cao, Yudong. (2018). Variational Quantum Factoring.

[2] Nikesh S. Dattani, Nathaniel Bryans. Quantum factorization of 56153 with only 4 qubits

[3] Andreas Baumhof. (2019). Breaking RSA Encryption – an Update on the State-of-the-Art.

[4] Qiskit Global Summer-Introduction to Quantum Approximation Optimization Algorithm and Applications (2021)

[5] Thomas Wong. (2022) Introduction to Classical and Quantum Computing

[6] Zaiku Group. Variational Quantum Factoring Open Source Research