Practical Quantum Algorithms for Order-Finding Problem

Karolis Spukas

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

Shor's algorithm is almost exponentially faster than the best classical algorithm for integer factorization. However, there currently does not exist a quantum computer that could run Shor's algorithm for relevant instances due to its requirements for thousands of perfect qubits and deep quantum circuits. Recent advances in Noisy Intermediate-Scale Quantum (NISQ) devices bring hope that a new class of practical quantum algorithms could be used to solve tasks such as factorization earlier than otherwise possible. Previous works on practical quantum algorithms for integer factorization mostly focused on mapping this problem to an optimization problem as first proposed by Burges. Their results, however, had scaling limitations and extensive need for classical preprocessing.

In this work, we investigate practical quantum algorithms for solving modular order-finding problem, which is related to factoring. To the best of our knowledge, this is the first ever attempt to use variational quantum algorithms for such task. We present three novel algorithms for order-finding problem based on QAOA and VQE. We theoretically analyse proposed algorithms and present experimental simulation results where possible. We find that it is not trivial to find a good cost function for order-finding problem, and thus QAOA-based algorithms may not be useful. We also find that our VQE-based algorithm can solve small problem instances, yet gets stuck in local minima for larger instances. Further work is needed to determine the feasibility of this algorithm. Lastly, we present an example application of our VQE-based algorithm for factoring semiprimes 21 and 437 in simulation using 3 and 9 qubits respectively.

Acknowledgements

I would like to thank Dr. Petros Wallden, my project supervisor, for all the help, encouragement and patience. Without my supervisor's help, I would not have been able to explore the field of quantum computing as in-depth as I could. This project was my first research experience and I am grateful for the lessons learned.

I would also like to thank my family and friends for the support and interest in my work.

Table of Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contributions . . . 2
1.3 Project outline . . . 3

2 Background **4**
2.1 Order-finding problem . . . 4
2.2 Variational Quantum Algorithms . . . 4
2.2.1 Hamiltonians . . . 5
2.2.2 Mapping optimization problems to Hamiltonians . . . 7
2.2.3 Variational Quantum Eignesolver . . . 8
2.2.4 Quantum Approximate Optimization Algorithm . . . 9

3 Review of factoring using Practical Quantum Algorithms **12**
3.1 Factoring problem as optimization problem . . . 12
3.2 Classical factoring algorithm with quantum subroutine . . . 14
3.3 Algorithms for order finding . . . 14

4 Algorithm 1 based on QAOA **16**
4.1 Theory . . . 16
4.1.1 Addressing problem 1 . . . 17
4.1.2 Addressing problem 2 . . . 17
4.1.3 Addressing problem 3 . . . 18
4.2 Discussion . . . 19

5 Algorithm 2 based on QAOA **20**
5.1 Theory . . . 20
5.1.1 Theoretical analysis . . . 21
5.2 Experiments . . . 22
5.2.1 Experiment setup . . . 22
5.2.2 Simulation results . . . 23
5.3 Discussion . . . 23

6 Algorithm 3 based on VQE **24**
6.1 Theory . . . 24
6.1.1 Theoretical analysis . . . 25

6.1.2 Choice of cost function 26
6.1.3 Choice of classical optimization 27
6.1.4 Choice of ansatz 27
6.1.5 Choice of initial state 27
6.1.6 Choice of encoding: binary vs. Gray 27
6.2 Experiments 28
6.2.1 Performance measure 28
6.2.2 Experiment setup 28
6.2.3 Simulation results: scaling of K, T 29
6.2.4 Simulation results: finding optimal hyperparameters 32
6.2.5 Quantum hardware results 33
6.3 Discussion 34

7 Application example: factoring semiprime 437 using 9 qubits 36

8 Conclusions and Future Work 38
8.1 Future work 38

Bibliography 40

A Reduction from integer factorization to order-finding 44

B Circuit for factoring semiprime 437 45

Chapter 1

Introduction

The current decade is going to be one of the most exciting times in the history of Computer Science. With recent advances made in Quantum Computing and the rise of Noisy Intermediate-Scale Quantum (NISQ) devices [35], the field is quickly approaching a breaking point where the best classical computers are no longer able to compete with quantum devices when solving some specific tasks. Over the past few years, there have been several claims of demonstrating such quantum supremacy [2, 49]. Although the number of imperfect qubits in NISQ devices ranges from dozens to hundreds, this is already pushing the limit of what can be simulated classically.

1.1 Motivation

Some of the widely used security protocols, such as RSA [36], are built on the underlying hardness assumption of factoring large semiprimes (products of two prime numbers). Factoring a semiprime N is closely related to the modular order-finding problem, which deals with finding an order of some element belonging to a multiplicative group modulo N. It can be formally shown that factoring problem polynomially reduces to order-finding problem [47] and efficiently solving the latter would also efficiently solve the former. The implications of such efficient algorithm would be groundbreaking and would constitute a major breakthrough in cryptography and mathematics. Many crucial security systems around the world, including banking and secure communication, could be broken overnight.

Currently, there are no known classical algorithms that could tackle order-finding or factoring problem in polynomial time. The best known factoring algorithm, the general number field sieve (GNFS) [33], runs in sub-exponential time. However, it is not the case in quantum computing world. In 1994, Peter W. Shor presented the famous Shor's algorithm [38] that was shown to factor numbers in polynomial time by solving the order-finding problem. This marked a new milestone in quantum computing as it was one of the first quantum algorithms demonstrated to have a theoretical quantum supremacy. However, Shor's algorithm has a major obstacle to overcome. It is not practically useful given the current state of Quantum Computing and is not going to be practical for quite some time due to its requirements for thousands of fault-tolerant

qubits and deep quantum circuits [35]. These requirements are simply impossible to achieve in the current era of NISQ devices that contain at most hundreds of imperfect qubits.

With the current rise of popularity in NISQ devices and practical quantum algorithms, we ask a question *whether practical quantum algorithms can be used to solve order-finding and factoring problems*. Such algorithm could give a new way to factor integers on quantum devices possibly decades earlier than when using Shor's algorithm. Even if the practical algorithm does not achieve the same asymptotic speed-up as Shor's algorithm, it could potentially still compete with the best classical factoring algorithms. Once fault-tolerance and large quantum devices are achieved, such practical algorithms would be replaced by Shor's algorithm, however there is a very long way to go until this is the case. Moreover, this problem is interesting to tackle from purely theoretical perspective, as practical quantum algorithms is a new and exciting subfield of Quantum Computing. While factoring using practical quantum algorithms has been addressed in the literature [37, 48, 8, 1], to the best of our knowledge no one has ever used them to tackle order-finding problem. Thus, our work presents a completely novel perspective into a well-known mathematical problem.

In this work, the practical quantum algorithms we consider are the Variational Quantum Algorithms (VQAs). VQAs consist of parametrized quantum circuits optimized by classical optimization methods. The main goal of our work is *to use variational quantum algorithms to solve the order-finding problem*. It should be noted that we do **not** expect to observe any speed-up over Shor's algorithm. A side goal of our work is *to review the current advances and state-of-the-art methods for factoring using practical quantum algorithms*.

1.2 Contributions

Our main contributions in this work can be summarized as follows:

- We presented the first ever attempt to address the order-finding problem using Variational Quantum Algorithms;
- We introduced three novel algorithms for this problem based on VQE and QAOA;
- We evaluated our proposed algorithms theoretically and experimentally where possible. We found that while algorithms based on QAOA may not be used, the algorithm based on VQE may show hope. We showed an example of factoring semiprimes 21 and 437 with 3 and 9 qubits respectively using the algorithm based on VQE in simulation;
- We critically reviewed and compared existing practical quantum algorithms for factoring, majority of which are based on Burges [5] method to map factorization problem into optimization problem.
- As a side contribution, we explored some properties of pseudo-Boolean cost functions involving modulo operation and absolute value in the context of QUBO (Quadratic Unconstrained Binary Optimization) problems. We found that the

traditional methods (Freedman & Drineas [13], Ishikawa [16]) of introducing auxilliary variables do not work for such functions, while conversion to unique polynomial representation introduces exponentially many new terms.

1.3 Project outline

In chapter 2, we introduce relevant background material for understanding our work by presenting order finding problem and variational quantum algorithms. In chapter 3, we critically review the current state of factoring using different practical quantum algorithms. In chapters 4, 5 and 6 we propose the novel algorithms for order-finding problem and their theoretical and experimental analysis where possible. In chapter 7 we practically demonstrate the ability of one of the algorithms to factor some example semiprimes in simulation. Lastly, in chapter 8 we conclude our work and give an overview of possible future research directions.

Chapter 2

Background

In this chapter, we present the background material required for this work. In section 2.1, we formally define the order-finding problem and its relation to factoring. In section 2.2, we present Variational Quantum Algorithms.

2.1 Order-finding problem

Modular order-finding problem can be stated as follows: given a multiplicative group mod N, find the order r of some element a. Then factorization of $N = pq$ into prime factors p, q can be polynomially reduced to finding such r as shown by Woll [47]. For those interested in an example reduction algorithm, it can be found in the appendix A as presented in the original Shor's paper [38]. It should be noted that this *modular* order-finding problem is different from the general order-finding problem, which uses an oracle and is provably exponential as shown by Cleve [7]. In this work, we use the term order-finding to mean the modular order-finding problem.

For alternative statement of the modular order-finding problem, let N be the semiprime to factor, $a \in \mathbb{Z}_N$ random integer such that $gcd(N, a) = 1$, $x \in \mathbb{Z}_N$. Consider the following periodic function:

$$f(x) = a^x \bmod N \tag{2.1}$$

There exists smallest positive integer r called the period of f s.t. $f(0) = f(r) = 1$. Further, assume f is strongly periodic, that is $f(x_1) = f(x_2)$ iff x_1 and x_2 differ by some multiple of the period. Then r is precisely the order of element a in the multiplicative group mod N. The formulation of finding period of f is the formulation used throughout our work.

2.2 Variational Quantum Algorithms

Variational Quantum Algorithms (VQA) is a class of hybrid quantum/classical algorithms that can be run on existing NISQ devices without the need for error correction.

The goal of such algorithms is to classically optimize parametrized quantum circuits such that, once measured, the quantum circuit gives the desired solution. If a Hermitian operator called Hamiltonian is used to encode the problem, then such algorithms can determine good approximations to its eigenvectors and eigenvalues. In this case, VQA consists of a quantum circuit $U(\vec{\theta})$ controlled by some parameters $\vec{\theta}$, which are used to optimize the circuit by classical optimization algorithms. That is, given a state $|\Psi(\vec{\theta})\rangle = U(\vec{\theta})\,|0\rangle$, the classical algorithm aims to find parameters $\vec{\theta}$ such that the expectation value of system's Hamiltonian $\langle H\rangle$ for this state is minimised:

$$\min_{\vec{\theta}} \langle H_{\Psi(\vec{\theta})}\rangle = \min_{\vec{\theta}} \langle \Psi(\vec{\theta})|H|\Psi(\vec{\theta})\rangle \tag{2.2}$$

Variational method of quantum mechanics states that this minimised expected value is the upper bound of the ground state energy of the system e_0 [46]. In formula:

$$\min_{\vec{\theta}} \langle H_{\Psi(\vec{\theta})}\rangle \geq e_0 \tag{2.3}$$

Therefore, given good enough conditions, we can approximate as close to the ground state energy of any Hamiltonian as we like.

The general procedure to employ VQA is as follows:

1. Express some classically hard problem as an optimization problem with associated cost function J that needs to be minimised/maximised
2. Map J to some Hamiltonian H whose ground state energy E_0 encodes the solution to the problem
3. Use one of the VQAs parametrized by $\vec{\theta}$ to approximate the ground state energy of H and obtain the solution to the problem by finding optimal $\vec{\theta}$

We start by introducing Hamiltonians in subsection 2.2.1 and discussing the process of mapping an optimization problem to some Hamiltonian in subsection 2.2.2. We then look at two variational quantum algorithms that are of particular interest to us: the Variational Quantum Eigensolver (VQE) in subsection 2.2.3 and the Quantum Approximate Optimization algorithm (QAOA) in subsection 2.2.4.

2.2.1 Hamiltonians

Hamiltonian H is "a [Hermitian] operator [on Hilbert space] corresponding to the total energy of that system, including both kinetic energy and potential energy." [43]. For example, the Hamiltonian of a hydrogen molecule encodes information about its energy levels. Hamiltonian also gives the time evolution of a state as described by Schrodinger equation. That is, given a state $|\psi(t_1)\rangle$ at time t_1, the state $|\psi(t_2)\rangle$ at time t_2 can be written as follows [30]:

$$|\psi(t_2)\rangle = e^{-iH(t_2-t_1)/h}\,|\psi(t_1)\rangle \tag{2.4}$$

where h is the Planck's constant and the exponential term corresponds to some unitary operator. As is the case for every Hermitian operator, a $N \times N$ Hamiltonian has N linearly independent eigenvectors that form an orthonormal basis (ONB), and each is associated with a real eigenvalue. Thus, by spectral theorem H can be decomposed as follows:

$$H = \sum_{i}^{N} e_i \left|E_i\right\rangle \left\langle E_i\right| \tag{2.5}$$

where e_i is the i-th eigenvalue and E_i is the i-th eigenvector. These eigenvectors are also known as energy states (stationary states), whereas the corresponding eigenvalues are known as the energies of such states [30]. The eigenvector associated with the lowest eigenvalue (ground state energy) is also called the ground state of H. These lowest energy states will be the primary focus of VQAs.

Hamiltonians can have various eigenvectors. For example, we know that any n-qubit Hermitian operator can be written as a sum of weighted tensor products of some Pauli operators, since they form an ONB. In formula [25]:

$$H = \sum_{\alpha} h_\alpha (\sigma^1_{\alpha_1} \otimes ... \otimes \sigma^n_{\alpha_n}) = \sum_{\alpha} h_\alpha P_\alpha \tag{2.6}$$

where α runs over terms, h_α is a constant coefficient, $\sigma^i_{\alpha_i} \in \{I, \sigma^i_x, \sigma^i_y, \sigma^i_z\}$ is the α_i Pauli operator acting on i-th qubit and $P_\alpha = \sigma^1_{\alpha_1} \otimes ... \otimes \sigma^n_{\alpha_n}$. Sometimes this may be reduced further to only $\sigma^i_{\alpha_i} \in \{I, \sigma^i_z\}$, making H diagonal in computational basis. Its eigenvectors are thus the computational basis states $|i\rangle$, and the Hamiltonian can be written as:

$$H = \sum_{i} c_i \left|i\right\rangle \left\langle i\right| \tag{2.7}$$

where i is an n-bit string. One such type of Hamiltonians diagonal in computational basis are Ising Hamiltonians. Ising model is a mathematical model of ferromagnetism consisting of n spins arranged in a graph (lattice), where each spin can be in one of two states $\{-1, +1\}$ [44]. Any two neighbouring spins i and j are coupled by an interaction J_{ij} - one value of J_{ij} is assigned if the spins are in different states, and another value if the spins are in the same state. Another term h_i manipulates each spin i individually. Ising model is described by some spin configuration $\vec{s}$, which assigns each spin a value $s_k \in \{-1, +1\}$. Classically, the energy of such system (Ising Hamiltonian) can be written as follows [21]:

$$H(\vec{s}) = -\sum_{i,j} J_{ij} s_i s_j - \sum_{i} h_i s_i \tag{2.8}$$

where i and j in the first term are adjacent spins. Replace each spin $s_i \in \{-1, +1\}$ with a Pauli Z operator σ^i_z acting on the i-th qubit, and we get the quantum version of the Ising Hamiltonian that describes the new n-qubit system:

$$H(\vec{\sigma}) = -\sum_{i,j} J_{ij}\sigma_z^i\sigma_z^j - \sum_i h_i\sigma_z^i \tag{2.9}$$

The state $|0\rangle$ now corresponds to positive spin and $|1\rangle$ corresponds to the negative spin, since $\langle 0|\sigma_z^i|0\rangle = 1$ and $\langle 1|\sigma_z^i|1\rangle = -1$. Notice that Ising Hamiltonians are quadratic in their terms, thus allowing only 2-body interactions. This is in contrast to equations (2.6) and (2.7) which described general n-body interactions.

2.2.2 Mapping optimization problems to Hamiltonians

Any classical cost function $C(\vec{x})$, which is a polynomial of degree k in binary variables $\{x_1, ..., x_n\}$, can be mapped to some Hamiltonian whose ground state energy gives the optimal cost [41]. $C(\vec{x})$ has a general form:

$$C(\vec{x}) = c + \alpha_1 x_1 + ... + \alpha_{12} x_1 x_2 + ... + \alpha_{1..n} x_1 ... x_n \tag{2.10}$$

where the term of degree k corresponds to the k-body interaction. In other words, C is a pseudo-Boolean function (PBF) of the form $C : B^n \to \mathbb{R}$ where $B = \{0, 1\}$. It is known that any PBF can be uniquely expressed as a multilinear polynomial [32]. Notice that since the inputs to this cost function are n bit strings, i.e. 010...01, this serves as a hint that the Hamiltonian should be diagonal in computational basis states (expressed as Pauli Z matrices), as the energy states are going to be exactly such n bit strings.

To achieve the above, we first express the cost function in terms of "spins" $z_i \in \{-1, 1\}$ by substituting $x_i = (1 - z_i)/2$ such that $x_i = 1 \implies z_i = -1$ and $x_i = 0 \implies z_i = 1$. For the quantum version, we replace each spin z_i by a Pauli Z operator σ_z^i, just like we did for Ising Hamiltonians. Thus, we substitute $x_i = (I - \sigma_z^i)/2$ to get a general Hamiltonian of form (2.7). The state $|0\rangle$ corresponds to bit 0 and $|1\rangle$ to bit 1, since $\langle 0|(I - \sigma_z^i/2)|0\rangle = 0$ and $\langle 1|(I - \sigma_z^i/2)|1\rangle = 1$.

One interesting case of such mapping is the mapping between Quadratic Unconstrained Binary Optimization (QUBO) problems and Ising Hamiltonians. QUBO is a problem that deals with optimising a quadratic polynomial over binary variables and has a general form [20]:

$$C(\vec{x}) = \sum_i c_{ii} x_i + \sum_{i,j} c_{ij} x_i x_j \tag{2.11}$$

where c_{ii} and c_{ij} are real coefficients. Clearly, this is a special case of equation (2.10) with $k = 2$, and the Hamiltonian it maps to is the Ising Hamiltonian. It has been shown by Lucas [21] that various mathematical problems (including Karp's 21 NP-complete problems) can be formulated as Ising Hamiltonians through this mapping.

It is well known that any PBF expressed as its unique polynomial representation can be quadratized to a QUBO by using auxiliary variables to remove terms with degrees higher than 2. If some higher degree term $xyz...q$ is negative ($a < 0$), Freedman & Drineas [13] showed that the reduction takes only one auxiliary variable w:

$$axyz...q = \min_{w} aw(x+y+z+...+q-k+1) \tag{2.12}$$

If, however, the higher degree term is positive ($a > 0$), Ishikawa [16] showed that it takes $n_d = \lfloor \frac{d-1}{2} \rfloor$ auxiliary variables w_i:

$$ax_1...x_d = a\sum_{i=1}^{d-1}\sum_{j=i+1}^{d} x_i x_j + a \min_{w_1,...w_{n_d}} \sum_{i=1}^{n_d} w_i(c_{i,d}(2i - \sum_{i=1}^{d} x_i) - 1) \tag{2.13}$$

where the coefficient $c_{i,d} = 1$ if d is odd and $i = n_d$, $c_{i,d} = 2$ otherwise. Both methods are currently the best known methods for quadratization. Recently, there have been attempts to achieve logarithmic bounds for positive terms (Boros et al. [4]), however that was shown to work only for several classes of specially structured functions and is not used in our work.

As a result of the aforementioned properties of Hamiltonians, we are interested in approximating the ground state energy of such operators. This Hamiltonian may describe some physical system such as a Hydrogen atom, or encode some cost function of an optimization problem. In both cases, it is possible to use VQA to find the desired solution.

2.2.3 Variational Quantum Eignesolver

VQE was first introduced in 2013 by by Peruzzo et al. in a paper titled "A variational eigenvalue solver on a quantum processor" [31]. The paper demonstrated an efficient way to approximate the ground state molecular energy of a $He - H^+$ molecule. Since then, VQE has also been successfully applied to problems outside of quantum chemistry.

Suppose we have Hamiltonian H of some system of interest, whose ground state is the solution we are looking for. The steps of the algorithm are as follows [31, 24]:

Algorithm VQE

1: Choose a family of efficiently preparable parametrized quantum states $|\Psi(\vec{\theta})\rangle$ (called ansatz states)
2: Choose initial parameters $\vec{\theta}_0$ and prepare initial state $|\Psi(\vec{\theta}_0)\rangle = U(\vec{\theta}_0)|0\rangle$
3: **for** $t \in [0, T]$ **do**
4: Measure $\langle H_{\Psi(\vec{\theta}_t)}\rangle$
5: Use a classical optimisation algorithm to update the parameters from $\vec{\theta}_t$ to $\vec{\theta}_{t+1}$ that decrease $\langle H_{\Psi(\vec{\theta}_t)}\rangle$
6: **if** Algorithm converged **then return** Final set of parameters $\vec{\theta}_t$
7: **end for**
8: **return** Final set of parameters $\vec{\theta}_T$

In line 1, there is a variety of choices to make for choosing the family of states. To have an advantage over classical computers, $|\Psi(\vec{\theta})\rangle$ should be good at describing the solu-

tion (close to the ground state), as well as difficult to prepare/sample from classically [24]. The original paper [31] gave the unitary coupled cluster ansatz as one example that is important in quantum chemistry; [24] also mentioned adiabatic ansatz; A widely used and well-known is Hardware-Efficient Ansatz [17], which is made up of d layers, each of which consists of rotation gates (R_X, R_Y, R_Z) followed by entanglement gates $(CNOT, CZ)$. Ansatz family for our algorithm is chosen suitably later in this work.

For line 4, the Hamiltonian should be expressed as a sum of weighted tensor products of some Pauli operators like in equaton (2.6), and the number of terms of H must be polynomial in the size of the system in order to not incur exponential cost. Computing the expectation value of H directly is difficult. Instead, by linearity we can reduce the expectation value measurement of H to independent expectation value measurements of tensor products of Pauli operators:

$$\begin{aligned}
\langle H_{\Psi(\vec{\theta})} \rangle &= \langle \Psi(\vec{\theta}) | H | \Psi(\vec{\theta}) \rangle \\
&= \langle \Psi(\vec{\theta}) | \sum_{\alpha} h_{\alpha} (\sigma^1_{\alpha_1} \otimes \ldots \otimes \sigma^n_{\alpha_n}) | \Psi(\vec{\theta}) \rangle \\
&= \sum_{\alpha} h_{\alpha} \langle \Psi(\vec{\theta}) | \sigma^1_{\alpha_1} \otimes \ldots \otimes \sigma^n_{\alpha_n} | \Psi(\vec{\theta}) \rangle \\
&= \sum_{\alpha} h_{\alpha} \langle \sigma^1_{\alpha_1} \otimes \ldots \otimes \sigma^n_{\alpha_n} \rangle = \sum_{\alpha} h_{\alpha} \langle P_{\alpha} \rangle
\end{aligned} \tag{2.14}$$

For Hamiltonians diagonal in Z-basis (such as Ising Hamiltonians), all terms in the decomposition can be evaluated by measuring each qubit in the computational basis m times to get the expectation and then the outcome is plugged back into the expression. For eq. 2.14, we can estimate the expectation values of each term P_{α} of H up to some precision p by repeatedly taking m measurements of each term for the state $\Psi(\vec{\theta})$ and calculating the mean of outcomes. If the k-th measurement outcome of term P_{α} is x_k, then:

$$\langle P_{\alpha} \rangle = \frac{\sum_k^m x_k}{m} \tag{2.15}$$

Line 5 is then used to adjust the parameters until convergence. One such classical optimisation algorithm for finding the optimal circuit parameters is simply the gradient descent. However, as claimed by [31], it may not be robust enough, and other optimizers are suggested as alternatives.

The converged expectation value is thus the ground state energy of the Hamiltonian, and the optimal parameters $\vec{\theta}_t$ can be used to efficiently prepare the ground state as $U(\vec{\theta}_t)\,|0\rangle$. This is the desired solution.

2.2.4 Quantum Approximate Optimization Algorithm

QAOA was introduced in 2014 by Farhi and Goldstone in [10] and was used to approximately solve hard classical optimization problems. It is possible to view QAOA

as a type of VQE with a specific choice of ground state preparation [3]. Also, QAOA deals only with classical cost functions over binary variables as defined in equations (2.10) and (2.11).

The ground state preparation technique of QAOA is inspired by Quantum Adiabatic Algorithms (QAA) [10, 12]. Both QAA and QAOA evolve the system state from the ground state of a reference Hamiltonian H_r to the ground state of a target Hamiltonian H_p. H_p encodes the cost function and H_r helps guide the evolution towards the ground state of H_p [25]. QAA performs this evolution over some period of time T which, according to the adiabatic theorem, if long enough is guaranteed to produce the needed optimal state of H_p [42]. That is, for a time dependent Hamiltonian: [10]:

$$H(t) = (1 - t/T)H_r + (t/T)H_p \tag{2.16}$$

where at time 0 we have $H(0) = H_r$, and at some time T we have $H(T) = H_p$. QAOA, on the other hand, approximates this evolution by alternately applying problem Hamiltonian $U(H_p, \gamma_i)$ and reference Hamiltonuan $U(H_r, \beta_i)$ operators for $p \geq 1$ times. Thus, QAOA has shorter circuit-depth requirements than QAA. The classical part of QAOA is responsible for finding the set of $2p$ optimal parameters $\{\gamma_i, \beta_i\}$ that approximates the optimal solution best. As the depth of the circuit goes to ∞ ($p \to \infty$), with the right parameters the optimal solution converges to that of the QAA [10]. This, of course, comes at a cost of increasing circuit depth, which can quickly reach the limits of quantum hardware.

QAOA requires the target Hamiltonian H_p to be expressed as a sum of tensor products of Pauli Z operators like in equation (2.7). As discussed before, computational basis states are the ground states of such Hamiltonian and encode the n-bit string inputs to the optimization cost function. The reference Hamiltonian H_r is usually chosen to be the sum of all single bit σ_x^i operators [10]:

$$H_r = \sum_{i=1}^{n} \sigma_x^i \tag{2.17}$$

where σ_x^i acts on the i-th qubit. Its ground state is the superposition of all computational basis states [11]:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle \tag{2.18}$$

Define the operators for evolving the initial state $|s\rangle$ and the evolution itself for some p [10]:

$$\begin{aligned} U(H_p, \gamma_i) &= \exp(-i\gamma_i H_p) \\ U(H_r, \beta_i) &= \exp(-i\beta_i H_r) \end{aligned} \tag{2.19}$$

$$|\gamma,\beta\rangle = U(H_r,\beta_p)U(H_p,\gamma_p)...U(H_r,\beta_1)U(H_p,\gamma_1)\,|s\rangle \tag{2.20}$$

where the angles γ_i and β_i determine how long each operator is applied for in iteration i [41] and $|\gamma,\beta\rangle$ is the final state that should, at the end of the algorithm, be the ground state of H_p.

The steps of the algorithm are as follows [10, 11]:

Algorithm QAOA

1: Choose $2p$ initial parameters $\{\gamma_{0,i},\beta_{0,i}\}$
2: Prepare the initial state $|s\rangle$
3: **for** $t \in [0,T]$ **do**
4: Evolve the initial state to $|\gamma,\beta\rangle_t$ by alternating $U(H_r,\beta_{t,i})$ and $U(H_p,\gamma_{t,i})$ operators
5: Measure $\langle H_{p|\gamma,\beta\rangle_t}\rangle$
6: Use a classical optimisation algorithm to update the set of parameters from $\{\gamma_{t,i},\beta_{t,i}\}$ to $\{\gamma_{t+1,i},\beta_{t+1,i}\}$ that decrease $\langle H_{p|\gamma,\beta\rangle_t}\rangle$
7: **if** Algorithm convergences **then return** Final set of parameters $\{\gamma_{t,i},\beta_{t,i}\}$
8: **end for**
9: **return** Final set of parameters $\{\gamma_{T,i},\beta_{T,i}\}$

Lines 1, 2 and 4 have already been described above. Line 5, 6 and 7 are identical to VQE. Since H is diagonal in computational basis, we can simply measure the state $|\gamma,\beta\rangle$ in computational basis for m times, each of which will give an n-bit string. The expected value is thus the sample mean of these m measurements. Once optimal set of parameters $\{\gamma_i,\beta_i\}$ is obtained, measurement of the final state $|\gamma,\beta\rangle$ directly gives the n-bit string optimal solution and the cost of the optimization problem can be directly estimated by plugging it to the cost function.

Chapter 3

Review of factoring using Practical Quantum Algorithms

In this chapter, we present a thorough review of the current state of factoring using practical quantum algorithms. In section 3.1, we critically evaluate the "traditional" approach of "practical quantum factoring", which expresses factoring as an optimization problem with the help of long multiplication. In section 3.2, we present a critique of the "traditional" approach, which confirms our evaluation, and a proposed alternative by Mosca et al.. In section 3.3, we briefly cover the order-finding problem in literature and present relevant benchmark algorithms.

3.1 Factoring problem as optimization problem

The majority of works tackling factoring using practical quantum algorithms are inspired by "Factoring as Optimization" paper released in 2002 by Burges [5]. Factoring of semi-primes can be mapped to an optimization problem with a pseudo-boolean cost function whose minimum is given by the prime factors. The method could be summarized as follows:

1. Choose a semi-prime $N = pq$ and express N, p, q in binary: $N = \left[n_1, ..., n_d\right]$, $p = \left[p_1, ..., p_{d_1}\right]$ and $q = \left[q_1, ..., q_{d_2}\right]$.
2. Use long binary multiplication and write down the multiplication equations in terms of binary variables (including carry bits)
3. Pre-process the equations by setting redundant bit values using classical methods
4. Each equation (squared) becomes a term in the pseudo-Boolean cost function F.
5. Reducing F to QUBO and replacing each binary variable with Pauli Z operators yields the Ising Hamiltonian H diagonal in computational basis
6. Apply a practical quantum algorithm to find the ground state of H, which is the desired solution

For example, it was demonstrated by [48] that it is possible to come up with the following pre-processed equations to factor $N = 143 = 11 \times 13 = pq$:

$$\begin{aligned} p_1 + q_1 - 1 &= 0 \\ p_2 + q_2 - 1 &= 0 \\ p_2 q_1 + p_1 q_2 - 1 &= 0 \end{aligned} \tag{3.1}$$

Replacing p_i, q_i with $\hat{p}_i = (1 - \sigma_z^{p,i})/2$ and $\hat{q}_i = (1 - \sigma_z^{q,i})/2$, the Hamiltonian is written as:

$$\begin{aligned} H = 5 &- 3\hat{p}_1 - \hat{p}_2 - \hat{q}_1 + 2\hat{p}_1\hat{q}_1 - 3\hat{p}_2\hat{q}_1 \\ &+ 2\hat{p}_1\hat{p}_2\hat{q}_1 - 3\hat{q}_2 + \hat{p}_1\hat{q}_2 + 2\hat{p}_2\hat{p}_2 + 2\hat{p}_2\hat{q}_1\hat{q}_2 \end{aligned} \tag{3.2}$$

The first works [37], [48], [8] that mapped factoring problem to an optimization problem in such way used the quantum adiabatic algorithm to find the solution. They were able to factor semiprimes 21 using 3 qubits, 143 using 4 qubits and even 56153 using 4 qubits. At the time, these were the highest numbers factored using quantum algorithms (compare to [22] whose authors factored 21 in 2012 and set the record for the highest integer factored by Shor's algorithm). While impressive upon first glance, these results should not be taken at face value. Both 143 and 56153 were only factored because they were classically reduced to 3 equations. The authors of [8] discovered that semiprimes 143 and 56153 are actually not the only integers that reduced to this number of equations after applying classical preprocessing step. In fact, there is a whole class of integers that reduce to 3 equations (such as 1,099,551,473,989 [18]). Therefore, one could claim any such number as the new factoring record. Similarly, there exists a class of numbers which all reduce to 4 equations, a class of numbers which all reduce to 5 equations and so on.

Anschuetz et al. were the first to apply variational quantum algorithms (QAOA) instead of QAA for the Burges approach and called this method Variational Quantum Factoring (VQF) [1]. In the original paper, they explore VQF performance for semiprimes 35, 77, 1207, 33667, 56153 and 291311. It is shown that the performance of VQF largely depends on the specific instance hardness and equations which it reduces to. The results were inconclusive, scaling and impact of classical preprocessing were not thoroughly analysed. Karamlou et al. recently released a paper [18] that addressed some of these questions. They experimentally investigated the impact different parameters of VQF (number of qubits and circuit depth) had on its performance and cost function landscape. Even then, however, only three instances containing up to 5 qubits were thoroughly analysed, which is too little to draw any sound conclusions about the scaling and practicality of VQF.

3.2 Classical factoring algorithm with quantum subroutine

Similar concerns of the validity of variational quantum factoring method and other experiments were expressed by Mosca et al. [28]. In particular, the paper critiqued the claims of breaking large primes and setting new integer factorization records using quantum algorithms, since the classical preprocessing step was crucial for achieving demonstrated results. Mosca et al. also claim that the practical usefulness of approaches based on factorization problem mapping to an optimization problem is likely not practical for large (or even small) semiprimes. It is noted that algorithms based on Burges method not only require possibly exponential classical preprocessing, but are also unlikely to compete with the classical factoring algorithms. Problems arise even for small instances, which is worrying.

In a related paper [27], Mosca et al. presents another idea for factoring semiprimes using practical quantum algorithms. Instead of mapping to optimization problem, they propose to take the best known classical factoring algorithm, identify its bottleneck and replace that classical subroutine with a quantum subroutine. Therefore, if the quantum subroutine achieves a speedup over the classical subroutine, speedup on the best known classical factoring algorithm can be achieved. This is done in their paper [27] for Number Field Sieve (NFS) by replacing the classical subroutine for finding smooth numbers. Assume that classical NFS runs in time:

$$O(e^{(\sqrt[3]{64/9}+o(1))\ln x^{1/3}(\ln\ln x)^{2/3}}) \tag{3.3}$$

Then they present a circuit based on the Elliptic Curve Method which, if achieved a γ-speedup using quantum algorithms over classical algorithms as defined in [27], would make NFS run in time:

$$O(e^{(\sqrt[3]{\frac{32(\gamma+1)}{9\gamma^2}}+o(1))\ln x^{1/3}(\ln\ln x)^{2/3}}) \tag{3.4}$$

For $\gamma = 2$ (quantum solvers being two times faster than classical solvers), this would mean:

$$O(e^{(\sqrt[3]{8/3}+o(1))\ln x^{1/3}(\ln\ln x)^{2/3}}) \tag{3.5}$$

which is only slightly better than the original NFS. On the other hand, the scaling of number of qubits required is left as an open question due to other difficulties (i.e. converting to QUBO).

3.3 Algorithms for order finding

Order finding problem is precisely the problem that Shor's algorithm is solving to factor semiprimes. To the best of our knowledge, there were no attempts to use prac-

tical quantum algorithms for order-finding problem. Therefore, our work is novel and presents a completely new alternative to VQF and Mosca et al.

Since no quantum benchmarks apart from Shor's algorithm exist, we present a classical brute-force algorithm for order-finding.

Algorithm Brute-force modular order finding algorithm

```
1: Input: N, a
2: Set x = 1
3: repeat
4:    if a^x mod N = 1 then
5:       return x
6:    else
7:       Increment x by one
8:    end if
9: until x == N or solution is found
```

We use this brute-force algorithm as a benchmark for our own algorithms. Let q be the size of N in bits. The runtime complexity analysis of this algorithm is as follows: line 2 takes $O(1)$ time. Loop in lines 3-9 is executed $O(2^q)$ times. Each line within this loop, apart from line 4, takes $O(1)$. Line 4 is the modular exponentiation part, which can be done in polynomial time of the number of bits in x, $O(q^3)$, using Montgomery's algorithm [26, 45]. Thus, the runtime complexity of brute-force algorithm is:

$$O(2^q q^3) \tag{3.6}$$

The algorithm is guaranteed to find the solution, since it can be shown that order r of any element a must divide the total order of the group $\varphi(N)$, therefore r must be between 0 and $N-1$. There are several other classical methods for order-finding "that are not entirely trivial" as pointed out by Cleve [7] which are based on this observation (division of $\varphi(N)$), however no classical polynomial time algorithm is known.

Chapter 4

Algorithm 1 based on QAOA

In this chapter, we present the first variational quantum algorithm for order-finding problem based on QAOA. While VQE can be applied to a broader range of tasks, QAOA is intended to solve combinatorial optimization problems. Since we try to map order-finding problem to an optimization problem in order to express it as an Ising Hamiltonian, QAOA is the natural first choice for our algorithm.

4.1 Theory

As shown in chapter 2, the order-finding problem is equivalent to finding the period r of f given the semiprime to factor N and some element $a \in \mathbb{Z}_N$:

$$f(x) = a^x \bmod N \tag{4.1}$$

where $x \in \mathbb{Z}_N$. To apply QAOA for this problem, we first need to write order finding as an optimization problem. Naturally, we start with the most basic cost function:

$$C_1(x) = (a^x - 1) \mod N \tag{4.2}$$

When $x \equiv 0 \bmod r$, $C_1(x)$ takes the minimum value of 0, otherwise it takes values between 1 and $N-1$. Thus, we try to find $x \neq 0$ that is as close to a multiple of r as possible. We show an example graph for this cost function. Let $a = 2$ and $N = 21$, then the plot of $C_1(x)$ is illustrated in figure 4.1.

To map C_1 into a Hamiltonian, we want to rewrite it as a PBF and convert it to QUBO. Three immediate problems arise:

1. Input is $x \in \mathbb{Z}_N$, whereas the input should be $x \in \{0,1\}^n$
2. Exponential term a^x needs to be expressed as a polynomial
3. Cost function has values modulo N

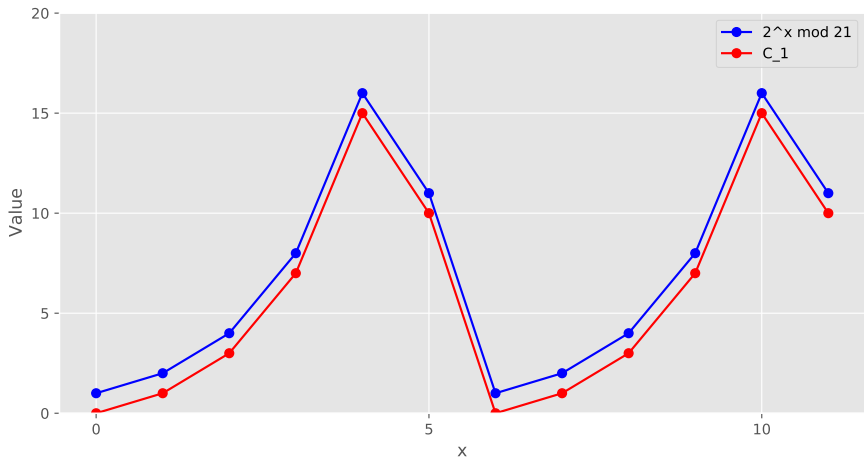


Figure 4.1: Graph for 2^x mod 21 (blue) and C_1 (red). $C_1(x) = 0$ precisely when $x \equiv 0 \bmod r$

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. Although converting eq. 4.2 to its unique polynomial representation directly would mean a simple conversion to QUBO, it would require possibly exponentially many extra variables. We could not find related works which investigated such cost functions with exponential terms and modulo operation for VQAs.

4.1.1 Addressing problem 1

Since $0 \leq x < N$. We can rewrite $x \in \mathbb{Z}_N$ in binary as $\vec{x} = (x_{\lfloor \log(N) \rfloor}, .., x_0)$, i.e. for $N = 21, x = 5 \implies \vec{x} = (0,0,1,0,1)$. Then:

$$a^x = (a)^{x_0} \times (a^2)^{x_1} \times (a^4)^{x_2} \times \ldots \times (a^{2^{\lfloor \log(N) \rfloor}})^{x_{\lfloor \log(N) \rfloor}} = \prod_{i=0}^{\lfloor \log(N) \rfloor} (a^{2^i})^{x_i} \tag{4.3}$$

4.1.2 Addressing problem 2

We notice that we can convert equation (4.3) to a polynomial. Specifically, for some $x_i \in \{0,1\}$, we either have $a^{x_i} = a$ or $a^{x_i} = 1$:

$$a^{x_i} = ax_i + (1 - x_i) = x_i(a-1) + 1 \tag{4.4}$$

Therefore, using equation (4.4) we can rewrite equation (4.3) as:

$$a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} x_i(a^{2^i} - 1) + 1 \tag{4.5}$$

The expanded polynomial expression in eq. 4.5 contains exponentially many terms, which is a problem for the complexity of our algorithm. This also creates interactions

between all combinations of variables x_i. In fact, it appears that the exponential term a^x cannot be written as a polynomial containing polynomially many terms. We try to use Taylor series for a^x which can contain polynomial number of terms, but unsuccessfully due to it only being an approximation of the desired function.

4.1.3 Addressing problem 3

The cost function is currently:

$$C_1(\vec{x}) = (\prod_{i=0}^{\lfloor \log(N) \rfloor} x_i(a^{2^i} - 1) + 1) - 1 \mod N \tag{4.6}$$

C_1 is clearly a PBF, since $\vec{x} \in B^{\lfloor \log(N) \rfloor}$ and the range of $C_1 \in \mathbb{Z}_N$. $\mathbb{Z}_N$ is a subset of $\mathbb{R}$, so $C_1 : B^{\lfloor \log(N) \rfloor} \to \mathbb{R}$. Notice that the polynomial part (inside the modulo) is also a PBF. We want to find a reduction from this PBF modulo N to QUBO modulo N. Let $\vec{w}$ be the auxiliary variables, we try to find a cost function C_1^* such that:

$$C_1(\vec{x}) = \min_{\vec{w}} C_1^*(\vec{x}, \vec{w}) \tag{4.7}$$

where C_1 is defined as in eq. 4.6 (PBF modulo N) and C_1^* is a QUBO modulo N. We fail to find such C_1^*, as the traditional quadratization methods (Freedman & Drineas and Ishikawa) do not work with modulo operation. A specific counter example for Freedman & Drineas: consider a function:

$$f(\vec{x}) = -3x_1x_2x_3 \bmod 21 \tag{4.8}$$

Since $a < 0$, we try to apply reduction by minimum selection with auxiliary variable w:

$$f(\vec{x}) = \min_w f^*(\vec{x}, w) = \min_w(-3w(x_1 + x_2 + x_3) \mod 21) \tag{4.9}$$

Suppose $\vec{x} = [1, 1, 1]$. Then $f(\vec{x}) = 18$. On the other hand, $f^*(\vec{x}, w) = -3w \mod 21$, and clearly the minimum value of $f^*(\vec{x}, w) = 0$ is achieved when $w = 0$. We see that $f(\vec{x}) \neq \min_w f^*(\vec{x}, w)$. Similar counterexample can be constructed for Ishikawa's method.

It is quite clear why these quadratization methods do not work under modulo operation. Reduction by minimum selection is based on the fact that when a is negative, setting $w = 1$ gives the minimum, which is a. If a is positive, on the other hand, setting $w = 0$ gives the minimum, which is 0. Thus, it works as a step function. Under modulo operation, it is always possible to achieve a minimum of 0 by setting $w = 0$. Similar reasoning applies when several terms are involved.

4.2 Discussion

Expressing order-finding problem as an optimization problem is not trivial. Unless efficient quadratization as described in the previous section can be found for C_1 (which we do not find), this cost function cannot be used. C_1 cannot be converted to QUBO using the standard quadratization methods without first converting the PBF to its unique polynomial representation, which introduces additional overhead.

Moreover, unless a^x can be expressed as a PBF using polynomially-many terms, any cost function containing a^x is bound to have exponentially many terms. The only way to mitigate this problem is to **not** have the term a^x in the cost function. If the calculation of a^x was still relevant, it could be performed in a classical way using efficient classical methods, such as Montgomery's algorithm.

Due to the aforementioned problems, we conclude that Algorithm 1 based on QAOA likely cannot be used for order-finding problem.

Chapter 5

Algorithm 2 based on QAOA

In this chapter, we present a second variational quantum algorithm for order-finding problem, which is also based on QAOA.

5.1 Theory

We note that the major flaw of Algorithm 1 is the modulo operation in the cost function. Algorithm 2 can be viewed as a variation of Algorithm 1 without the modulo operation. We use the cost function from eq. 4.2 to arrive at a new cost function C_2:

$$\begin{aligned} a^x - 1 &\equiv 0 \mod N \\ a^x - 1 &= kN \\ C_2(x,k) &= a^x - 1 - kN \end{aligned} \tag{5.1}$$

where $x \in \mathbb{Z}_N$, $k \in \mathbb{Z}$. The variables x,k are correlated. Notice that for some x,k our cost function can get negative values, i.e. $(a^x - 1) < kN$ for $a = 2, x = 2, k = 5, N = 21$. We can solve this problem in two ways. Either square the whole R.H.S.:

$$C_2(x,k) = (a^x - 1 - kN)^2 \tag{5.2}$$

Or take its absolute value:

$$C_2(x,k) = |a^x - 1 - kN| \tag{5.3}$$

Before making a choice between equations 5.2 and 5.3, convert the inputs x,k to binary form $\vec{x},\vec{k}$ similar to what has been done in chapter 4:

$$a^x = \prod_{i=0}^{\lfloor \log(N) \rfloor} x_i(a^{2^i} - 1) + 1 \quad \text{and} \quad k = \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i \tag{5.4}$$

where $\vec{k} = (k_{\lfloor \log(d) \rfloor}, .., k_0)$ the binary representation of k for some $d \in \mathbb{Z}$. Assuming binary inputs $\vec{x}, \vec{k}$, C_2 is a PBF as defined in both equations 5.2 and 5.3. For the choice of cost function, we favour eq. 5.3, since squaring in eq. 5.2 introduces extra terms and extra interactions between x and k. This has a negative effect on algorithm's scaling and usability.

However, eq. 5.3 faces the same quadratization problem as seen with modulo operation for Algorithm 1 using traditional quadratization methods. For example, consider the term $f(x) = |ax_1x_2x_3|$. We cannot find a way to obtain the quadratization f^* using traditional quadratization methods:

$$f(\vec{x}) = \min_{\vec{w}} f^*(\vec{x}, \vec{w}) \tag{5.5}$$

such that f_2^* is QUBO. If we apply either Freedman & Drineas or Ishikawa's method, it is again easy to arrive at a counterexample. We therefore use eq. 5.2 and the final cost function for algorithm 2 looks as follows:

$$C_2(\vec{x}, \vec{k}) = ([\prod_{i=0}^{\lfloor \log(N) \rfloor} x_i(a^{2^i} - 1) + 1] - 1 - N \sum_{i=0}^{\lfloor \log(d) \rfloor} 2^i k_i)^2 \tag{5.6}$$

The PBF in eq. 5.6 can now be reduced to a QUBO in a straightforward manner given a specific problem instance, which can then be mapped to Ising Hamiltonian as described in chapter 2. Depending on the problem instance, a penalty term to this cost function should be added to avoid setting $\vec{x} = \vec{0}$.

5.1.1 Theoretical analysis

The number of qubits required for this algorithm is the number of qubits required to describe x, k and depends on N, a. Consider a problem instance $N = 21, a = 2$, then $0 \leq x \leq N - 1$ and $1 \leq k \leq \frac{2^{N-1}}{N} = d$, which is exponential in N. A reduction from PBF to QUBO introduces extra auxiliary variables, each of which adds an additional qubit. Before applying squaring, C_2 consists of:

- $\sum_{i=1}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i}$ positive terms involving each combination of elements of $\vec{x}$.
- $\lfloor \log(d) \rfloor$ negative terms involving each element of $\vec{k}$.

Once squaring is applied, C_2 consists of the following terms (note that $x^2 = x$ for any $x \in B$):

- $\sum_{i=1}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i}$ positive terms involving only elements of $\vec{x}$ and all of their combinations. Of them, $\sum_{i=3}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i}$ involve more than two variables and need to be quadratized.
- $\lfloor \log(d) \rfloor + \binom{\lfloor \log(d) \rfloor}{2}$ positive terms involving only elements of $\vec{k}$ and all of combinations involving two variables. All of them involve at most two variables.

- $\lfloor \log(d) \rfloor) \sum_{i=1}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i}$ negative terms involving various combinations of elements of $\vec{x}, \vec{k}$. Of them, $\lfloor \log(d) \rfloor) \sum_{i=2}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i}$ involve more than two variables and need to be quadratized.

Using Ishikawa's method, quadratizing a positive term of degree l requires $\lfloor \frac{l-1}{2} \rfloor$ variables. Using Freedman & Dinneas method, quadratizing a negative term requires 1 variable. The number of auxilliary variables in total:

$$\sum_{i=3}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i} \lfloor \frac{i-1}{2} \rfloor + \lfloor \log(d) \rfloor) \sum_{i=2}^{\lfloor \log(N) \rfloor} \binom{\lfloor \log(N) \rfloor}{i} \tag{5.7}$$

Therefore, the total number of variables (and thus qubits) required for cost function in eq. 5.6:

$$\lfloor \lg(N) \rfloor + \lfloor \lg(d) \rfloor + \sum_{i=3}^{\lfloor \lg(N) \rfloor} \binom{\lfloor \lg(N) \rfloor}{i} \lfloor \frac{i-1}{2} \rfloor + \lfloor \lg(d) \rfloor) \sum_{i=2}^{\lfloor \lg(N) \rfloor} \binom{\lfloor \lg(N) \rfloor}{i} \tag{5.8}$$

Adding an extra constraint to not allow $\vec{x} = \vec{0}$ does not add any new terms or variables, since the same terms would already exist prior to quadratization. Therefore, only the scalar value of the terms would change. Runtime complexity cannot be known.

5.2 Experiments

5.2.1 Experiment setup

Exponential scaling of required qubits makes it infeasible to perform experiments using Algorithm 2 even on small instances. For example, consider $N = 21$, $a = 2$, $r = 6$ and $0 \leq x \leq 7$ to limit the range of k to $1 \leq k \leq 2^3$. The total number of qubits required as per eq. 5.8 is 19. This consists of 3 qubits for x, 3 qubits for k and 13 qubits for auxiliary variables. Consider a smaller instance $N = 21$, $a = 4$, $r = 3$ and $0 \leq x \leq 3$, where $1 \leq k \leq 2^3$. The number of qubits required as per eq. 5.8 is 8 qubits. This consists of 2 qubits for x, 3 qubits for k and 3 qubits for auxiliary variables. The QUBO cost function for this problem instance after performing quadratization:

$$\begin{aligned}
&C_{2,qubo}(\vec{x}, \vec{k}, \vec{w}) = ([\prod_{i=0}^{2} x_i(4^{2^i} - 1) + 1] - 1 - 21 \sum_{i=0}^{3} 2^i k_i)^2 = \\
&= (45x_0x_1 + 3x_0 + 15x_1 - 21k_0 - 42k_1 - 84k_2)^2 = \\
&= -126k_0x_0 - 1890w_0k_0 - 1890w_0x_0 - 1890w_0x_1 + 3780w_0 - 630k_0x_1 \\
&\quad - 252k_1x_0 - 504k_2x_0 - 1260k_1x_1 - 2520k_2x_1 - 3780w_1k_1 - 3780w_1x_0 \\
&\quad - 3780w_1x_1 + 7560w_1 - 7560w_2k_2 - 7560w_2x_0 - 7560w_2x_1 + 15120w_2 \\
&\quad + 441k_0 + 1764k_1k_0 + 3528k_2k_0 + 1764k_1 + 7056k_2 + 7056k_1k_2 \\
&\quad + 9x_0 + 3735x_0x_1 + 225x_1 + [1 - x_0 - x_1 + x_0x_1]
\end{aligned} \tag{5.9}$$

We use Qiskit[1] on IBM Quantum Experience[2] to run the experiment. We use the following Qiskit classes provided for us by the library: `QuadraticProgram`, `QAOA` and `MinimumEigenOptimizer`. Note that it is enough to pass $C_{2,qubo}$ to Qiskit, as it takes care of translating a provided QUBO problem to the Ising Hamiltonian by substituting $(I - \sigma_z^i)/2$ for relevant i.

5.2.2 Simulation results

Running on simulation using `QAOA` class we are able to confirm that the optimal solution found to 5.9 is:

$$\vec{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}; \quad \vec{k} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}; \quad \vec{w} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}; \tag{5.10}$$

which correctly corresponds to $x = r = 3, k = 3$ and $C_2(3,3) = (4^3 - 1 - 3 \times 21)^2 = 0$. `QAOA` class does not provide much information about how the solution was obtained (i.e. number of iterations), just that the solution was found. However, running the experiments under setup identical to that presented in the next chapter 6, we do not see good results in terms of convergence.

5.3 Discussion

Although Algorithm 2 works, the cost function C_2 introduces exponentially many variables and is not practically feasible even for small problem instances as shown in simulation results. Such cost function, albeit easily convertable to QUBO, contains exponentially many terms due to (1) the number of elements of $\vec{k}$; (2) the exponential term a^x which has to be converted to a polynomial; (3) squaring of the R.H.S. as in eq. 5.2; and (4) the resulting number of auxiliary variables once quadratization is performed. Even the smallest instance used in section 5.2 requires 8 qubits, of which only 2 qubits encode x. With larger problem instances, proportion of qubits describing x would become even smaller. Therefore, Algorithm 2 does not look promising for practical applications.

Our algorithm cannot compare to the classical brute-force algorithm presented in chapter 3 due to scaling. More in-depth analysis of experiments could be performed, but due to scaling limitations we do not investigate this algorithm further. It is unlikely that the algorithm would give good results for larger instances.

To conclude algorithms 1 and 2, we fail to find a way to map order-finding to an optimization problem without exponential scaling of auxiliary variables or terms.

[1] `https://qiskit.org`
[2] `https://quantum-computing.ibm.com`

Chapter 6

Algorithm 3 based on VQE

In this chapter, we explore a variational quantum algorithm for order-finding problem based on VQE. This algorithm takes a completely different approach from the previous algorithms presented in chapters 4, 5.

6.1 Theory

As we have already seen in previous chapters, Algorithms 1 and 2 have significant scaling limitations. It may be useful to take a different approach and try VQE instead of QAOA. We present the following algorithm based on VQE:

Algorithm Order-finding algorithm based on VQE

Input: a, semiprime N
Randomly generate p initial parameters $\vec{\theta_0} = \begin{bmatrix}\theta_0 & \dots & \theta_p\end{bmatrix}$
for $t \in [0, T]$ **do**
  Prepare state $|\psi(\theta_t)\rangle$ using variational form circuit
  Set $C_{total} = 0$
  for K times **do**
    Measure $|\psi(\theta_t)\rangle$ on computational basis and obtain a computational basis state $|x\rangle$ where x is the bit string of the computational basis state
    Evaluate cost of the state $C_3(x)$ and increase total cost $C_{total} = C_{total} + C_3(x)$
  end for
  Let $C_{avg} = C_{total}/K$ be the average cost of the state $|\psi(\theta_t)\rangle$
  Given C_{avg}, update the parameters θ_t to θ_{t+1}
end for
return Final set of parameters θ_T

where the number of parameters required by the variational form circuit p; the number of shots for measurement K; the number of iterations for optimization until convergence T; and $C(x)$ the cost function. Essentially, the goal of the algorithm is to move the probabilities of the trial state from worse to better computational basis states. There is no Hamiltonian to drive the parameter search, since we were unable to find any good

Hamiltonians for order-finding problem as shown in previous chapters. The parameter search is guided directly by $C_3(x)$, which can be efficiently evaluated classically. Some parts of the algorithm, such as the choice of cost function, are left ambiguous on purpose. These parts can have a significant impact on the performance of the algorithm and are investigated experimentally.

6.1.1 Theoretical analysis

The number of qubits required for this algorithm is the number of bits in the semiprime we want to factor N:

$$q = \lceil \log_2 N \rceil \tag{6.1}$$

This is the best scaling we could practically hope to do. The runtime of the algorithm:

$$O(Tp + TKq^3) \tag{6.2}$$

Referring to Algorithm 3: line 2 is $O(p)$. The loop at line 3 is executed T times, where T is the number of iterations before convergence. Inside this loop, lines 5 and 10 take $O(1)$. We assume lines 4 and 11 take $O(p)$, since they deal with p parameters. The loop at line 6 takes K times. During each iteration, we assume that line 7 takes $O(1)$ and line 8 takes $O(q^3)$ using Montgomery's algorithm given our choices for cost function.

The number of parameters, p, depends on the chosen ansatz and depth of the circuit. For the purposes of this work, we can assume it is polynomial in the number of qubits. K can be set in advance, whereas T cannot be known in advance (because it is heuristics). Thus, the complexity of the algorithm boils down to:

$$O(TKq^3) \tag{6.3}$$

T, K determine the runtime complexity of the algorithm. Compare eq. 6.3 to the brute-force runtime complexity in eq. 3.6. For the algorithm to be at least as good as brute-force, TK must be at most exponential in q. To compete with the best-known classical factoring algorithm, TK should be subexponential. It is expected that K is not constant and it is reasonable to assume that the average cost of a state could be determined using polynomial number of shots. This is a direct result of Hoeffding's inequality [15]. We can view the state of our quantum system as a classical random variable, since the measurements are taken in the computational basis state, and we know that these measurement runs are independent and identically distributed random variables. Hoeffding's inequality says that under such assumptions, it is enough for the number of measurements to scale polynomially such that the empirical mean of random variables deviates from the expected value by some exponentially small term. Not much can be said about T, however, and we aim to infer its scaling experimentally from smaller problem instances.

6.1.2 Choice of cost function

Several cost functions could be considered for our algorithm giving different performance. The basic cost function:

$$C_3(x) = \begin{cases} N-1, & \text{if } x = 0 \\ (a^x - 1) \bmod N, & \text{otherwise} \end{cases} \tag{6.4}$$

where cost for the trivial solution $x = 0$ is set to some non-zero cost. We could consider a symmetric version of $C(x)$, where the worst cost is $\frac{N-1}{2}$:

$$C_{3,s}(x) = \begin{cases} \frac{N-1}{2}, & \text{if } x = 0 \\ \frac{N-1}{2} - |(a^x - 1 \bmod N) - \frac{N-3}{2}|, & \text{if } (a^x - 1 \bmod N) > \frac{N-1}{2} \\ \frac{N-1}{2} - |(a^x - 1 \bmod N) - \frac{N-1}{2}|, & \text{otherwise} \end{cases} \tag{6.5}$$

Such symmetric cost function is based on observation that, under modulo operation, cost of $(1 \bmod N)$ may be as good as $(N-1 \bmod N)$. For example, $x \equiv 10 \bmod 21$ and $x \equiv 11 \bmod 21$ are equally close to $x \equiv 0 \bmod 21$ as illustrated in the following figure 6.1:

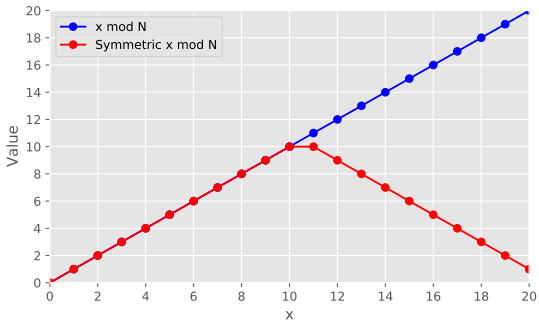


Figure 6.1: Values for x mod 21 (blue) and how close they are to 0 as converted by symmetric function (red)

Symmetric version of $2^x \mod 21$ is shown in the following figure 6.2:

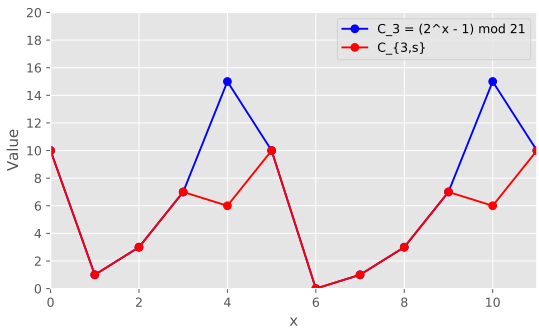


Figure 6.2: Cost function $C_3(x) = 2^x - 1 \bmod 21$ (blue) and its symmetric version $C_{3,s}$ (red)

This specific cost function does not have large values modulo N near the zeros, so the effect is not as visible. In fact, a symmetric cost function with some adjustments could be applied to Algorithms 1 and 2. However, even more new terms and auxiliary variables could have to be introduced. Another possible cost function is the so-called binary cost function that gets rid of local minimas and makes the algorithm perform like a brute-force search:

$$C_{3,b}(x) = \begin{cases} 1, & \text{if } (a^x - 1 \bmod N) > 0 \text{ or } x = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.6}$$

6.1.3 Choice of classical optimization

The choice of optimizer has an impact on the convergence of our algorithm. It may be that the task is more suitable for some specific classes of optimizers. Both gradient-based and gradient-free optimizers are used in the context of VQAs, most popular ones being COBYLA [34], Nelder-Mead [29] and traditional gradient descent or its variations like ADAM [19]

6.1.4 Choice of ansatz

For this algorithm, we only consider the hardware-efficient ansatz [17] presented in chapter 2. As name suggests, the states of this ansatz are prepared by gates which are tailored to the current physical devices. Moreover, all computational basis states can be achieved through such ansatz, therefore it is a good choice. We consider the following rotation and entanglement gates for the hardware-efficient ansatz: $(\{R_Y\},\{CX\})$, $(\{R_Y\},\{CZ\})$, $(\{R_Y,R_Z\},\{CZ\})$. In each case, we set a full entanglement. Choice of the depth of the circuit d is determined experimentally, though shallow circuits are of course preferred.

6.1.5 Choice of initial state

Two possible initial states for such algorithm can be considered: (1) Random initial state, with the possibility of having 0 probability on the solution state; (2) The superposition of all computational basis states (identical to applying Hadamar gate to every qubit), ensuring non-zero probability for the solution state.

As number of qubits grows, approach (1) means that most of the states will have 0 probabilities. In approach (2) probabilities for all states go to 0 (each state has probability of $\frac{1}{2^q}$).

6.1.6 Choice of encoding: binary vs. Gray

Computational basis states are encoded in binary. A flaw of binary encoding is that two successive decimal numbers may be “far away” in binary, where the distance between

two numbers in binary refers to Hamming distance[1] in bits. Consider $7 = 0111_2$ and $8 = 1000_2$, where Hamming distance is 4 bits.

This problem is solved by Gray code [14], where two successive decimal numbers have Hamming distance of 1 bit. Under Gray code, $7 = 0100_2$ and $8 = 1100_2$ with Hamming distance of 1 bit as required. It is interesting to explore whether Gray code of computational basis states could improve our algorithm. In fact, Matteo et al. have very recently (April 2021) released a paper investigating the impact of Gray code on the Hamiltonian encoding [9].

6.2 Experiments

Since we are interested in the potential time complexity estimation of Algorithm 3, we focus our attention on the scaling of K, T under different hyperparameter settings: ansatz, optimizer, cost function and encoding.

6.2.1 Performance measure

We define custom metrics for evaluating algorithm's performance. Performance is measured as a success ratio. Denote the set of all bit strings obtained from measuring some state as B, where $b \in B$ is a single bit string. Denote all correct bit strings as the set $B_s = \{b \in B | C_3(b) = 0\}$. In other words, B_s is the set of all bit strings that give the cost of 0.

Definition 6.2.1. Success ratio is a function that takes value of 1 if at least one bit string in the measurement of the final state contains the correct solution, else 0.

$$s(|\psi(\theta)\rangle) = \begin{cases} 1, & \text{if } \exists b \in B \text{ s.t. } C_3(b) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.7}$$

Definition 6.2.2. Success ratio is a function that is equal to the ratio of correct bit strings in all observed bit strings when measuring the final state.

$$s(|\psi(\theta)\rangle) = \frac{|B_s|}{|B|} \tag{6.8}$$

According to the definition 6.2.1, we care about finding at least one correct bit string. According to the definition 6.2.2, we care about the proportion of correct bit strings. Both definitions are suitable for performance measure, however the latter gives more information and is more useful.

6.2.2 Experiment setup

Problem instances used in all experiments are given in table 6.1. The instances are chosen at random, therefore no assumptions about the hardness of these instances are

[1] https://en.wikipedia.org/wiki/Hamming_distance

made. Number of qubits for each instance depends on N. For example, we use 6 qubits for $N = 35$ because $2^6 > 35 > 2^5$. We investigate problem instances up to 12 qubits due to computational constraints.

Qubits	Instance 1	Instance 2
3	(2,15)	(4,15)
4	(2,15)	(4,15)
5	(2,21)	(8,21)
6	(2,35)	(3,35)
7	(2,115)	(7,115)
8	(2,205)	(7,205)
9	(2,437)	(12,437)
10	(2,703)	(15,703)
11	(2,1795)	(17,1795)
12	(2,3183)	(19,3183)

Table 6.1: Problem instances used in the experiments. Each instance is of form (a,N) such that $a^x \bmod N$

As presented in previous section, we investigate the following choice of hyperparameters for performance impact and scaling of K:

- Cost functions: $C_3, C_{3,s}$
- Hardware-efficient ansatz type: $(\{R_Y\},\{CX\})$, $(\{R_Y\},\{CZ\})$ $(\{R_Y,R_Z\},\{CZ\})$
- Encoding: binary and Gray
- Optimizer: COBYLA, Nelder-Mead, SPSA [39] and Adam

Although more hyperparameters could be explored, the scope of possible combinations would quickly become too large to test given computational resources available. The experiments under a chosen hyperparameter combination are performed as follows: for each problem instance, the algorithm is repeatedly run 10 times and the performance is averaged out. After that, average performance of each problem instance is averaged out over all problem instances with the same number of qubits. For example, to get the average performance for 3 qubits as presented in table 6.1, Algorithm 3 would run 10 times for both instances (2,15) and (4,15), obtaining average performance for each instance, which are then averaged again to obtain the average performance over 3 qubits.

All simulation and hardware experiments are performed using Qiskit on IBM Quantum Experience.

6.2.3 Simulation results: scaling of K,T

As already mentioned, K being constant is not expected to give good performance. Therefore, we try a variety of values for K to see what increase of K could give a small decrease in average performance as the number of qubits keeps increasing. Moreover, we observe how T changes as K and number of qubits change. Having identified the increase in K,T for smaller instances, we could extrapolate scaling for larger instances.

We explore $K \in \{100, 500, 1000, 2500, 5000, 7500\}$. This range of K covers different ranges of qubits. If $K = 100 \approx 2^7$, then the size of solution space becomes larger than K at around 7 qubits. If $K = 7500 \approx 2^{13}$, then the size of solution space becomes larger than K at around 13 qubits. We expect that larger K would still achieve relatively good performance for instances of 10 qubits and more. Results for this experiment under the best identified combination of hyperparameters are presented in figure 6.3. Obtaining the best combination of hyperparameters used throughout this subsection is presented in the next subsection 6.2.4.

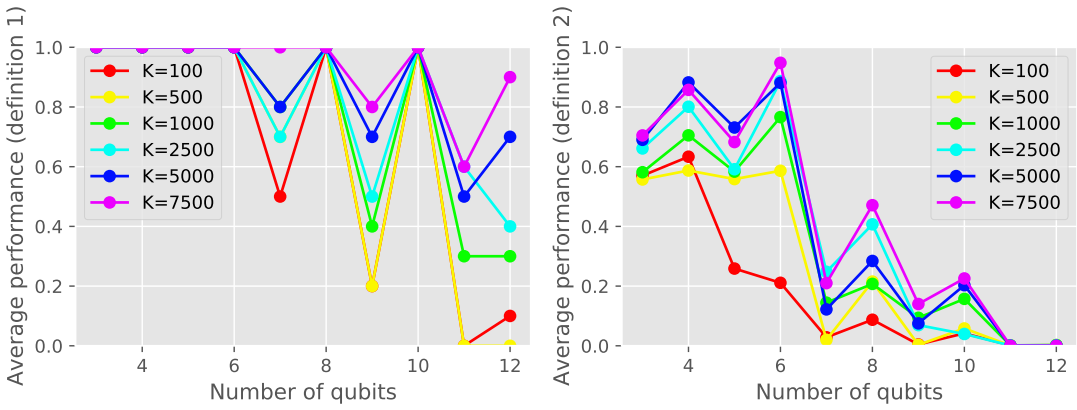


Figure 6.3: Performance metrics of Algorithm 3 for provided problem instances using optimizer COBYLA, ansatz R_Y, CX, $C_{3,s}$ and binary encoding with various K. Left: number of qubits vs. average performance as in definition 6.2.1. Right: number of qubits vs. average performance as in definition 6.2.2

First, notice from figure 6.3 that if K is kept constant for smaller K, then the average performance drop happens relatively fast as expected. For example, consider $K = 100$ set as constant as shown in figure 6.4:

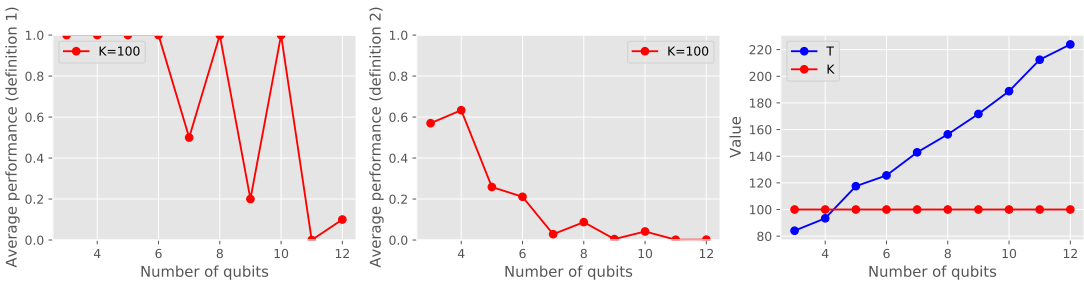


Figure 6.4: Performance metrics of Algorithm 3 for provided problem instances using optimizer COBYLA, ansatz R_Y, CX, $C_{3,s}$ and binary encoding with $K = 100$. Left: number of qubits vs. average performance as in definition 6.2.1. Center: number of qubits vs. average performance as in definition 6.2.2. Right: number of qubits vs. K and T

As expected, a sharp average performance drop can be seen at around 7 qubits by both definitions. The algorithm struggles to find solution as per definition 6.2.1 for larger problem instances. Interestingly, notice that the number of iterations T that the optimizer takes increases linearly when K is constant. This, in fact, has been observed for all constant K.

However, from figure 6.3 it seems that even with increased number of shots the algorithm still fails to converge on larger instances. Consider setting $K = 7500$ as constant to observe the average performance in figure 6.5.

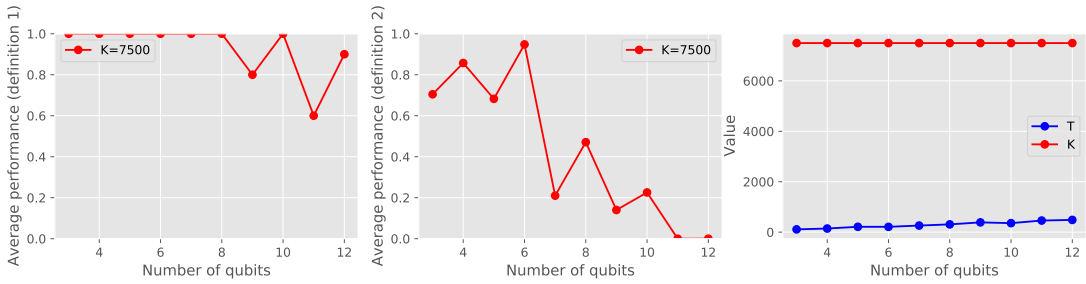


Figure 6.5: Performance metrics of Algorithm 3 for provided problem instances using optimizer COBYLA, ansatz R_Y, CX, $C_{3,s}$ and binary encoding with $K = 7500$. Left: number of qubits vs. average performance as in definition 6.2.1. Center: number of qubits vs. average performance as in definition 6.2.2. Right: Number of qubits vs. K and T

The average performance starts decreasing at around 9 qubits, even though $K = 7500$ still spans the whole search space. We show performance on a particular problem instance to inspect the cause of such average performance loss for $K = 7500$ in figure 6.6 and for $K = 500$ in figure 6.7.

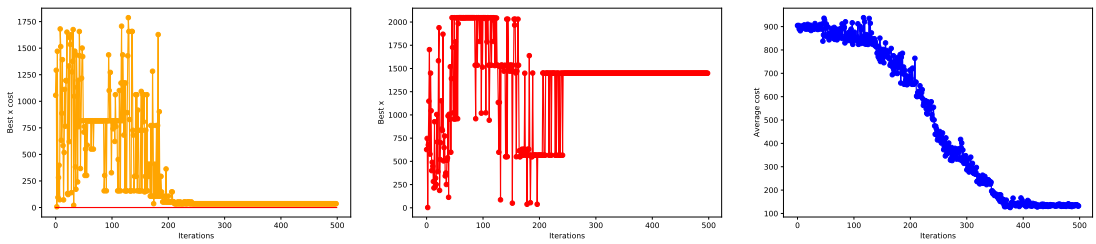


Figure 6.6: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 1795 using optimizer COBYLA, ansatz R_Y, CX, $C_{3,s}$, binary encoding, 11 qubits and $K = 7500$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

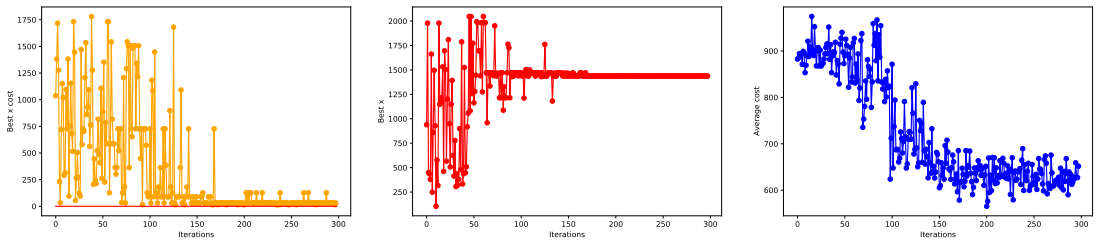


Figure 6.7: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 1795 using optimizer COBYLA, ansatz R_Y, CX, $C_{3,s}$, binary encoding, 11 qubits and $K = 500$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

In both cases, the average cost decreases over time and converges as can be seen on the right graphs of both figures. Notice that the higher K gives less variation in datapoints, but longer running time until convergence. However, the average cost in the right graph of both figures does not converge to 0 for both $K = 500$ and $K = 7500$. The best solution cost in the left graph does not converge to 0 either, but close to 0, which is a local minima.

6.2.4 Simulation results: finding optimal hyperparameters

As already mentioned, hyperparameters have an impact on the average performance of the algorithm. For example, consider one of the most important choices for our algorithm: optimizer. Let $N = 35$, number of qubits $q = 6$ and $a = 2$. An example run of the algorithm on such instance using COBYLA optimizer is presented in figure 6.8:

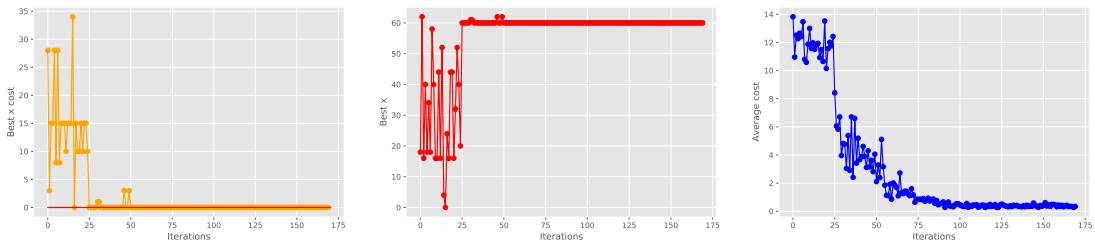


Figure 6.8: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 35 using optimizer COBYLA, ansatz $R_Y, CX, C_{3,s}$, binary encoding, 6 qubits and $K = 1000$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

The performance looks good and the average cost decreases smoothly as seen in the graph on the right. The same problem instance with ADAM optimizer is presented in figure 6.9:

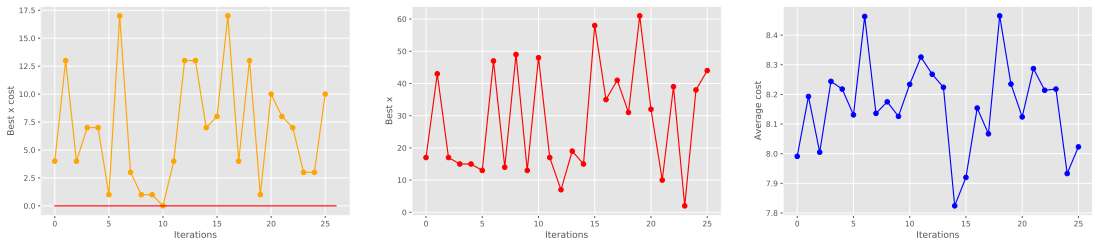


Figure 6.9: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 35 using optimizer ADAM, ansatz $R_Y, CX, C_{3,s}$, binary encoding, 6 qubits and $K = 1000$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

In this case, performance is not as smooth. The algorithm fails to reduce the average cost as seen in the graph on the right. The same problem instance with SPSA optimizer is presented in figure 6.10:

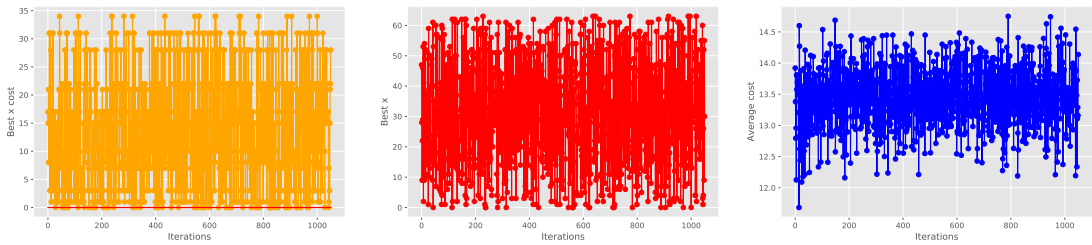


Figure 6.10: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 35 using optimizer SPSA, ansatz R_Y, CX, $C_{3,s}$, binary encoding, 6 qubits and $K = 1000$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

SPSA optimizer, clearly, is unusable. Therefore, we show that some optimizers can be discarded at relatively smaller instances if the performance is unsatisfactory.

We find that the choice of specific type of hardware-efficient ansatz and encoding do not make a significant difference in performance. However, since simpler circuits and less parameters are prefered, choice of R_Y rotation gates and CX entanglement gates seems reasonable. We also find that circuit depths of 2, 3 and 4 produce relatively similar results for investigated instances. Lastly, $C_{3,s}$ performs slightly better than C_3, but more in-depth analysis is required.

6.2.5 Quantum hardware results

Long periods of access to quantum hardware is still relatively uncommon. Therefore, we only use quantum hardware to test performance of our algorithm on several chosen problem instances. Moreover, we do not optimize the circuits on quantum hardware. Instead, we obtain optimal parameters for the algorithm in simulation and only use quantum hardware to run the circuit with the optimal parameters. We consider the following problem instances:

$$\begin{aligned} C(x) &= (2^x - 1) \bmod 21 \quad \text{(with 3 qubits)} \\ C(x) &= (2^x - 1) \bmod 35 \quad \text{(with 6 qubits)} \\ C(x) &= (2^x - 1) \bmod 437 \quad \text{(with 9 qubits)} \end{aligned} \tag{6.9}$$

We obtain optimal parameters using COBYLA optimizer, $\{RY\}, \{CX\}$ hardware-efficient ansatz, $C_{3,s}$ and $K = 1000$. Variational quantum circuit model with optimal parameters for the instance of 3 qubits is presented in figure 6.11.

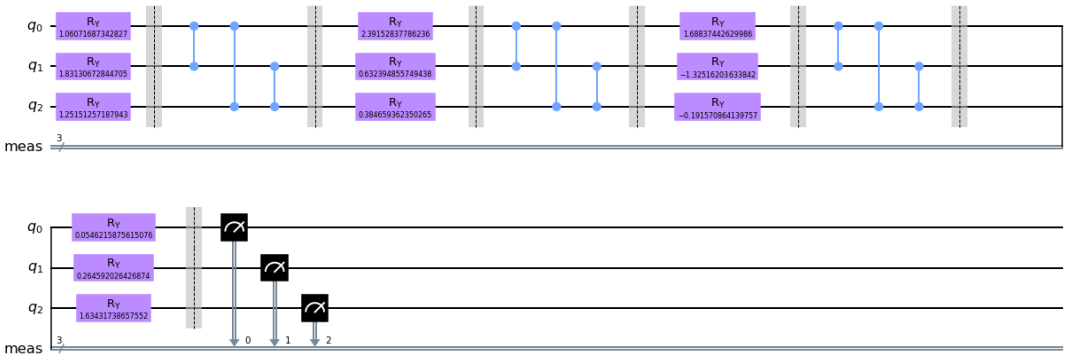


Figure 6.11: Optimal circuit for $C(x) = (2^x - 1) \bmod 21$ with 3 qubits

The simulation and hardware experiment results for this circuit are presented in figure 6.12. In simlation, the correct solution ($110_2 = 6$) obtained with overwhelming probability (94%). On real quantum hardware (`ibmq_santiago`), the correct solution ($110_2 = 6$) is also found, albeit it is less exact (60%) than the simulation result.

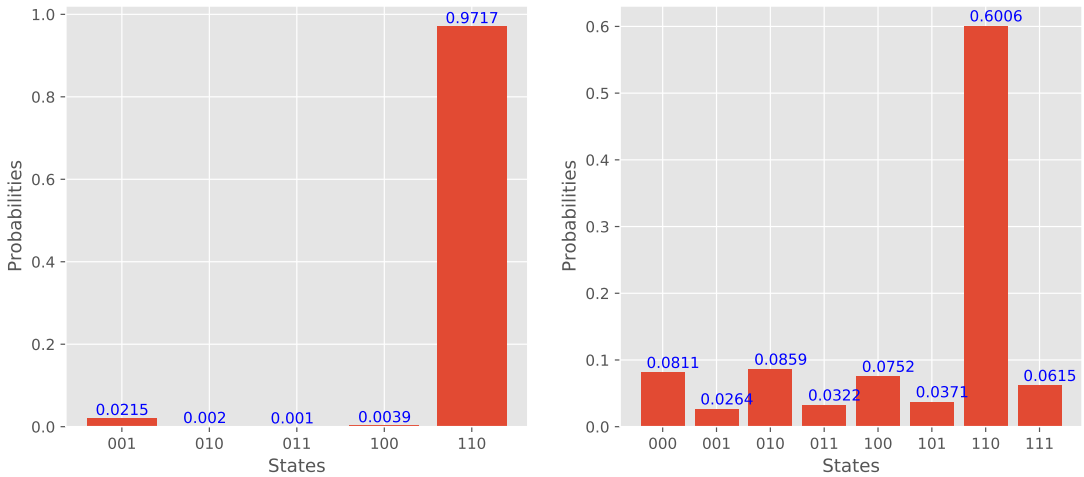


Figure 6.12: Measurement results for $C(x) = (2^x - 1) \bmod 21$ with 3 qubits using optimal parameters. Left: Simulation. Right: Hardware (`ibmq_santiago`)

Experiments for other two instances with 6 and 9 qubits, while show perfect simulator results, do not give good results when ran on the real hardware. We claim this is due to noise, as the circuits for these problems become quite deep even for today's standards. Attempts could be made to obtain an optimal circuit with smaller depth and not full entanglement.

6.3 Discussion

As demonstrated experimentally, algorithm 3 does work for smaller instances with good scaling of required qubits and runtime. However, it does not currently work past 10 qubits. Therefore, at the moment we cannot infer the scaling of K or T. This

behaviour could be explained by the fact that we do not use Hamiltonians to guide the search and the convergence is not guaranteed. If the cost function landscape is hard to optimize, it may simply be not possible to reach global optimum. We claim that, as the number of qubits increase, the algorithm gets stuck in local optima because cost function landscape does not give enough information to achieve global optima. In other words, being close to the solution does not necessarily translate into achieving a good solution under the considered cost functions. Due to smaller search space dimensions, this is not a problem for smaller instances.

We also note that hardware results are significantly more noisy for the optimal parameters. The most probable state for 3 qubits goes from 94% in simulator to 60% on real hardware. A more in-depth analysis of different noise models could help explore the impact of noise on the training on hardware. A more thorough hyperparameter search (including depth of the circuit) would also be beneficial.

Another thing to mention is that the explored cost functions can find not only the period r, but also multiples of the period, as there are no constraints imposed. This is not good for functions that are highly periodic, but totally fine for functions that are **not** highly periodic (which are usually the harder instances). One way to mitigate it for more periodic functions is to make a strong assumption in advance about $0 < x < 2r$. Another way to mitigate this is to come up with an improved cost function that has some additional penalty term to the $a^x - 1 \bmod N$. It is possible that such cost function may even give better performance for 10 and more qubits by avoiding the local minima.

As a side note, another possible improvement to Algorithm 3 is completely avoiding the order-finding itself and finding the factors directly through a variation of Algorithm 3. Consider the following cost function where $N = pq$:

$$C_{3,new}(p,q) = (a^{(p-1)(q-1)} - 1) \bmod N \tag{6.10}$$

which comes from Euler's theorem $a^{\phi(N)} \equiv 1 \bmod N$ and $\phi(N) = (p-1)(q-1)$. However, this is just "food for thought" and has not been experimentally or theoretically evaluated.

Chapter 7

Application example: factoring semiprime 437 using 9 qubits

In this chapter, we factor semiprime 437 using Algorithm 3 in simulation for illustration purposes. Note that the experiments presented in chapter 6 have already in some cases factored various semiprimes. For example, 21 as shown in figure 6.12, where we find that $r = 6$. Clearly, $\gcd(2^3 - 1, 21) = 7$ and $\gcd(2^3 + 1, 21) = 3$, which are the factors of $7 \times 3 = 21$. Not many simulation experiments were presented for Variational Quantum Factoring in [1, 18] apart from factoring numbers reducing to 3, 4 and 5 equations. Thus, we show that we can factor semiprimes (such as 437) that do not necessarily reduce to these numbers of equations.

Factoring example follows the reduction from factoring to order-finding problem as described in appendix A. Due to lack of space and for the purposes of brevity, we do not simulate randomly choosing the element a. This is because not all a can be used for factoring a chosen semiprime (period of a cannot be odd for instance).

Let $N = 437, a = 2$. We use 9 qubits since $2^8 < 437 < 2^9$. Then use Algorithm 3 to obtain optimal parameters $\vec{\Theta}$ of the variational form. Optimization is shown in figure 7.1.

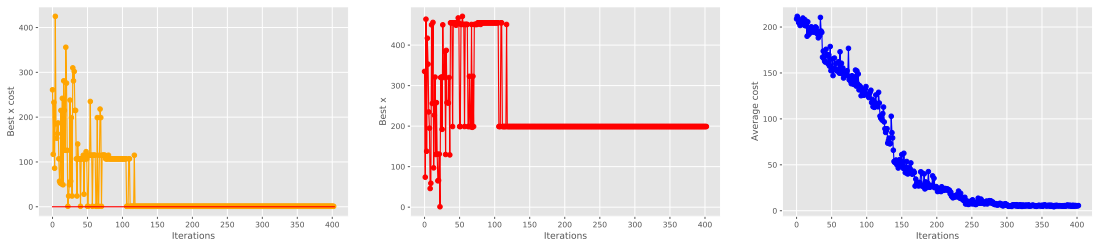


Figure 7.1: Performance metrics of Algorithm 3 for a specific problem instance 2^x mod 437 using optimizer COBYLA, ansatz $R_Y, CX, C_{3,s}$, binary encoding, 9 qubits and $K = 1000$. Left: the cost of the most probable computational basis state over time. Center: the most probable computational basis state over time. Right: average cost C_{avg} of the state over time

Exact information (optimal parameters and circuit design) for reproduction can be

found in appendix B. Measurement results of the optimal parameters are given in figure 7.2.

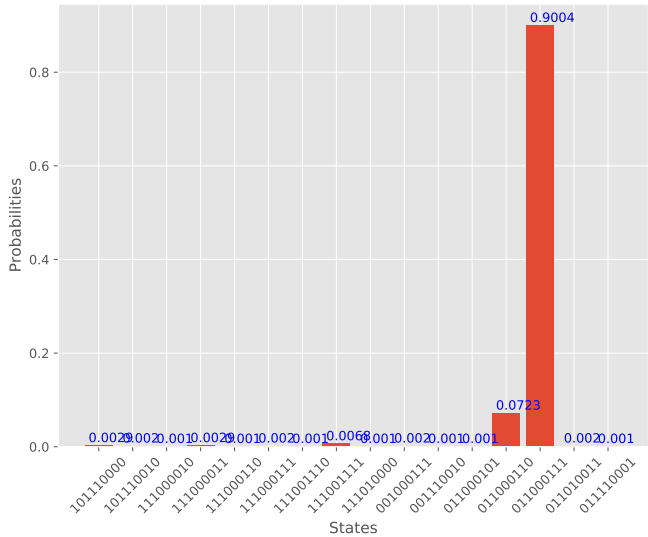


Figure 7.2: Measurement results for 2^x mod 437 with 9 qubits using optimal parameters

The two most likely bitstrings are $011000111_2 = 199$ with probability of 89% and $011000110_2 = 198$ with probability of 7.4%. In fact, evaluating these two bitstrings classically shows that the one with smaller probability is correct:

$$2^{199} \equiv 2 \bmod 437; \quad 2^{198} \equiv 1 \bmod 437 \implies r = 198 \tag{7.1}$$

Interestingly, the algorithm converged on a state that gave more probability to the computational basis state that has a higher cost. We apply the next steps of the algorithm from A. Namely, r is not odd and $2^{99} \not\equiv -1 \bmod 437$, therefore:

$$\gcd(2^{99} - 1, 437) = 23 \quad \text{and} \quad \gcd(2^{99} + 1, 437) = 19 \tag{7.2}$$

which are the correct factors of 437, since $23 \times 19 = 437$.

Chapter 8

Conclusions and Future Work

Having reviewed recent advances in factorization using practical quantum algorithms and having presented three novel algorithms for order-finding problem, we reiterate the fact that finding a good algorithm is not trivial in both classical and quantum worlds.

We find that QAOA-based algorithms, such as Algorithms 1 and 2, cannot be used for order-finding, unless better cost functions are found or limitations of current cost functions are addressed. This is hard to do, since such cost functions should likely not involve modulo operator (either directly or for conversion to the unique polynomial representation) and the exponential term a^x.

VQE-based algorithms could work as demonstrated by Algorithm 3, however further improvements are needed for larger instances to ensure solutions are found with decent probability and local minima are avoided. This is subject of further research. Although a wider hyperparameter search (i.e. choice of optimizer and depth of the circuit) could prove itself useful, we believe the major performance improvement should come from the choice of a different cost function. New problem instances, including "harder" and "easier" ones, could also be used to evaluate the average performance more thoroughly. An improvement that could make the algorithm converge to the correct solution with a decent average performance with up to 50 qubits, even if not for the "very hard" problem instances, could still prove to be useful from cryptanalysis perspective, as this would already be beyond the capabilities of majority of classical computers.

8.1 Future work

Variational Quantum Algorithms are at the heart of many recent advances in Quantum Computing. While they may not show much promise for order-finding or integer factorization at the moment, the knowledge gained from this work is applicable beyond the aforementioned problems.

Due to this reason, it is beneficial to take a more general look and understand various properties of variational quantum algorithms. Especially important is understanding how these algorithms scale and what obstacles they face as the number of qubits increases. Some researchers have recently found that VQAs suffer from noise-induced

barren plateaus [40] and noise-free barren plateaus [23].

Authors of [23] find that some ansatz, such as the hardware efficient ansatz used in our work, present a vanishing gradient problem as the number of qubits and depth of the circuit increases. To overcome noise-free barren plateaus, the runtime complexity of algorithms may become exponential. A more recent work by Cerezo et al. [6] found that noise-free barren plateaus are cost function dependent and can be observed for shallower circuits as well. Therefore, even if a theoretically good cost function is found, it is not guaranteed to work in practice. Authors of [40] found that vanishing gradients can also be caused by noise from NISQ devices. Both types of barren plateaus pose a huge trainability problem for VQAs.

As the number of qubits in NISQ devices grow and new applications of VQA are invented, trainability problems due to both noise-free and noise-induced barren plateaus are going to inevitibly show up. An open question thus remains: even if algorithms for integer facorization or order-finding (or any other problem) can run on NISQ devices, can they be trained efficiently and find the solution given the problem of vanishing gradient? Even more importantly, how does the physical imperfections of NISQ devices affect the performance of these algorithms?

Bibliography

[1] Eric R. Anschuetz, Jonathan P. Olson, Alán Aspuru-Guzik, and Yudong Cao. Variational quantum factoring, 2018. arXiv:1808.08927 [quant-ph].

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, and et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019.

[3] Panagiotis Kl. Barkoutsos, Giacomo Nannicini, Anton Robert, Ivano Tavernelli, and Stefan Woerner. Improving Variational Quantum Optimization using CVaR. *Quantum*, 4:256, April 2020.

[4] Rodríguez-Heck E. Boros E., Crama Y. Compact quadratizations for pseudo-boolean functions. *Journal of Combinatorial Optimization*, 39, 04 2020.

[5] Chris J.C. Burges. Factoring as optimization. Technical Report MSR-TR-2002-83, January 2002.

[6] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1), Mar 2021.

[7] Richard Cleve. The query complexity of order-finding. In *In Proceedings of the Fifteenth IEEE Conference on Computational Complexity*, pages 54–59, 2000.

[8] Nikesh S. Dattani and Nathaniel Bryans. Quantum factorization of 56153 with only 4 qubits, 2014. arXiv:1411.6758 [quant-ph].

[9] Olivia Di Matteo, Anna McCoy, Peter Gysbers, Takayuki Miyagi, R. M. Woloshyn, and Petr Navrátil. Improving hamiltonian encodings with the gray code. *Physical Review A*, 103(4), Apr 2021.

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014. arXiv:1411.4028 [quant-ph].

[11] Edward Farhi and Aram W Harrow. Quantum supremacy through the quantum approximate optimization algorithm, 2019. arXiv:1602.07674 [quant-ph].

[12] Gutmann S. Sipser M. Farhi E., Goldstone J. Quantum computation by adiabatic evolution, 2000. arXiv:quant-ph/0001106.

[13] D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 939–946 vol. 2, 2005.

[14] F. Gray. Pulse code communication, 1953. US Patent 2,632,058.

[15] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[16] H. Ishikawa. Transformation of general binary mrf minimization to the first-order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249, 2011.

[17] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, Sep 2017.

[18] Amir H. Karamlou, William A. Simon, Amara Katabarwa, Travis L. Scholten, Borja Peropadre, and Yudong Cao. Analyzing the performance of variational quantum factoring on a superconducting quantum processor, 2021. arXiv:2012.07825 [quant-ph].

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[20] Mark Lewis and Fred Glover. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis. *Netw.*, 70(2):79–97, September 2017.

[21] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[22] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realization of shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776, Oct 2012.

[23] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), Nov 2018.

[24] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, feb 2016.

[25] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess,

Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, jun 2018.

[26] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[27] Michele Mosca, João Marcos Vensi Basso, and Sebastian R. Verschoor. On speeding up factoring with quantum SAT solvers. *CoRR*, abs/1910.09592, 2019.

[28] Michele Mosca and Sebastian R. Verschoor. Factoring semi-primes with (quantum) sat-solvers. *CoRR*, abs/1902.01448, 2019.

[29] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[30] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[31] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), Jul 2014.

[32] Sergiu Rudeanu Peter L. Hammer. *Boolean Methods in Operations Research and Related Areas*. Springer, 1968.

[33] Carl Pomerance. A tale of two sieves. *NOTICES AMER. MATH. SOC*, 43:1473–1485, 1996.

[34] Michael J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J. P. Hennart, editors, *Advances in Optimization and Numerical Analysis*, volume 275 of *Mathematics and its Applications*, pages 51–67. Springer, 1994.

[35] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, Aug 2018.

[36] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[37] Gernot Schaller and Ralf Schützhold. The role of symmetries in adiabatic quantum algorithms, 2009. arXiv:0708.1882 [quant-ph].

[38] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[39] J. C. Spall. A stochastic approximation technique for generating maximum likelihood parameter estimates. In *1987 American Control Conference*, pages 1161–1167, 1987.

[40] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-induced barren plateaus in variational quantum algorithms, 2021. arXiv:2007.14384 [quant-ph].

[41] Zhihui Wang, Stuart Hadfield, Zhang Jiang, and Eleanor G. Rieffel. Quantum approximate optimization algorithm for maxcut: A fermionic view. *Phys. Rev. A*, 97:022304, Feb 2018.

[42] Wikipedia. Adiabatic theorem. `https://en.wikipedia.org/wiki/Adiabatic_theorem`. (Accessed on 04/06/2021).

[43] Wikipedia. Hamiltonian (quantum mechanics). `https://en.wikipedia.org/wiki/Hamiltonian_(quantum_mechanics)`. (Accessed on 04/06/2021).

[44] Wikipedia. Ising model. `https://en.wikipedia.org/wiki/Ising_model`. (Accessed on 04/06/2021).

[45] Wikipedia. Montgomery modular multiplication. `https://en.wikipedia.org/wiki/Montgomery_modular_multiplication`. (Accessed on 04/11/2021).

[46] Wikipedia. Variational method (quantum mechanics). `https://en.wikipedia.org/wiki/Variational_method_(quantum_mechanics)`. (Accessed on 04/06/2021).

[47] Heather Woll. Reductions among number theoretic problems. *Information and Computation*, 72(3):167–179, 1987.

[48] Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system. *Phys. Rev. Lett.*, 108:130501, Mar 2012.

[49] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

Appendix A

Reduction from integer factorization to order-finding

We briefly present a reduction from factorization to order-finding as described in the original Shor's paper [38].

Algorithm Integer factorization to order-finding

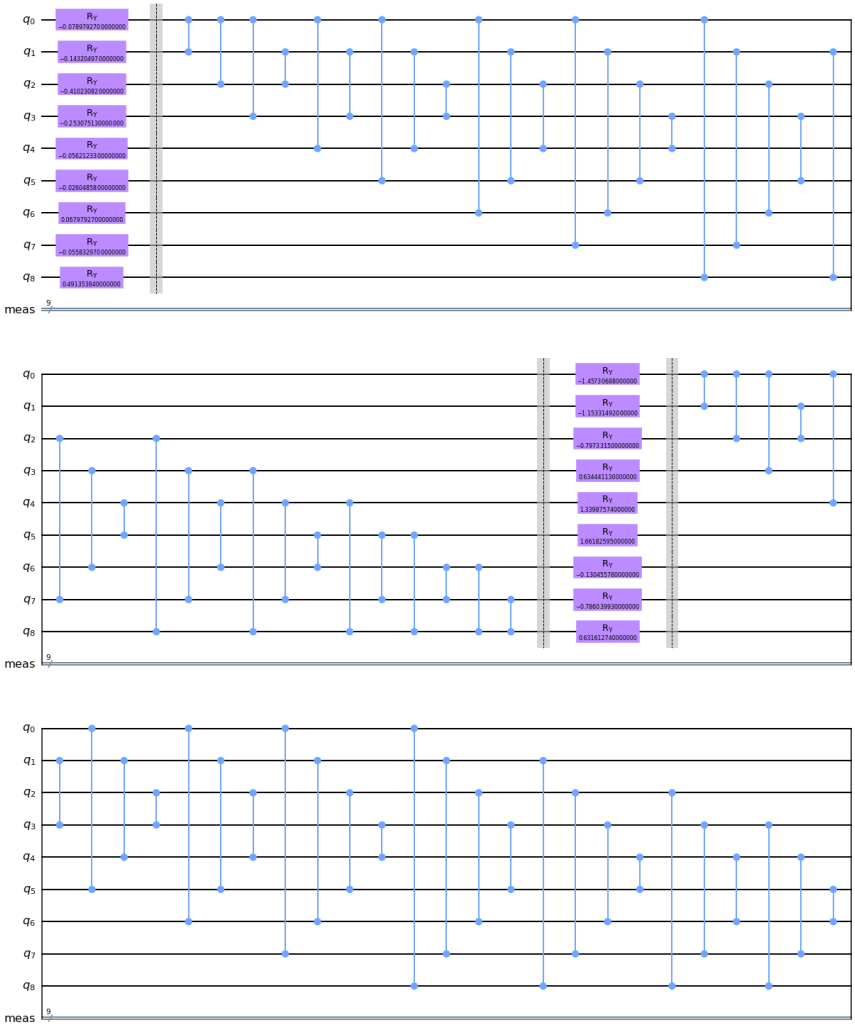
1: **Input:** Semiprime to factor N
2: Choose at random $1 < a < N$ such that $\gcd(a,N) = 1$ (otherwise we found the solution)
3: Use order-finding algorithm to find the period r of $f(x) = a^x \bmod N$
4: If r is odd or $a^{r/2} \equiv -1 \bmod N$, go back to **Line 2**
5: **return** Factors $\gcd(a^{r/2}-1,N)$ and $\gcd(a^{r/2}+1,N)$

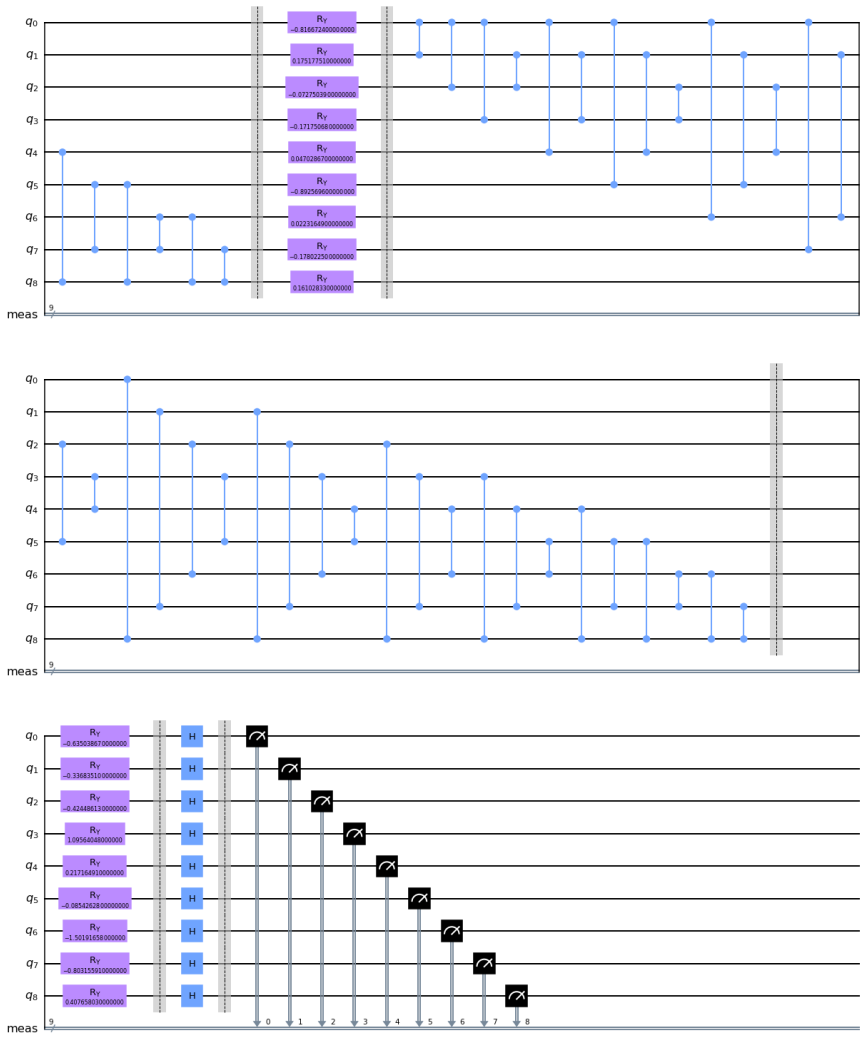
This algorithm works because of some well-known mathematical tricks, which are also the basis for other well-known factoring algorithms. Fermat's method [33] states that given a semi-prime $N = pq$, we can find some a,b s.t. $a^2 - b^2 = (a+b)(a-b) = N$. Then clearly $p = a-b$ and $q = a-b$. Kraitchik upgraded Fermat's method [33]: we can find a,b s.t. $(a+b)(a-b) = kN$. In such case, it is possible that $(a-b)$ divides N and/or $(a+b)$ divides N, in which case $\gcd(a-b,N)$ and $\gcd(a+b,N)$ are non-trivial factors.

Appendix B

Circuit for factoring semiprime 437

In this appendix, we give the full circuit with optimal parameters used to factor semiprime 437. The circuit with optimal parameters for factoring 21 is given in chapter 6 under hardware experiments subsection.




q0
q1
q2
q3
q4
q5
q6
q7
q8
meas
RY
−0.816672400000000
0.175177510000000
−0.072750390000000
−0.171750680000000
0.047028670000000
−0.892569600000000
0.022316490000000
−0.178022500000000
0.161028330000000
RY
−0.635038670000000
−0.336835100000000
−0.424486130000000
1.095604800000000
0.217164910000000
−0.085426280000000
−1.501916580000000
−0.803155910000000
0.407658030000000
H
0
1
2
3
4
5
6
7
8