UNIVERSIDADE FEDERAL DE SANTA CATARINA
BIBLIOTECA UNIVERSITÁRIA

Gilson Trombetta Magro

COMPARISON OF QUADRATIZATION METHODS FOR INTEGER FACTORIZATION VIA ADIABATIC QUANTUM COMPUTING

Florianópolis
2022

Gilson Trombetta Magro

COMPARISON OF QUADRATIZATION METHODS FOR INTEGER FACTORIZATION VIA ADIABATIC QUANTUM COMPUTING

Trabalho de Conclusão de Curso submetido ao Curso de Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Jerusa Marchi, Dra.
Coorientador: Prof. Eduardo Inácio Duzzioni, Dr.

Florianópolis
2022

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.
Arquivo compilado às 23:42h do dia 21 de dezembro de 2022.

Gilson Trombetta Magro
Comparison of quadratization methods for integer factorization via adiabatic quantum computing : / Gilson Trombetta Magro; Orientadora, Profa. Jerusa Marchi, Dra.; Coorientador, Prof. Eduardo Inácio Duzzioni, Dr. – Florianópolis, 23:42, 21 de dezembro de 2022.
135 p.

Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Centro Tecnológico, Curso de Ciência da Computação.

Inclui referências

1. Fatoração de inteiros. 2. Computação quântica adiabática. 3. Quadratização de Hamiltonianos. 4. Funções pseudo-booleanas. I. Profa. Jerusa Marchi, Dra. II. Prof. Eduardo Inácio Duzzioni, Dr. III. Curso de Ciência da Computação IV. Comparison of quadratization methods for integer factorization via adiabatic quantum computing

CDU 02:141:005.7

Gilson Trombetta Magro

COMPARISON OF QUADRATIZATION METHODS FOR INTEGER FACTORIZATION VIA ADIABATIC QUANTUM COMPUTING

Can you reach me? No, you can't (Aha).
– Lorde

Resumo

A computação quântica adiabática (CQA) tem sido estudada como uma alternativa ao modelo de computação quântica baseado em circuitos, especialmente tratando-se de problemas de otimização. Estudos recentes mostraram a aplicabilidade da CQA no contexto da fatoração de inteiros. Este trabalho propõe uma fórmula genérica para um operador Hamiltoniano, que codifica a solução para o problema da fatoração de inteiros. Esta fórmula inclui simplificações booleanas, seguidas da quadratização do Hamiltoniano através de dois métodos diferentes. Este trabalho também apresenta uma comparação entre estes dois métodos de quadratização, considerando métricas como o número de variáveis auxiliares necessário e o intervalo de coefiecientes dos Hamiltonianos resultantes.

Palavras-chaves: Fatoração de inteiros. Computação quântica adiabática. Quadratização de Hamiltonianos. Funções pseudo-booleanas.

Abstract

Adiabatic quantum computing (AQC) has been studied as an alternative for the circuit-based quantum computing model, specially regarding optimization problems. Recent studies have shown the applicability of AQC in the context of integer factorization. We have provided a general formula for a Hamiltonian operator, which encodes the solution to the integer factorization problem. This formula includes Boolean simplifications, followed by a Hamiltonian quadratization via two different methods. We also present a comparison between these two quadratization methods, considering metrics such as the number of auxiliary variables required and the range of coefficients in the resulting Hamiltonians.

Keywords: Integer factorization. Adiabatic quantum computing. Hamiltonian quadratization. Pseudo-Boolean functions.

List of Figures

List of Tables

Contents

Introduction

The integer factorization problem poses the question: given a positive integer N, what are its prime factors? At first, this might seem like an easy question to answer. Naively, one may simply attempt to check every number between 1 and N, and discover which of them divides N perfectly. In fact, this is fine for small numbers, since we don't have to check many values before finding a prime factor for N.

The problem arises when N is a very large number, with hundreds, if not thousands of bits. It is easy to see that, as the number of bits n increases[1], the search space grows at a rate of about 2^n. Thus, the naive approach of checking every number becomes rapidly intractable.

On the other hand, if we take a less naive path, one might consider using the General Number Field Sieve (GNFS) algorithm, proposed by Lenstra et al. (1993), which is the best known classical algorithm for factoring integers. Still, this algorithm only provides a sub-exponential runtime, expressed in the L-notation as follows:

$$L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$$

Surprisingly, though, and despite its importance, the exact

[1] Throughout this study, we will always refer to N as the number being factored, and n as the number of bits of N. Notice that $n \approx \log_2 N$.

complexity of the integer factorization problem remains unknown (KNUTH, 1997). It is therefore possible – although unlikely – that an efficient classical algorithm, capable of factoring integers in polynomial time, is still waiting to be discovered.

For now, though, since no one has ever found an efficient classical algorithm for factorization, it is believed that finding the factors of a number is a hard problem for classical computers to solve, in general. It also turns out that this assumed difficulty on trying to factor large numbers can be taken advantage of by cryptography schemes, such as the RSA (Rivest-Shamir-Adleman) cryptosystem. The RSA cryptosystem is a public-key cryptography scheme widely used around the world, for secure data transmission and digital signatures (NIELSEN; CHUANG, 2010, p. 11).

The RSA scheme involves a pair of public and private keys. These keys are generated from a large number N, which is the product of two large primes p and q, of about the same size. The security provided by RSA comes from the fact that it is computationally very difficult to derive a private key from someone's public key, unless one knows the factorization of $N = pq$. Thus, the advent of a new, more powerful factoring algorithm is of great interest, because it poses a threat to such cryptosystems (STALLINGS, 2013, p. 272-275).

Usually, minor advances in factorization can be overcome by RSA, by simply choosing larger key-sizes (i.e., choosing a larger N). Since 2015, the *National Institute of Standards and Technology* (NIST) recommends key-sizes of at least 2048 bits for RSA, which are considered secure nowadays (BARKER; DANG, 2015).

Currently, the world record for largest RSA number ever factored is known as RSA-250, which has 250 decimal digits (or 829 bits). It was factored in February 2020 using the GNFS algorithm, and it took researchers almost 2700 CPU core-years[2] of processing in a classical computer (BOUDOT et al., 2020).

Unfortunately, as the current size of transistors approaches a physical limit, and quantum effects start interfering with the behavior of conventional electronic circuits, the growth rate of classical

[2] Using the 2.1GHz Intel Xeon Gold 6130 CPU as a reference.

computing power predicted by Moore's Law[3] has decreased. Therefore, it seems unlikely for the near future that RSA schemes should be threatened solely by classical computers.

In that sense, quantum computing is being studied as a model of computation alternative to classical computing. It is believed to be impossible for a classical computer to efficiently simulate a quantum computer. Thus, the quantum paradigm seems to offer a welcome advantage when compared to classical computing.

In fact, there are certain quantum algorithms whose efficiency overcomes that of any known classical algorithm. Perhaps the most famous example of this is the algorithm proposed by Shor (1994), which is capable of factoring a number N in a polynomial number of steps[4], using a (circuit-based) quantum computer. Unfortunately, the current technology of quantum circuits does not allow for any meaningful, practical implementation of Shor's algorithm, because today's noisy intermediate-scale quantum (NISQ) computers simply do not offer a significant amount of error-corrected qubits.

The current record for largest number ever factored using Shor's algorithm, in an actual quantum computer, was achieved in 2012. The number factored in that occasion was $N = 21$, which is obviously very small (MARTÍN-LÓPEZ et al., 2012).

Nonetheless, this result goes to show that efficient factorization through quantum computing may one day be a reality. This is a relevant topic, because, as mentioned before, being able to efficiently factor large integers makes it possible to efficiently break certain types of encryption. Hence, it is speculated that the dawn of a powerful quantum computer could lead to Internet security issues in the future (STALLINGS, 2013).

Meanwhile, as the notion of circuit-based quantum computers grows in popularity – so much as to have become the standard model of quantum computation –, other approaches to quantum computing are also being developed. An example of that is the idea

[3] Moore's Law, formulated in 1965 by Gordon Moore, stated that the computational power of classical computers would double regularly every two years. According to Nielsen and Chuang (2010), this prediction remained approximately correct from the 1960s until the early 2000s.

[4] Regarding the number of bits n of the input number.

of using adiabatic evolution as the means for achieving quantum computation. This paradigm is commonly referred to as adiabatic quantum computing, or AQC for short.

Initially proposed by Farhi et al. (2000), in the context of quantum computing via adiabatic evolution, a given problem is encoded in the construction of a quantum operator, called the Hamiltonian operator. The quantum system is then evolved adiabatically – that is, slowly – to match the ground state of said Hamiltonian, which in turn yields the solution to the problem.

Several examples of integer factorization achieved using AQC – and also using quantum annealing, which is a similar technique – can be found in recent literature. For instance, Peng et al. (2008) showed the factorization of the number 21 via adiabatic quantum computing. Some time later, Xu et al. (2012) successfully factored the number 143 in a quantum device. In 2014, however, Dattani and Bryans (2014) explained that the same Hamiltonian employed by the previous paper is sufficient to factor larger numbers, such as 3599, 11663, and 56153. They also demonstrate the factorization of the number 175, which is the product of three primes (i.e., the product of $5 \times 5 \times 7$).

Although targeted with some criticism[5] for only considering hand-picked, easy instances of the integer factorization problem, these studies present promising results for factorization through adiabatic evolution. Additionally, most of the proposed Hamiltonians seem to require some type of quadratization method, that transforms 3- and 4-body interactions into 2-body interactions, due to physical limitations in current quantum devices.

Consequently, different quadratization methods for pseudo-Boolean functions[6] have been proposed in recent years[7]. Following some new developments[8], it is clear that different quadratization methods can affect the adiabatic computation in different ways –

[5] See Mosca and Verschoor (2019).

[6] Boros and Hammer (2002) define pseudo-Boolean functions as mappings from strings of binary digits to real (or integer) values. They are usually denoted in the form $f : \mathbb{B}^n \to \mathbb{R}$.

[7] See Freedman and Drineas (2005); Ishikawa (2011); Dattani and Chau (2019).

[8] See Dattani (2019).

these include the number of qubits needed and the range of coupling strengths between those qubits.

For these reasons, the present study aims at exploring the effects that different quadratization methods have on the Hamiltonian for integer factorization. We will specifically address the methods proposed by Ishikawa (2011) and Dattani and Chau (2019).

OBJECTIVE

The present work aims at studying the complexity of integer factorization, using the adiabatic quantum computing model as a background. The main goal is to study how different quadratization methods affect the Hamiltonian for the integer factorization problem. Below, we also provide a list of specific goals and results this study hopes to attain:

1. Study different ways to encode the integer factorization problem into a Hamiltonian operator.

2. Present a general formula for the integer factorization Hamiltonian.

3. Implement the quadratization methods proposed by Ishikawa (2011) and Dattani and Chau (2019) over the integer factorization Hamiltonian, and explore their effects on the final Hamiltonian.

4. Provide a comparison between the quadratization methods proposed by Ishikawa (2011) and Dattani and Chau (2019), considering the following set of metrics:

    a) The number of auxiliary variables added by each method.
    b) The number of resulting quadratic monomials in the final Hamiltonian.
    c) The estimated connectivity between the qubits.
    d) The range of coefficients in the final Hamiltonian.

WORK OUTLINE

This document is organized as follows: Chapter 1 characterizes the foundations of adiabatic quantum computing. It also dives into some concepts from quantum mechanics, including the very important adiabatic theorem. Chapter 2 focuses on the actual integer factorization problem, dealing with its precise definition and how it can be encoded into the Hamiltonian operator.

Moreover, Chapter 3 presents a comparison between two Hamiltonian simplification (or rather quadratization) methods, including the calculation of different metrics that help evaluate each methods' cost and effectiveness. Finally, the last chapter summarizes what has been achieved during this study, and discusses potential future works.

We also provide two appendices. Appendix A explains the main concepts of quantum information and quantum computation, all the way from quantum bits to quantum circuits and gates. Meanwhile, Appendix B provides a proof for the equality of two Hamiltonian expressions, which are discussed in detail in Chapter 2.

At the end of this work, we also provide an annex comprised of an article summarizing the present study's achievements and main results.

Chapter 1

Adiabatic Quantum Computing

The adiabatic quantum computing (AQC) model is a fairly recently proposed quantum computational paradigm. It distinguishes itself quite a lot from the standard circuit-based quantum model, because it is not based around sequences of unitary quantum gates. Rather, in AQC the solution to a problem is encoded in the state of a Hermitian operator, known as the Hamiltonian operator – or simply Hamiltonian.

The idea behind AQC is to slowly evolve a quantum system from an initial Hamiltonian, whose ground state is known and easy to prepare, into another Hamiltonian, whose ground state is initially unknown, but which encodes the solution to a given problem. In theory, the system is guaranteed to converge to the ground state of the final Hamiltonian, as long as the conditions imposed by the adiabatic theorem are met (ALBASH; LIDAR, 2018).

This chapter aims to explain the general idea behind adiabatic quantum computing, including the adiabatic theorem which makes AQC possible. We begin by establishing some concepts borrowed from quantum mechanics, which are necessary to properly understand this model.

1.1 QUANTUM MECHANICS

In physics, an observable is any physical quantity that can be measured, such as the position or momentum of a body, for example. In classical physics, these observables are described by real-valued functions. In quantum systems, however, they are described by Hermitian operators, as per Nielsen and Chuang (2010, Postulate 3, p. 85) and Griffiths (2004, p. 97).

A Hermitian operator is a matrix H that obeys the relation $H = H^\dagger$, meaning it is self-adjoint. In particular, the result of measuring a quantum observable is always an eigenvalue of the associated Hermitian operator.

1.1.1 The Hamiltonian Operator

The Hermitian operator associated with the *total energy* of a quantum system receives a special name: it is called the Hamiltonian operator.

Definition 1.1.1 (Hamiltonian Operator)**.** A Hamiltonian operator is a hermitian operator associated with the observable that represents the total energy of a quantum system.

Moreover, let H be the Hamiltonian of a particular quantum system. Let $\{\lambda_0, \lambda_1, ..., \lambda_n\}$ be the set of eigenvalues of H, which corresponds to the energy spectrum of the system. Without loss of generality, assume $\lambda_0 \leq \lambda_1 \leq ... \leq \lambda_n$. Thus, a measurement of the energy level of this system is only allowed to yield energy levels E_i, that are exactly defined by the set of eigenvalues of H, meaning $E_i = \lambda_i$. Additionally, the lowest energy level – or levels, since there can be multiple states with the same energy level – is called the ground state of that system.

Definition 1.1.2 (Ground State)**.** The ground state of a system is the state with the lowest energy level among all possible states. If a Hamiltonian has more than one ground state, it is called a degenerate Hamiltonian (GRIFFITHS, 2004).

Also, each energy level (eigenvalue) has an eigenstate associated to it, meaning that for some state vectors, the following relation holds:

$$H\left|\psi_i\right\rangle = E_i\left|\psi_i\right\rangle$$

1.1.2 Time Evolution of a Quantum System

The time evolution of a closed quantum system can be expressed in at least two different ways. According to Nielsen and Chuang (2010, Postulate 2, p. 81), it is natural to represent the evolution of a closed quantum system from an initial state $\left|\psi_i\right\rangle$ to a final state $\left|\psi_f\right\rangle$ as:

$$\left|\psi_f\right\rangle = U\left|\psi_i\right\rangle,$$

where U is a unitary operator.

In the context of circuit-based quantum computing, U can be interpreted as being associated with a sequence of one or more quantum gates. This interpretation directly corresponds to the idea of quantum circuits, in which the computation occurs in somewhat discrete steps.

Another way to define this evolution is to describe it in terms of a Hermitian operator. This can be done through the time-dependent Schrödinger equation (NIELSEN; CHUANG, 2010, Postulate 2', p. 82):

$$i\hbar\frac{d}{dt}\left|\psi(t)\right\rangle = H\left|\psi(t)\right\rangle. \tag{1.1}$$

Here, H is the Hamiltonian operator and $\left|\psi(t)\right\rangle$ is the state of the quantum system, which is a function of time. Note, however, that we can write the Hamiltonian as a function of time as well, thus arriving at Equation 1.2:

$$i\hbar\frac{d}{dt}\left|\psi(t)\right\rangle = H(t)\left|\psi(t)\right\rangle. \tag{1.2}$$

By writing it this way, not only does $H(t)$ describe the system evolution through time, but the Hamiltonian itself also changes with time. This is important, because it allows us to evolve an initial Hamiltonian into a final Hamiltonian, effectively swapping the operators amid a computation. This idea will be discussed in more detail later.

1.1.3 Adiabaticity

The word *adiabatic* (from the Greek *adiabátos*, meaning impassable) can mean different things depending on the field of study. In thermodynamics, for example, it is used to describe a system isolated from any heat transfer. In quantum mechanics, however, it refers to a slowly changing system, which suffers interference from the outside world, but in a very gradual manner.

1.1.3.1 Adiabatic Process

As claimed by Griffiths (2004, p. 368), in quantum mechanics, an adiabatic process is defined by a gradual change of external conditions. The author uses the following analogy: imagine a perfect pendulum, with no friction or air resistance, oscillating in a vertical plane inside a box (see Figure 1.1). If a person grabs hold of the box and shakes it, the pendulum will lose its regular motion, and transition into a chaotic movement regime. On the other hand, however, if the person moves the box slowly and gently, the pendulum will suffer some small disturbance, but will tend to maintain its oscillating regular motion in approximately the same vertical plane as before.

1.1.3.2 The Adiabatic Theorem

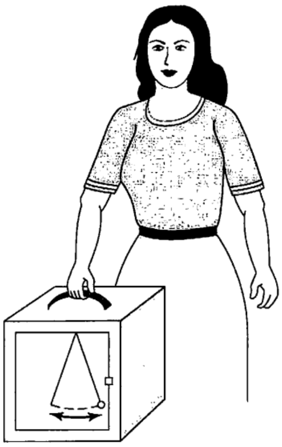
The adiabatic approximation, given in the definition of an adiabatic process, can be cast into a theorem, for the context of quantum processes.

According to Albash and Lidar (2018), it was Born and Fock (1928) who first proposed the modern version of the adiabatic approximation. Later, Kato (1950) is said to have formalized the first quantum adiabatic theorem in more strict, mathematical terms.

Below, we give a less rigorous definition for the adiabatic theorem, which is mainly based in the description given by Albash and Lidar (2018).

Theorem 1.1 (The Adiabatic Theorem). *A system prepared in the ground state $|\psi_0(0)\rangle$ of a time-dependent Hamiltonian $H(t)$ will remain in the instantaneous ground state $|\psi_0(t)\rangle$ of $H(t)$, provided that:*

Figure 1.1 – Example of adiabatic motion.



The picture shows the drawing of a person holding a box with a pendulum inside. It serves as an analogy for the adiabatic process. If the box is handled gently and steadily, the pendulum inside the box will keep swinging with the same amplitude. Source: extracted from (GRIFFITHS, 2004, Figure 10.1)

1. *$H(t)$ varies slowly and gradually enough.*
2. *The energy gap between the ground state energy level E_0 and the first excited state energy level E_1 is sufficiently large.*

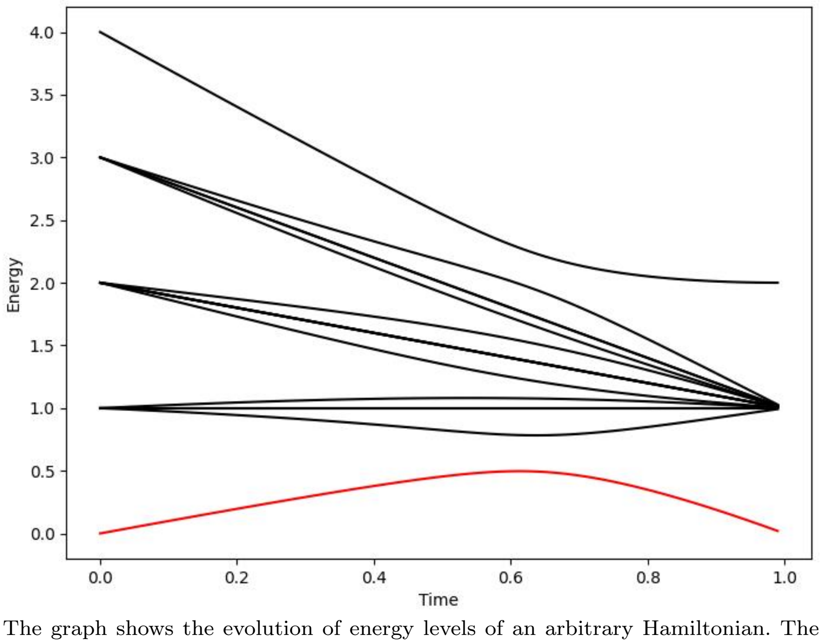
In other words, if a quantum system, prepared in its ground state $|\psi_0(t)\rangle$, suffers small interferences from the outside world, causing its Hamiltonian $H(t)$ to change very gradually, the system should still remain in its ground state. Although the ground state *per se* at a time $t = t_0$ may be different from the ground state at the initial time $t = 0$ (because it is time-dependent and thus varies with time), the system itself will be in a state which is the ground state at said time $t = t_0$.

This notion of the current ground state of an evolving system is usually called the instantaneous ground state, meaning the ground state of $H(t)$ at a fixed time $t = t_0$, or rather the ground state of $H(t = t_0)$.

Note, however, that this is only guaranteed if both conditions from the adiabatic theorem are met. This means that not only should the evolution be adiabatic – slow, gradual –, but the gap between the minimum energy level and all other energy levels should be large enough. This second condition is necessary to ensure that no unwanted state transitions occur during the evolution.

Take Figure 1.2 as an example. This picture shows the evolution of the energy levels of an arbitrary Hamiltonian $H(t)$. The line in red, at the bottom, shows the minimum energy level at each moment – i.e., the energy level of the ground state of the system. If this line were to cross any other energy level, a state transition could occur, and thus the adiabatic theorem would not apply.

Figure 1.2 – Evolution of energy levels.



The graph shows the evolution of energy levels of an arbitrary Hamiltonian. The line in red marks the evolution of the minimum energy level, which corresponds to the energy level of the ground state of the system. Source: developed by the author.

1.2 THE ADIABATIC MODEL OF COMPUTATION

Consider a quantum system such as the one described in Theorem 1.1. One can view the time-dependent Hamiltonian $H(t)$ as an interpolation between an initial Hamiltonian H_i, and a final Hamiltonian H_f. This gives rise to the usual description of adiabatic quantum computing.

Assume the ground state of H_i is known and easy to prepare. On the other hand, imagine we can encode the solution to a problem in the ground state of the H_f[1]. If we are able to evolve the system from H_i to H_f adiabatically, then the adiabatic theorem ensures the system will approach the ground state of H_f, which encodes the solution to our problem. This is the general idea for a computation in the adiabatic quantum computing paradigm.

Definition 1.2.1 (Computation – AQC)**.** A computation, in the context of AQC, is represented by the evolution of a quantum system, in adiabatic conditions, from an initial Hamiltonian H_i – whose ground state is known and easy to prepare – into a final Hamiltonian H_f, whose ground state encodes the solution to a problem.

Additionally, we can also describe the time-dependent Hamiltonian $H(t)$ through an equation, which marks the interpolation between H_i and H_f.

To do so, let T be the total time for a computation. Let $t \in [0, T]$ mark the evolution of time inside the system. It can be convenient to work with a normalized interval, so let us define $s = t/T$ such that $s \in [0, 1]$. Thus, we say the computation starts at time $s = 0$ and ends at time $s = 1$.

Instead of working with $H(t)$, let us consider $H(s)$ without any loss of generality. It is possible to write the time-dependent Hamiltonian as follows:

$$H(s) = f_0(s)H_i + f_1(s)H_f, \tag{1.3}$$

where $f_0, f_1 : [0, 1] \to \mathbb{R}$ are interpolation functions.

[1] Note we do not know the ground state of H_f *a priori*, since knowing it would imply we also know the solution to the problem we are trying to solve.

Note that, ideally, we want a smooth adiabatic interpolation between the initial and final Hamiltonians, such that:

$$H(s=0) = H_i, \text{ and}$$
$$H(s=1) = H_f.$$

Thus, some restrictions must be applied to the functions in Equation 1.3, such that:

$$f_0(s=0) = 1, \quad f_1(s=0) = 0, \text{ and}$$
$$f_0(s=1) = 0, \quad f_1(s=1) = 1.$$

The simplest example of such an interpolation is to choose $f_0(s) = (1-s)$ and $f_1(s) = s$, yielding the Hamiltonian:

$$H(s) = (1-s)H_i + (s)H_f.$$

1.2.0.1 Time Complexity and Gap Dependence

Moreover, note that the adiabatic theorem does not give an actual value for the total time of computation T. This is actually an important issue, because the time it takes to evolve the Hamiltonian will dictate the overall complexity of the problem that is being computed.

In this sense, given what has been established so far, we can define the minimum energy gap, described in Theorem 1.1, as follows:

$$\Delta_{min} = \min_{s\in[0,1]} (E_1(s) - E_0(s)),$$

where E_0 and E_1 are energy levels of the system.

Well, it turns out that, according to Albash and Lidar (2018), a result proposed by Elgart and Hagedorn (2012) sets a rigorous upper bound for T, provided that $H(s)$ satisfies some additional constraints – namely, $H(s)$ must belong to the *Gevrey* class G^α. Here, $H(s)$ is said to belong to G^α if:

$$\frac{dH(s)}{ds} \neq 0, \ \forall s \in [0,1],$$

and there are constants $C, R > 0$ such that, for every $k \geq 1$:

$$\max_{s\in[0,1]} ||H^{(k)}(s)|| \ \leq \ CR^k k^{\alpha k}.$$

This specific result from Elgart and Hagedorn (2012) establishes that T scales with the inverse of the minimum gap squared Δ^2_{min}, except for a logarithmic correction.

Although this upper bound only applies for a certain class of Hamiltonians, it gives an idea as to how the computation time should scale – particularly in comparison to the minimum energy gap. It is clear from this result that the energy gap directly impacts the complexity of the computation.

1.2.0.2 Universality of Adiabatic Quantum Computing

It is famously known that the circuit-based quantum computing model is, in fact, universal for quantum computing – meaning it can efficiently simulate a quantum Turing machine.

Beyond that, it is also known that the circuit-based model and the adiabatic quantum computing model are equivalent – except for a polynomial amount of resources –, meaning they can also simulate each other efficiently. This is enough to establish that AQC is also a universal model of quantum computation (ALBASH; LIDAR, 2018).

Chapter 2

Integer Factorization

This chapter shifts focus to explore the integer factorization problem, and how it can be encoded into the Hamiltonian operator. Firstly, we impose some conditions, in order to properly define the problem of interest. Then, this chapter explores how it is possible to construct a Hamiltonian that encodes the solution to the factorization problem.

Further on, we study how some Boolean simplifications can be applied over said Hamiltonian, in order to reduce the degree of certain monomials that contain repeated variables. This will eventually lead to a general simplified formula for the integer factorization problem. Lastly, we are finally able to explore two methods of quadratization for our Hamiltonian: the first was proposed by Ishikawa (2011), and the second, by Dattani and Chau (2019).

2.1 FORMALIZATION

In order to simplify the process of factoring an integer N, we can impose certain conditions over said number, which can help to better study the problem. Note that verifying certain conditions prior to factoring a number is very standard – even Shor's algorithm has a classical pre-processing step to it –, and helps to focus on the actual difficult part of factorization.

Firstly, the most trivial simplification one can think of is to impose that N be an odd number, since even numbers have a trivial factor 2. Secondly, N must be a composite number – i.e., not a prime. Checking if a number is composite is a bit less trivial, but can be achieved in polynomial time through an algorithm such as the AKS primality test (AGRAWAL; KAYAL; SAXENA, 2004).

Lastly, we may want to impose that N have exactly two factors – let us call them p and q from now on. This is acceptable because it directly corresponds to RSA numbers, which are considered to be the hardest instances of factorization (that is, numbers with exactly two factors of about the same size).

After having established these simplifications, we can properly define the factorization problem we are interested in.

Definition 2.1.1 (Integer Factorization Problem)**.** Let N be a positive, odd, composite integer with exactly two non-trivial factors. Find these two non-trivial factors p and q such that $N = pq$.

2.2 FACTORIZATION VIA ADIABATIC EVOLUTION

If we want to solve factorization via AQC, we must first find a way to construct a Hamiltonian H from an instance of factorization, such that the ground state of H encodes the solution to the problem.

To do so, it can be useful to define a cost function, which attributes more energy to configurations of factors that do not satisfy our instance, and less energy to configurations that give the correct factorization of our number N. Initially, let us consider a very simple cost function $f_N : \mathbb{Z}^2 \to \mathbb{Z}$, for a given integer N:

$$f_N(x, y) = (N - xy)^2. \tag{2.1}$$

Notice that f_N is always non-negative, and that $f_N = 0$ if and only if $N = xy$. For any other values of x, y that do not multiply to give N, $f_N(x, y)$ will be evaluated to a value larger than zero. Thus, intuitively, minimizing f_N for some value of x, y is enough to solve the problem of factoring N.

Suppose we want to write x and y in their binary forms. Although, in general, we do not know the length – in binary digits – of the factors of N, let us assume for a moment that we do. Let us call n_x the length of bits of x. Similarly, let us call n_y the number of bits in y. Then, if we let x_i be the i-th bit in x and so on, we can write both factors in their binary form as such:

$$x = |x_{n_x}\ x_{n_x-1}\ ...\ x_2\ x_1\ x_0|_{(2)},$$
$$y = |y_{n_y}\ y_{n_y-1}\ ...\ y_2\ y_1\ y_0|_{(2)}.$$

Since we assume N is an odd number, we know x and y must also be odd. Thus, we can set the least significant bit of x and y to be $x_0 = y_0 = 1$.

$$x = |x_{n_x}\ x_{n_x-1}\ ...\ x_2\ x_1\ 1|_{(2)},$$
$$y = |y_{n_y}\ y_{n_y-1}\ ...\ y_2\ y_1\ 1|_{(2)}.$$

Additionally, suppose we write x as a binary expansion, such that each Boolean variable x_i with coefficient 2^i represents the i-th bit of x. Suppose we do the same to y, but with a different index k:

$$x = \sum_{i=1}^{n_x} 2^i x_i + 1, \quad \text{and} \quad y = \sum_{k=1}^{n_y} 2^k y_k + 1.$$

Then, we can rewrite f_N as a pseudo-Boolean[1] function $F_N : \mathbb{B}^{n_x+n_y} \to \mathbb{Z}$, which will describe our Hamiltonian H_N, such that:

$$F_N(x_1, ..., x_{n_x}, y_1, ..., y_{n_y}) = \left[N - \left(\sum_{i=1}^{n_x} 2^i x_i + 1\right)\left(\sum_{k=1}^{n_y} 2^k y_k + 1\right)\right]^2. \tag{2.2}$$

From Equation 2.2 it is easy to construct a Hamiltonian: we just have to replace each Boolean variable x_i, y_k with the appropriate quantum operator $\hat{x_i}, \hat{y_k}$. A very natural choice for these operators

[1] Pseudo-Boolean functions are functions of the form $f : \mathbb{B}^n \to \mathbb{R}$, that is, functions defined in the Boolean domain, with real – or integer – codomain. For the purposes of this study, we will only deal with pseudo-Boolean functions with integer codomain $\mathbb{Z}$.

is given below:

$$\hat{x_i} = \frac{I - \sigma_i^z}{2}, \quad \text{and} \quad \hat{y_k} = \frac{I - \sigma_k^z}{2},$$

where σ^z is the Z Pauli matrix and I is simply the identity matrix.

Finally, we arrive at Equation 2.3, which describes the Hamiltonian for the problem, such as the one presented by Hegade et al. (2021).

$$H_N = \left[NI - \left(\sum_{i=1}^{n_x} 2^i \hat{x_i} + I \right) \left(\sum_{k=1}^{n_y} 2^k \hat{y_k} + I \right) \right]^2. \tag{2.3}$$

2.2.1 Bounds on the Length of Factors

Note that, to construct the Hamiltonian for Equation 2.3 we assumed the lengths of x, y in bits were n_x and n_y respectively. However, we do not know these exact values before factoring N.

Thankfully, though, there are certain bounds we can use for the lengths of x and y. For example, Hegade et al. (2021) mentions a result from Peng et al. (2008) that sets an upper bound on the values of n_x and n_y. To achieve that, Peng et al. (2008) imposes, without loss of generality, the following conditions:

1. $x \leq y$,
2. and $3 \leq x \leq \sqrt{N}$,
3. and $\sqrt{N} \leq y \leq N/3$.

Then, provided these conditions are met, the authors define the length bounds as follows[2]:

$$n_x = \lceil \log_2 \lfloor \sqrt{N} \rfloor_{odd} \rceil - 1, \quad \text{and} \quad n_y = \lceil \log_2 \lfloor \frac{N}{3} \rfloor \rceil - 1. \tag{2.4}$$

2.3 GENERAL SIMPLIFIED FORMULA

When expanded, the Hamiltonian presented in Equation 2.2 looks like a pseudo-Boolean polynomial of degree four. However,

[2] Here, $\lfloor \cdot \rfloor_{odd}$ means the largest odd integer not larger than $(\cdot)$.

some of its terms can be further simplified. Let us look at an example for when $N = 21$:

$$\begin{aligned} H_{21} = {} & 16x_1^2y_1^2 + 64x_1^2y_1y_2 + 16x_1^2y_1 + 64x_1^2y_2^2 + 32x_1^2y_2 + 4x_1^2 \\ & + 16x_1y_1^2 + 64x_1y_1y_2 - 152x_1y_1 + 64x_1y_2^2 - 304x_1y_2 \\ & - 80x_1 + 4y_1^2 + 16y_1y_2 - 80y_1 + 16y_2^2 - 160y_2 + 400. \end{aligned}$$

We can see this Hamiltonian involves only three Boolean variables: x_1, y_1 and y_2. Yet, several of its terms involve repeated variables (e.g., the first term $16x_1^2y_1^2$ involves x_1 twice and y_1 twice), which leads to some squared Boolean variables in our formula. This needn't be the case, and we can simplify this Hamiltonian by getting rid of these squared variables.

Of course, this simplification is only possible because we know all of our variables are Boolean variables, which means they can only take the discrete values 0 and 1. Therefore, it is easy to see that any Boolean variable $b \in \mathbb{B}$ remains unchanged under exponentiation, meaning $b^m = b$, because $0^m = 0$ and $1^m = 1$, at least as long as the exponent m is an integer. Knowing this, we can go back to our Hamiltonian H_{21} and replace every x_i^2 with just x_i. We do the same for every y_k variable:

$$\begin{aligned} H_{21} = {} & 16x_1y_1 + 64x_1y_1y_2 + 16x_1y_1 + 64x_1y_2 + 32x_1y_2 + 4x_1 \\ & + 16x_1y_1 + 64x_1y_1y_2 - 152x_1y_1 + 64x_1y_2 - 304x_1y_2 \\ & - 80x_1 + 4y_1 + 16y_1y_2 - 80y_1 + 16y_2 - 160y_2 + 400 \\ H_{21} = {} & 128x_1y_1y_2 - 104x_1y_1 - 144x_1y_2 - 76x_1 + 16y_1y_2 \\ & - 76y_1 - 144y_2 + 400. \end{aligned}$$

Naturally, we would like to be able to apply this simplification to the general case. To do so, we must work on expanding the function from Equation 2.2. Also, to simplify notation, we will from now on refer to the function F_N as simply H_N, which is our Hamiltonian.

Firstly, let us expand the product $x \times y$ (in their binary expansions) from Equation 2.2, which will take us to the following:

$$\begin{aligned}H_N &= \left[N - \left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k + \sum_{i=1}^{n_x} 2^i x_i + \sum_{k=1}^{n_y} 2^k y_k + 1\right)\right]^2 \\ &= \left[(N-1) - \left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k\right) - \left(\sum_{i=1}^{n_x} 2^i x_i\right) - \left(\sum_{k=1}^{n_y} 2^k y_k\right)\right]^2 .\end{aligned}$$

Secondly, we must expand the outermost square that surrounds the whole formula. Doing so will yield a larger, messier formula which is shown next. In addition to that, we change some indexes from i to j (regarding x variables) and from k to l (regarding y variables) to avoid confusion. This has been done to preserve the integrity of the products between different summations.

$$\begin{aligned}H_N = &\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k\right)^2 + \left(\sum_{i=1}^{n_x} 2^i x_i\right)^2 + \left(\sum_{k=1}^{n_y} 2^k y_k\right)^2 \\ &+ 2\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j \sum_{k=1}^{n_y} 2^k y_k\right) - 2(N-1)\left(\sum_{i=1}^{n_x} 2^i x_i\right) \\ &+ 2\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) - 2(N-1)\left(\sum_{k=1}^{n_y} 2^k y_k\right) \\ &- 2(N-2)\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k\right) + (N-1)^2.\end{aligned} \tag{2.5}$$

Further on, the first three terms in Equation 2.5 are squared, so we should resolve them too. Once again, we change some indexes to avoid confusion:

$$\begin{aligned} H_N = &\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j \sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) \\ &+ \left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j\right) + \left(\sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) \\ &+ 2\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j \sum_{k=1}^{n_y} 2^k y_k\right) - 2(N-1)\left(\sum_{i=1}^{n_x} 2^i x_i\right) \\ &+ 2\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) - 2(N-1)\left(\sum_{k=1}^{n_y} 2^k y_k\right) \\ &- 2(N-2)\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{k=1}^{n_y} 2^k y_k\right) + (N-1)^2. \end{aligned} \tag{2.6}$$

Finally, we can move the coefficients that are outside the parenthesis to inside the summations, so they can be written together with the coefficients that are already there.

$$\begin{aligned} H_N = &\left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j \sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) \\ &+ \left(\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j\right) + \left(\sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) \\ &+ \left(\sum_{i=1}^{n_x} 2^{i+1} x_i \sum_{j=1}^{n_x} 2^j x_j \sum_{k=1}^{n_y} 2^k y_k\right) - \left(\sum_{i=1}^{n_x} 2^{i+1}(N-1) x_i\right) \\ &+ \left(\sum_{i=1}^{n_x} 2^{i+1} x_i \sum_{k=1}^{n_y} 2^k y_k \sum_{l=1}^{n_y} 2^l y_l\right) - \left(\sum_{k=1}^{n_y} 2^{k+1}(N-1) y_k\right) \\ &- \left(\sum_{i=1}^{n_x} 2^{i+1}(N-2) x_i \sum_{k=1}^{n_y} 2^k y_k\right) + (N-1)^2. \end{aligned} \tag{2.7}$$

Now, we must deal with the products between summations with different indexes. It is not hard to see that some of them will produce terms with repeated variables – which is what we are trying to get rid of, in order to simplify the final expression. Take, for instance, the second term on Equation 2.7, which is the product of two summations over x:

$$\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j. \tag{2.8}$$

As we can see, the first summation is indexed with i and the second, with j. For that reason, we should consider three different cases:

$$\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j = \begin{cases} \sum\limits_{1 \le i < j \le n_x} 2^{i+j} x_i x_j, & i < j \\ \sum\limits_{1 \le j < i \le n_x} 2^{i+j} x_i x_j, & i > j \\ \sum\limits_{1 \le i \le n_x} 2^{2i} x_i^2, & i = j. \end{cases} \tag{2.9}$$

Note however, that if we replace i for j and vice-versa in the second case, we can write $i < j$ for both the first and second cases. Because they are symmetric in relation to each other (both i and j vary from 1 through n_x), we can transform the first and second cases into one single expression. Moreover, as we've seen before, any Boolean variable squared equals itself. Thus, we can reduce the x_i^2 in the third case of Equation 2.9 into simply x_i.

$$\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j = \begin{cases} \sum\limits_{1 \le i < j \le n_x} 2^{i+j+1} x_i x_j, & i < j \\ \sum\limits_{1 \le i \le n_x} 2^{2i} x_i, & i = j. \end{cases} \tag{2.10}$$

Finally, we arrive at the simplified Equation 2.11 for that single term shown in Equation 2.8.

$$\sum_{i=1}^{n_x} 2^i x_i \sum_{j=1}^{n_x} 2^j x_j = \left(\sum_{1 \le i < j \le n_x} 2^{i+j+1} x_i x_j + \sum_{1 \le i \le n_x} 2^{2i} x_i \right). \tag{2.11}$$

Naturally, we can repeat this process for all the terms in the Hamiltonian from Equation 2.7. This will yield the longer – yet simpler – Hamiltonian presented below.

$$
\begin{aligned}
H_N = &\left(\sum_{\substack{1\le i<j\le n_x \\ 1\le k<l\le n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l + \sum_{\substack{1\le i<j\le n_x \\ 1\le k\le n_y}} 2^{i+j+2k+1} x_i x_j y_k \right. \\
&\left. + \sum_{\substack{1\le i\le n_x \\ 1\le k<l\le n_y}} 2^{2i+k+l+1} x_i y_k y_l + \sum_{\substack{1\le i\le n_x \\ 1\le k\le n_y}} 2^{2i+2k} x_i y_k \right) \\
&+ \left(\sum_{1\le i<j\le n_x} 2^{i+j+1} x_i x_j + \sum_{1\le i\le n_x} 2^{2i} x_i \right) \\
&+ \left(\sum_{1\le k<l\le n_y} 2^{k+l+1} y_k y_l + \sum_{1\le k\le n_y} 2^{2k} y_k \right) \\
&+ \left(\sum_{\substack{1\le i<j\le n_x \\ 1\le k\le n_y}} 2^{i+j+k+2} x_i x_j y_k + \sum_{\substack{1\le i\le n_x \\ 1\le k\le n_y}} 2^{2i+k+1} x_i y_k \right) \\
&- \left(\sum_{1\le i\le n_x} 2^{i+1} (N-1) x_i \right) \\
&+ \left(\sum_{\substack{1\le i\le n_x \\ 1\le k<l\le n_y}} 2^{i+k+l+2} x_i y_k y_l + \sum_{\substack{1\le i\le n_x \\ 1\le k\le n_y}} 2^{i+2k+1} x_i y_k \right) \\
&- \left(\sum_{1\le k\le n_y} 2^{k+1} (N-1) y_k \right) \\
&- \left(\sum_{\substack{1\le i\le n_x \\ 1\le k\le n_y}} 2^{i+k+1} (N-2) x_i y_k \right) + (N-1)^2.
\end{aligned}
\tag{2.12}
$$

Now, if we rearrange some terms and join them with terms involving the same variables, we arrive at the following expression, which we will refer to as our general simplified formula:

$$\begin{aligned}H_N = &\sum_{\substack{1\le i<j\le n_x\\1\le k<l\le n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l\\
&+\sum_{\substack{1\le i<j\le n_x\\1\le k\le n_y}} (2^{i+j+2k+1} + 2^{i+j+k+2}) x_i x_j y_k\\
&+\sum_{\substack{1\le i\le n_x\\1\le k<l\le n_y}} (2^{2i+k+l+1} + 2^{i+k+l+2}) x_i y_k y_l\\
&+\sum_{\substack{1\le i\le n_x\\1\le k\le n_y}} [2^{2i+2k} + 2^{2i+k+1} + 2^{i+2k+1} - 2^{i+k+1}(N-2)] x_i y_k\\
&+\sum_{1\le i<j\le n_x} 2^{i+j+1} x_i x_j + \sum_{1\le k<l\le n_y} 2^{k+l+1} y_k y_l\\
&+\sum_{1\le i\le n_x} [2^{2i} - 2^{i+1}(N-1)] x_i + \sum_{1\le k\le n_y} [2^{2k} - 2^{k+1}(N-1)] y_k\\
&+ (N-1)^2.\end{aligned} \tag{2.13}$$

2.4 HAMILTONIAN QUADRATIZATION

The process of quadratizing a Hamiltonian – or any pseudo-Boolean function, for that matter – consists of transforming all its terms of degree $d > 2$ into quadratic terms (i.e., $d = 2$). Usually, this is done at the expense of adding auxiliary binary variables to the initial expression.

Normally, one would want to do this quadratization process in order to reduce the complexity of implementing a Hamiltonian in a real physical system. This is because two-body physical interactions occur more naturally than interactions involving many bodies (DATTANI, 2019). Thus, it is generally easier to implement Hamiltonians with at most two-body interactions, than it is to implement those with cubic and quartic terms, for example.

For this reason, in this section we will explore two general methods for quadratization. The first one was proposed by Ishikawa (2011) and the second, by Dattani and Chau (2019). We will look at them both in the context of the Hamiltonian for factorization that we have defined in the previous section, in Equation 2.13.

2.4.1 Ishikawa's Quadratization Method

Ishikawa (2011) presents a method for quadratizing a general monomial of degree d, which is an extension of another quadratization method proposed by Freedman and Drineas (2005). The author proposes this method in the context of Markov Random Fields (MRF) minimization applied to computer vision. Nonetheless, it can also be employed to quadratize general pseudo-Boolean functions, such as the factorization Hamiltonian from Equation 2.13. But first, we must look at the method itself, which is explained below.

Consider a d-degree monomial of the form $\alpha x_1 x_2 ... x_d$, such that x_i are Boolean variables and α is a real coefficient. Let us also define two symmetric polynomials using these variables:

$$S_1 = \sum_{i=1}^{d} x_i, \quad \text{and} \quad S_2 = \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} x_i x_j = \frac{S_1(S_1 - 1)}{2}.$$

Now, according to Ishikawa (2011), we must consider two different cases for α, should we want to quadratize this monomial. They are given below:

Case 1: $\alpha < 0$. When α is negative, the quadratization is done via a simple substitution given by the equation:

$$\alpha x_1 \cdots x_d = \alpha \min_{b \in \mathbb{B}} b(S_1 - d + 1), \tag{2.14}$$

where b is a newly introduced auxiliary binary variable. This equation comes directly from the method initially proposed by Freedman and Drineas (2005).

Case 2: $\alpha > 0$. When α is positive, on the other hand, the substitution is a bit more convoluted:

$$\alpha x_1 \cdots x_d = \alpha \min_{b_1 \cdots b_{n_d} \in \mathbb{B}} \sum_{i=1}^{n_d} b_i(c_{i,d}(-S_1 + 2i) - 1) + \alpha S_2, \tag{2.15}$$

where:

$$n_d = \left\lfloor \frac{d-1}{2} \right\rfloor, \qquad c_{i,d} = \begin{cases} 1, & \text{if } d \text{ is odd and } i = n_d \\ 2, & \text{otherwise.} \end{cases}$$

As can be seen above, both cases are based around a minimization expression, which involves the function *min* parametrized with one or more auxiliary variables b_i. In both cases, the idea is that the monomial on the left side of the equation equals the expression on the right, as long as b_i takes on values which minimize said expression.

This is generally fine in the context of adiabatic quantum computing, because all computations are essentially trying to find the global minimum of a cost function, which is interpreted as the Hamiltonian's ground state – which in turn encodes the solution to a problem. Naturally, this is also the case for our factorization Hamiltonian H_N, whose ground state encodes the factors of N. In that sense, it is perfectly okay to introduce new auxiliary variables to our expression, because they will be minimized together with the variables that are already part of our Hamiltonian (namely our x_i and y_k variables). For this reason, we can essentially ignore the extra *min* function added by the Ishikawa substitution, since our Hamiltonian will be minimized anyway, throughout the adiabatic evolution.

Furthermore, it is interesting to note that in the first case ($\alpha < 0$), only one extra variable per monomial is needed for the quadratization. But in the second case ($\alpha > 0$), n_d extra variables are needed, in which the exact number of variables depends on the degree d of the monomial we wish to quadratize.

Let us now look at how this method can be applied to our factorization Hamiltonian. From Equation 2.13, it is easy to see that only the first three summations produce monomials with degree $d = 3$ and $d = 4$. Thus, the Hamiltonian of interest can be rewritten in an abbreviated manner, like this:

$$\begin{aligned} H_N = &\sum_{\substack{1\le i<j\le n_x\\ 1\le k<l\le n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l \\ &+ \sum_{\substack{1\le i<j\le n_x\\ 1\le k\le n_y}} (2^{i+j+2k+1} + 2^{i+j+k+2}) x_i x_j y_k \\ &+ \sum_{\substack{1\le i\le n_x\\ 1\le k<l\le n_y}} (2^{2i+k+l+1} + 2^{i+k+l+2}) x_i y_k y_l \\ &+ (\textit{original terms of degree less than 3}). \end{aligned} \tag{2.16}$$

Moreover, it is also clear that the coefficients of the monomials of interest are all positive, meaning $\alpha > 0$. Therefore, we only care about the second case of Ishikawa's quadratization method, namely Equation 2.15, when applied to the monomials of degree $d = 3$ and $d = 4$. Thus, we can calculate the values for $n_{d=3}$ and $n_{d=4}$:

$$n_{d=3} = \left\lfloor \frac{3-1}{2} \right\rfloor = \left\lfloor \frac{2}{2} \right\rfloor = 1,$$
$$n_{d=4} = \left\lfloor \frac{4-1}{2} \right\rfloor = \left\lfloor \frac{3}{2} \right\rfloor = 1.$$

It turns out that in both cases $n_d = 1$, meaning each monomial needs only one extra variable. Furthermore, we can also calculate the constants $c_{(i=1,d=3)}$ and $c_{(i=1,d=4)}$ which are used in Equation 2.15.

$$c_{(i=1,d=3)} = c_{1,3} = 1, \text{ because } d \text{ is odd and } i = n_{d=3} = 1,$$
$$c_{(i=1,d=4)} = c_{1,4} = 2, \text{ because } d \text{ is even.}$$

Now we have everything we need to rewrite Equation 2.15 for our specific cases of $d = 3$ and $d = 4$:

Case 2.1: $\alpha > 0$ **and** $d = 3$**.**

$$\begin{aligned}
\alpha x_1x_2x_3 &= \alpha \min_{b\in\mathbb{B}} b(c_{1,3}(-S_1+2)-1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(-S_1+2-1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(1-S_1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(1-(x_1+x_2+x_3))+\alpha(x_1x_2+x_1x_3+x_2x_3) \\
&= \alpha \min_{b\in\mathbb{B}} b(1-x_1-x_2-x_3)+\alpha(x_1x_2+x_1x_3+x_2x_3) \\
&= \alpha b(1-x_1-x_2-x_3)+\alpha(x_1x_2+x_1x_3+x_2x_3).
\end{aligned}$$

Case 2.2: $\alpha > 0$ **and** $d = 4$**.**

$$\begin{aligned}
\alpha x_1x_2x_3x_4 &= \alpha \min_{b\in\mathbb{B}} b(c_{1,4}(-S_1+2)-1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(2(-S_1+2)-1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(-2S_1+4-1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(3-2S_1)+\alpha S_2 \\
&= \alpha \min_{b\in\mathbb{B}} b(3-2(x_1+x_2+x_3+x_4)) \\
&\quad +\alpha(x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4) \\
&= \alpha \min_{b\in\mathbb{B}} b(3-2x_1-2x_2-2x_3-2x_4) \\
&\quad +\alpha(x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4) \\
&= \alpha b(3-2x_1-2x_2-2x_3-2x_4) \\
&\quad +\alpha(x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4).
\end{aligned}$$

Finally, based on the equations from cases 2.1 and 2.2, it is possible to fully quadratize the Hamiltonian from Equation 2.16:

$$\begin{aligned}
H_N = &\sum_{\substack{1\leq i<j\leq n_x \\ 1\leq k<l\leq n_y}} 2^{i+j+k+l+2}\beta_{ijkl} \\
&+ \sum_{\substack{1\leq i<j\leq n_x \\ 1\leq k\leq n_y}} (2^{i+j+2k+1} + 2^{i+j+k+2})\beta_{ijk} \\
&+ \sum_{\substack{1\leq i\leq n_x \\ 1\leq k<l\leq n_y}} (2^{2i+k+l+1} + 2^{i+k+l+2})\beta_{ikl} \\
&+ (\textit{original terms of degree less than 3}).
\end{aligned} \tag{2.17}$$

where:

$$\begin{aligned}
\beta_{ijkl} &= [b_{ijkl}(3 - 2x_i - 2x_j - 2y_k - 2y_l) \\
&\qquad + x_ix_j + x_iy_k + x_iy_l + x_jy_k + x_jy_l + y_ky_l] \\
\beta_{ijk} &= [b_{ijk0}(1 - x_i - x_j - y_k) + x_ix_j + x_iy_k + x_jy_k] \\
\beta_{ikl} &= [b_{i0kl}(1 - x_i - y_k - y_l) + x_iy_k + x_iy_l + y_ky_l].
\end{aligned}$$

Note that now the Hamiltonian contains monomials of at most degree $d = 2$, meaning it is entirely composed of at most two-body physical interactions. Also note that we have parametrized each auxiliary variable b as either b_{ijkl}, b_{ijk0} or b_{i0kl} to avoid confusion, but also to emphasize that each of them is a different extra variable. Having said that, finally, the full formula for the factorization Hamiltonian is shown below, now quadratized using the Ishikawa method, and without abbreviations:

$$\begin{aligned}
H_N = &\sum_{\substack{1\le i<j\le n_x\\ 1\le k<l\le n_y}} 2^{i+j+k+l+2}[b_{ijkl}(3-2x_i-2x_j-2y_k-2y_l)\\
&+x_ix_j+x_iy_k+x_iy_l+x_jy_k+x_jy_l+y_ky_l]\\
&+\sum_{\substack{1\le i<j\le n_x\\ 1\le k\le n_y}}(2^{i+j+2k+1}+2^{i+j+k+2})\\
&*[b_{ijk0}(1-x_i-x_j-y_k)+x_ix_j+x_iy_k+x_jy_k]\\
&+\sum_{\substack{1\le i\le n_x\\ 1\le k<l\le n_y}}(2^{2i+k+l+1}+2^{i+k+l+2})\\
&*[b_{i0kl}(1-x_i-y_k-y_l)+x_iy_k+x_iy_l+y_ky_l]\\
&+\sum_{\substack{1\le i\le n_x\\ 1\le k\le n_y}}[2^{2i+2k}+2^{2i+k+1}+2^{i+2k+1}-2^{i+k+1}(N-2)]x_iy_k\\
&+\sum_{1\le i<j\le n_x}2^{i+j+1}x_ix_j+\sum_{1\le k<l\le n_y}2^{k+l+1}y_ky_l\\
&+\sum_{1\le i\le n_x}[2^{2i}-2^{i+1}(N-1)]x_i+\sum_{1\le k\le n_y}[2^{2k}-2^{k+1}(N-1)]y_k\\
&+(N-1)^2.
\end{aligned} \tag{2.18}$$

2.4.2 Dattani-Chau's Quadratization Method

The second quadratization method we are going to explore is the one proposed by Dattani and Chau (2019). The authors of this paper show that any 4-variable pseudo-Boolean function can be perfectly quadratized with just one auxiliary variable. This result is cast into a theorem (Theorem 1), which is then proven by showing explicit quadratizations for various different cases. In each case, the authors consider the following function of Boolean variables $x_i \in \mathbb{B}$ and real-valued coefficients α:

$$\begin{aligned}
f(x_1,x_2,x_3,x_4) = \alpha_{1234}x_1x_2x_3x_4+\alpha_{123}x_1x_2x_3+\alpha_{124}x_1x_2x_4\\
+\alpha_{134}x_1x_3x_4+\alpha_{234}x_2x_3x_4.
\end{aligned} \tag{2.19}$$

From there, Dattani and Chau (2019) provide four lemmas (Lemmas 1-4) that give explicit quadratizations for a few specific configurations of the α coefficients. Then, they extend the applicability of these lemmas by allowing bit-flips over the function from Equation 2.19. This process results in 35 different cases, for which the authors provide explicit quadratizations. In each case, the paper suggests a quadratization procedure, which is always a combination of a substitution (described by one of the Lemmas 1-4), plus a list of variables of $f(x_1, x_2, x_3, x_4)$ that should be bit-flipped.

For the purposes of this study, we will not go into the details of all 35 cases, because our Hamiltonian really only needs the first quadratization case, described by Lemma 1, and does not require any bit-flips. In summary, this first lemma states that, if $\alpha_{1234} \geq 0$ and $\alpha_{ijk} \geq 0$ for all ijk, then Equation 2.19 is perfectly quadratized by the following:

$$\begin{aligned} &\left(3\alpha_{1234} + \sum_{ijk} \alpha_{ijk}\right) b + \sum_{ij} \left(\alpha_{1234} + \sum_{k \notin ij} \alpha_{ijk}\right) x_i x_j \\ &- \sum_{i} \left(2\alpha_{1234} + \sum_{jk;i \notin jk} \alpha_{ijk}\right) b x_i, \end{aligned} \tag{2.20}$$

where, once again, b is an auxiliary binary variable.

Now, this means it is possible to quadratize up to five monomials (one of degree 4 and another four of degree 3) using only one extra variable, which seems better than the method proposed by Ishikawa (2011). But, of course, the factorization Hamiltonian from Equation 2.16 is far from the form of Equation 2.19. Therefore, for it to work, this quadratization must be performed over small groups of 4-variable sub-functions of H_N. Let us consider an example. Take, for instance, the Hamiltonian for $N = 33$.

$$\begin{aligned}H_{33} = {} & 256x_1x_2y_1y_2 + 512x_1x_2y_1y_3 + 128x_1x_2y_1 + 1024x_1x_2y_2y_3 \\ & + 384x_1x_2y_2 + 1280x_1x_2y_3 + 16x_1x_2 + 128x_1y_1y_2 - 200x_1y_1 \\ & + 256x_1y_1y_3 + 512x_1y_2y_3 - 336x_1y_2 - 416x_1y_3 + 384x_2y_1y_2 \\ & + 768x_2y_1y_3 - 336x_2y_1 + 1536x_2y_2y_3 - 480x_2y_2 - 192x_2y_3 \\ & - 240x_2 - 124x_1 + 16y_1y_2 + 32y_1y_3 - 124y_1 + 64y_2y_3 \\ & - 240y_2 - 448y_3 + 1024.\end{aligned}$$

It is possible to rearrange the terms of H_{33} in order to separate quartic and cubic monomials from those of quadratic and linear degree. While doing so, one can also try to divide those monomials of higher degree into groups, that share the same variables:

$$\begin{aligned}H_{33} = {} & f_{1212} + f_{1213} + f_{1223} \\ & + 16x_1x_2 - 200x_1y_1 - 336x_1y_2 - 416x_1y_3 - 336x_2y_1 \\ & - 124x_1 - 480x_2y_2 - 192x_2y_3 - 240x_2 + 16y_1y_2 + 32y_1y_3 \\ & - 124y_1 + 64y_2y_3 - 240y_2 - 448y_3 + 1024.\end{aligned} \tag{2.21}$$

where:

$$\begin{aligned}f_{1212} &= 256x_1x_2y_1y_2 + 128x_1x_2y_1 + 384x_1x_2y_2 + 128x_1y_1y_2 \\ &\quad + 384x_2y_1y_2 \\ f_{1213} &= 512x_1x_2y_1y_3 + 1280x_1x_2y_3 + 256x_1y_1y_3 + 768x_2y_1y_3 \\ f_{1223} &= 1024x_1x_2y_2y_3 + 512x_1y_2y_3 + 1536x_2y_2y_3.\end{aligned}$$

This way, each sub-function f is of the form presented in Equation 2.19. Therefore, they can each be quadratized individually using the Expression 2.20, like such:

$$\begin{aligned}f_{1212} = {} & 768x_1x_2 + 512x_1y_1 + 768x_1y_2 + 768x_2y_1 + 1024x_2y_2 \\ & + 768y_1y_2 + b_1(1792 - 1152x_1 - 1408x_2 - 1152y_1 - 1408y_2) \\ f_{1213} = {} & 1792x_1x_2 + 768x_1y_1 + 2048x_1y_3 + 1280x_2y_1 + 2560x_2y_3 \\ & + 1536y_1y_3 + b_2(3840 - 2560x_1 - 3072x_2 - 2048y_1 - 3328y_3) \\ f_{1223} = {} & 1024x_1x_2 + 1536x_1y_2 + 1536x_1y_3 + 2560x_2y_2 + 2560x_2y_3 \\ & + 3072y_2y_3 + b_3(5120 - 2560x_1 - 3584x_2 - 4096y_2 - 4096y_3).\end{aligned}$$

And then we can substitute these back into H_{33}, so that the whole Hamiltonian is fully quadratized with just 3 auxiliary variables b_1, b_2 and b_3:

$$\begin{aligned}
H_{33} = &-1152b_1x_1 - 1408b_1x_2 - 1152b_1y_1 - 1408b_1y_2 + 1792b_1\\
&- 2560b_2x_1 - 3072b_2x_2 - 2048b_2y_1 - 3328b_2y_3 + 3840b_2\\
&- 2560b_3x_1 - 3584b_3x_2 - 4096b_3y_2 - 4096b_3y_3 + 5120b_3\\
&+ 3600x_1x_2 + 1080x_1y_1 + 1968x_1y_2 + 3168x_1y_3 - 124x_1\\
&+ 1712x_2y_1 + 3104x_2y_2 + 4928x_2y_3 - 240x_2 + 784y_1y_2\\
&+ 1568y_1y_3 - 124y_1 + 3136y_2y_3 - 240y_2 - 448y_3 + 1024.
\end{aligned}$$

Naturally, one would like to generalize this procedure. One way to do this would be to construct the Hamiltonian using Equation 2.13, and then add a pre-processing step devoted to grouping similar monomials and quadratizing each group individually. This would work fine, but would not provide insight as to how the Hamiltonian looks in the general case. For that reason, it is preferable to construct a general formula for the quadratized Hamiltonian.

The goal here is to find a reliable way to group certain monomials together. The challenge, though, is that there are multiple ways one could construct these sub-functions inside the Hamiltonian expression.

Consider, for instance, the term $128x_1x_2y_1$ in the Hamiltonian H_{33} from Equation 2.21. In the previous example, we chose to put this monomial in the sub-function f_{1212}, but it could have just as easily been grouped with the other monomials from the sub-function f_{1213}, since they also share the same variables. For the purposes of this quadratization method, both choices would have worked.

Thus, in order to avoid this ambiguity, one can define a set of grouping rules that will dictate which cubic monomials will be grouped with what quartic monomials. We have chosen a somewhat arbitrary set of grouping rules, which follows a lexicographic ordering:

1. Monomials of the form $x_ix_jy_l$ should be grouped with:

   a) $x_ix_jy_1y_2$, if $l = 1$.

b) $x_i x_j y_1 y_l$, otherwise.

2. Monomials of the form $x_j y_k y_l$ should be grouped with:

   a) $x_1 x_2 y_k y_l$, if $j = 1$.

   b) $x_1 x_j y_k y_l$, otherwise.

From these rules, we have empirically constructed a set of nine different groupings for different intervals, in order to cover all the monomials from the Hamiltonian, but without any overlaps. These groupings are listed in Table 2.1.

Table 2.1 – Nine non-overlapping monomial groupings.

#	**Quartic monomial**	**Cubic monomials**	**Intervals**
1	$x_1x_2y_1y_2$	$x_1x_2y_1$ $x_1x_2y_2$ $x_1y_1y_2$ $x_2y_1y_2$	-
2	$x_1x_2y_1y_l$	$x_1x_2y_l$ $x_1y_1y_l$ $x_2y_1y_l$	$l \geq 3$
3	$x_1x_jy_1y_2$	$x_1x_jy_1$ $x_1x_jy_2$ $x_jy_1y_2$	$j \geq 3$
4	$x_1x_jy_1y_l$	$x_1x_jy_l$ $x_jy_1y_l$	$j \geq 3,\ l \geq 3$
5	$x_1x_2y_ky_l$	$x_1y_ky_l$ $x_2y_ky_l$	$k \geq 2,\ l > k$
6	$x_ix_jy_1y_2$	$x_ix_jy_1$ $x_ix_jy_2$	$i \geq 2,\ j > i$
7	$x_1x_jy_ky_l$	$x_jy_ky_l$	$j \geq 3,\ k \geq 2,\ l > k$
8	$x_ix_jy_1y_l$	$x_ix_jy_l$	$i \geq 2,\ j > i,\ l \geq 3$
9	$x_ix_jy_ky_l$	-	$i \geq 2,\ j > i,\ k \geq 2,\ l > k$

Source: developed by the author.

Notice how each of the cubic monomials in Table 2.1 is grouped with its corresponding quartic monomial, according to the grouping rules we have previously established. From these, it is possible to build nine sub-functions (f_1 through f_9), that will split our Hamiltonian from Equation 2.16 into smaller, non-overlapping parts. These sub-functions are presented below. We note that the coefficients for each term also come directly from Equation 2.16.

$$\begin{aligned}
f_1 &= 2^8 x_1 x_2 y_1 y_2 + 2^7 x_1 x_2 y_1 + 3 * 2^7 x_1 x_2 y_2 + 2^7 x_1 y_1 y_2 \\
&\quad + 3 * 2^7 x_2 y_1 y_2 \\
f_2 &= \sum_{3 \le l \le n_y} 2^{l+6} x_1 x_2 y_1 y_l + 2^{l+5} x_1 y_1 y_l + (2^{l+5} + 2^{2l+4}) x_1 x_2 y_l \\
&\quad + (2^{l+5} + 2^{l+6}) x_2 y_1 y_l \\
f_3 &= \sum_{3 \le j \le n_x} 2^{j+6} x_1 x_j y_1 y_2 + 2^{j+5} x_1 x_j y_1 + (2^{j+5} + 2^{j+6}) x_1 x_j y_2 \\
&\quad + (2^{j+5} + 2^{2j+4}) x_j y_1 y_2 \\
f_4 &= \sum_{\substack{3 \le j \le n_x \\ 3 \le l \le n_y}} 2^{j+l+4} x_1 x_j y_1 y_l + (2^{j+l+3} + 2^{j+2l+2}) x_1 x_j y_l \\
&\quad + (2^{j+l+3} + 2^{2j+l+2}) x_j y_1 y_l \\
f_5 &= \sum_{2 \le k < l \le n_y} 2^{k+l+5} x_1 x_2 y_k y_l + 2^{k+l+4} x_1 y_k y_l \\
&\quad + (2^{k+l+4} + 2^{k+l+5}) x_2 y_k y_l \\
f_6 &= \sum_{2 \le i < j \le n_x} 2^{i+j+5} x_i x_j y_1 y_2 + 2^{i+j+4} x_i x_j y_1 \\
&\quad + (2^{i+j+4} + 2^{i+j+5}) x_i x_j y_2 \\
f_7 &= \sum_{\substack{3 \le j \le n_x \\ 2 \le k < l \le n_y}} 2^{j+k+l+3} x_1 x_j y_k y_l + (2^{j+k+l+2} + 2^{2j+k+l+1}) x_j y_k y_l \\
f_8 &= \sum_{\substack{2 \le i < j \le n_x \\ 3 \le l \le n_y}} 2^{i+j+l+3} x_i x_j y_1 y_l + (2^{i+j+l+2} + 2^{i+j+2l+1}) x_i x_j y_l \\
f_9 &= \sum_{\substack{2 \le i < j \le n_x \\ 2 \le k < l \le n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l .
\end{aligned}$$

Finally, the point is that we can rewrite the Hamiltonian from Equation 2.16 as the sum of all these sub-functions that we have created, such that:

$$\begin{aligned} H_N = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \\ + (\textit{original terms of degree less than 3.}) \end{aligned} \tag{2.22}$$

In Appendix B of this document we provide a complete proof as to why the Hamiltonians from Equations 2.16 and 2.22 are indeed equal.

Beyond that, since each sub-function f_i is written in the form of Equation 2.19 (except for the summations, of course), we can employ the method proposed by Dattani and Chau (2019), and use the substitution from Expression 2.20 to quadratize each sub-function individually. Naturally, this will lead to the quadratization of the entire Hamiltonian from Equation 2.22, which is presented next.

Once again, notice that each b auxiliary variable has been parameterized with indexes $ijkl$ to avoid confusion, since each b_{ijkl} is a different variable introduced by the quadratization method. Furthermore, this time we have also included the original quadratic and linear terms from Equation 2.13, which had previously been abbreviated.

$$\begin{aligned} H_N = {} & f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \\ & + \sum_{\substack{1 \le i \le n_x \\ 1 \le k \le n_y}} [2^{2i+2k} + 2^{2i+k+1} + 2^{i+2k+1} - 2^{i+k+1}(N-2)] x_i y_k \\ & + \sum_{1 \le i < j \le n_x} 2^{i+j+1} x_i x_j + \sum_{1 \le k < l \le n_y} 2^{k+l+1} y_k y_l \\ & + \sum_{1 \le i \le n_x} [2^{2i} - 2^{i+1}(N-1)] x_i + \sum_{1 \le k \le n_y} [2^{2k} - 2^{k+1}(N-1)] y_k \\ & + (N-1)^2, \end{aligned} \tag{2.23}$$

where:

$$
\begin{aligned}
f_1 &= 2^8(3x_1x_2 + 2x_1y_1 + 3x_1y_2 + 3x_2y_1 + 4x_2y_2 + 3y_1y_2) \\
&\quad - 2^7(9x_1 + 11x_2 + 9y_1 + 11y_2 - 14)b_{1212} \\
f_2 &= \sum_{3\le l\le n_y} 2^{2l+4}(x_1x_2 + x_1y_l + x_2y_l) \\
&\quad + 2^{l+5}(3x_1x_2 + 3x_1y_1 + 4x_1y_l + 5x_2y_1 + 6x_2y_l + 6y_1y_l) \\
&\quad - 2^{l+5}(6x_1 + 8x_2 + 8y_1 + 9y_l - 11)b_{121l} \\
&\quad - 2^{2l+4}(x_1 + x_2 + y_l - 1)b_{121l} \\
f_3 &= \sum_{3\le j\le n_x} 2^{2j+4}(x_jy_1 + x_jy_2 + y_1y_2) \\
&\quad + 2^{j+5}(6x_1x_j + 3x_1y_1 + 5x_1y_2 + 4x_jy_1 + 6x_jy_2 + 3y_1y_2) \\
&\quad - 2^{j+5}(8x_1 + 9x_j + 6y_1 + 8y_2 - 11)b_{1j12} \\
&\quad - 2^{2j+4}(x_j + y_1 + y_2 - 1)b_{1j12} \\
f_4 &= \sum_{\substack{3\le j\le n_x \\ 3\le l\le n_y}} -2^{j+l+3}(5*x_1 + 5*y_1 + 6*x_j + 6*y_l - 8)b_{1j1l} \\
&\quad + 2^{j+l+3}(2x_1y_1 + 3x_1x_j + 3x_1y_l + 3x_jy_1 + 3y_1y_l + 4x_jy_l) \\
&\quad + 2^{j+2l+2}(x_1x_j + x_1y_l + x_jy_l) \\
&\quad + 2^{2j+l+2}(x_jy_1 + y_1y_l + x_jy_l) \\
&\quad - 2^{j+2l+2}(x_1 + x_j + y_l - 1)b_{1j1l} \\
&\quad - 2^{2j+l+2}(y_1 + x_j + y_l - 1)b_{1j1l} \\
f_5 &= \sum_{2\le k<l\le n_y} -2^{k+l+4}(5x_1 + 7x_2 + 8y_k + 8y_l - 10)b_{12kl} \\
&\quad + 2^{k+l+4}(2x_1x_2 + 3x_1y_k + 3x_1y_l + 5x_2y_k + 5x_2y_l + 6y_ky_l) \\
f_6 &= \sum_{2\le i<j\le n_x} -2^{i+j+4}(8x_i + 8x_j + 5y_1 + 7y_2 - 10)b_{ij12} \\
&\quad + 2^{i+j+4}(6x_ix_j + 3x_iy_1 + 3x_jy_1 + 5x_iy_2 + 5x_jy_2 + 2y_1y_2)
\end{aligned}
$$

$$
\begin{aligned}
f_7 = \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} & 2^{j+k+l+3}(x_1x_j + x_1y_k + x_1y_l) \\
& + (3 * 2^{j+k+l+2} + 2^{2j+k+l+1})(x_jy_k + x_jy_l + y_ky_l) \\
& - (5 * 2^{j+k+l+2} + 2^{2j+k+l+1})(x_j + y_k + y_l)b_{1jkl} \\
& - 2^{j+k+l+4}b_{1jkl}x_1 + (7 * 2^{j+k+l+2} + 2^{2j+k+l+1})b_{1jkl} \\
f_8 = \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} & 2^{i+j+l+3}(x_iy_1 + x_jy_1 + y_1y_l) \\
& + (3 * 2^{i+j+l+2} + 2^{i+j+2l+1})(x_ix_j + x_iy_l + x_jy_l) \\
& - (5 * 2^{i+j+l+2} + 2^{i+j+2l+1})(x_i + x_j + y_l)b_{ij1l} \\
& - 2^{i+j+l+4}b_{ij1l}y_1 + (7 * 2^{i+j+l+2} + 2^{i+j+2l+1})b_{ij1l} \\
f_9 = \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} & 2^{i+j+k+l+2}(x_ix_j + x_iy_k + x_iy_l + x_jy_k + x_jy_l + y_ky_l) \\
& - 2 * 2^{i+j+k+l+2}(x_i + x_j + y_k + y_l)b_{ijkl} \\
& + 3 * 2^{i+j+k+l+2}b_{ijkl}.
\end{aligned}
$$

Chapter 3

Results

This chapter aims at drawing a quantitative comparison between the two quadratization methods presented earlier. For each case, we wish to evaluate certain metrics, such as:

– How the number of auxiliary variables grows.

– What is the number of resulting quadratic terms in the final Hamiltonian expression.

– What is the connectivity between qubits of the Hamiltonian.

– The range of coupling strengths required for the computation.

3.1 AUXILIARY VARIABLES

As we have seen before, the cost of quadratizing a Hamiltonian can be expressed in the number of new auxiliary variables that are introduced by the quadratization process. We have also seen that the method from Ishikawa (2011) is expected to introduce one auxiliary variable for each monomial of degree greater than 2. In our case, this only applies for monomials of the 3rd- and 4th-degree. On the other hand, Dattani and Chau (2019) proposed a method that introduces one auxiliary variable for each monomial of degree equal to four. Hence, a simple way to evaluate how many extra variables are introduced by each method is to count how many cubic and quartic monomials appear in the Hamiltonian from Equation 2.13.

From observing said equation, we find that the 3rd-degree monomials appear in one of two forms: $x_i x_j y_k$ (two x's and one y) or $x_i y_k y_l$ (one x and two y's). Meanwhile, the 4th-degree monomials have only one form: $x_i x_j y_k y_l$.

Let us establish the following notation:

$$\#(\textbf{format}) = \text{"}\textit{Number of monomials in said } \textbf{format}\text{"}.$$

In this context, we can view each monomial form as a combination of multiple x and y variables, for which we know the counts are n_x and n_y, respectively. Let us also remind ourselves that the total number of different combinations of p elements taken from a set of n elements is given by the expression:

$$C_p^n = \frac{n!}{p!(n-p)!}. \tag{3.1}$$

Thus, we can view the number of 3rd-degree monomials involving two x variables and one y variable to be the number of combinations of two x's, that is $C_2^{n_x}$, multiplied by the number of combinations of a single y, or simply $C_1^{n_y}$.

$$\begin{aligned}
\#(x_i x_j y_k) &= C_2^{n_x} \; * \; C_1^{n_y} \\
&= \frac{n_x!}{2!(n_x-2)!} \; * \; \frac{n_y!}{1!(n_y-1)!} \\
&= \frac{n_x(n_x-1)[(n_x-2)!]}{2!(n_x-2)!} \; * \; \frac{n_y[(n_y-1)!]}{1!(n_y-1)!} \\
&= \frac{n_x(n_x-1)}{2!} \; * \; \frac{n_y}{1!} \\
&= \frac{n_x n_y (n_x-1)}{2}.
\end{aligned}$$

A similar argument can be made for 3rd-degree monomials that contain one x variable and two y variables, leading us to:

$$\begin{aligned}
\#(x_i y_k y_l) &= C_1^{n_x} \; * \; C_2^{n_y} \\
&= \frac{n_x!}{1!(n_x-1)!} \; * \; \frac{n_y!}{2!(n_y-2)!} \\
&= \frac{n_x[(n_x-1)!]}{1!(n_x-1)!} \; * \; \frac{n_y(n_y-1)[(n_y-2)!]}{2!(n_y-2)!} \\
&= \frac{n_x}{1!} \; * \; \frac{n_y(n_y-1)}{2!} \\
&= \frac{n_x n_y(n_y-1)}{2}.
\end{aligned}$$

Finally, the number of quartic monomials of the format $x_i x_j y_k y_l$ is equal to the number of quadratic monomials composed only of x variables, multiplied by the number of quadratic monomials composed of y variables. In other words:

$$\begin{aligned}
\#(x_i x_j y_k y_l) &= C_2^{n_x} \; * \; C_2^{n_y} \\
&= \frac{n_x!}{2!(n_x-2)!} \; * \; \frac{n_y!}{2!(n_y-2)!} \\
&= \frac{n_x(n_x-1)[(n_x-2)!]}{2!(n_x-2)!} \; * \; \frac{n_y(n_y-1)[(n_y-2)!]}{2!(n_y-2)!} \\
&= \frac{n_x(n_x-1)}{2!} \; * \; \frac{n_y(n_y-1)}{2!} \\
&= \frac{n_x n_y(n_y-1)(n_y-1)}{4}.
\end{aligned}$$

From these results, we can conclude exactly how many 3rd and 4th-degree monomials of each type we expect to be part of the Hamiltonian from Equation 2.13. This leads us to the values presented in Table 3.1.

3.1.1 Ishikawa

We know that the method proposed by Ishikawa (2011), as presented in the previous chapter, introduces an auxiliary variable for each monomial of degree 3 and 4. Given the counts in Table 3.1, we show that the total number of extra variables introduced by this method is given by Theorem 3.1.

Table 3.1 – Number of monomials in factoring Hamiltonian.

Degree	**Monomial format**	**Monomial count**
3	$x_i x_j y_k$	$\frac{n_x n_y (n_x - 1)}{2}$
3	$x_i y_k y_l$	$\frac{n_x n_y (n_y - 1)}{2}$
4	$x_i x_j y_k y_l$	$\frac{n_x n_y (n_x - 1)(n_y - 1)}{4}$

Source: developed by the author.

Theorem 3.1 (Ishikawa's Auxiliary Variables Theorem)**.** *The number of auxiliary variables introduced into the integer factorization Hamiltonian by Ishikawa's quadratization method, referred to as M_{Ishi}, is given by the count of 3rd- and 4th-degree monomials that appear in the Hamiltonian:*

$$\begin{aligned}
M_{Ishi} &= \frac{n_x n_y (n_x - 1)}{2} + \frac{n_x n_y (n_y - 1)}{2} + \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{n_x n_y}{2} \left((n_x - 1) + (n_y - 1) + \frac{(n_x - 1)(n_y - 1)}{2} \right) \\
&= \frac{n_x n_y}{2} \left(\frac{2(n_x - 1) + 2(n_y - 1) + (n_x - 1)(n_y - 1)}{2} \right) \\
&= \frac{n_x n_y}{2} \left(\frac{2n_x - 2 + 2n_y - 2 + n_x n_y - n_x - n_y + 1}{2} \right) \\
&= \frac{n_x n_y}{2} \left(\frac{n_x n_y + n_x + n_y - 3}{2} \right) \\
&= \frac{n_x n_y (n_x n_y + n_x + n_y - 3)}{4}.
\end{aligned}$$

3.1.2 Dattani-Chau

Similarly, we have seen in the previous chapter that the quadratization method proposed by Dattani and Chau (2019) introduces one extra binary variable for each 4th-degree monomial in our Hamiltonian. Therefore, the total number of auxiliary variables used by this method is given by Theorem 3.2.

Theorem 3.2 (Dattani-Chau's Auxiliary Variables Theorem)**.** *The number of auxiliary variables introduced into the integer factorization Hamiltonian by Dattani-Chau's quadratization method, referred to as M_{Datt}, is equal to the number of quartic monomials present in the Hamiltonian:*

$$\begin{aligned} M_{Datt} &= \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\ &= \frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{4}. \end{aligned}$$

3.1.3 Comparison

From Theorems 3.1 and 3.2 we can conclude that both methods introduce a similar amount of extra variables, because both expressions present a growth of $O(n^4)$ in terms of the number of bits of N. This can be seen more clearly if we recall our previous definition for n_x and n_y from Equations 2.4, where both n_x and n_y are defined as growing with a rate of about $O(lgN)$, which is equivalent to $O(n)$, where n is the number of bits in N.

Thus, we argue that a simple estimate for the asymptotic growth of both M_{Ishi} and M_{Datt} is to replace every n_x and n_y in those expressions with just n. In this case, as N grows to become a very large number and tends to infinity, it is fair to say that the growth of both these expressions is of about $O(n^4)$.

For small values of N, however, the Dattani-Chau method presents a considerable advantage over Ishikawa's approach, since it reduces the number of auxiliary variables needed by about a cubic polynomial-worth of qubits, which is the difference between the values M_{Ishi} and M_{Datt}:

$$\begin{aligned} M_{\text{Ishi}} - M_{\text{Datt}} &= \frac{n_x n_y(n_x n_y + n_x + n_y - 3)}{4} - \frac{n_x n_y(n_x - 1)(n_y - 1)}{4} \\ &= \frac{n_x n_y(n_x n_y + n_x + n_y - 3)}{4} - \frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{4} \\ &= \frac{n_x n_y((n_x n_y + n_x + n_y - 3) - (n_x n_y - n_x - n_y + 1))}{4} \\ &= \frac{n_x n_y(n_x n_y + n_x + n_y - 3 - n_x n_y + n_x + n_y - 1)}{4} \\ &= \frac{n_x n_y(2n_x + 2n_y - 4)}{4} \\ &= \frac{n_x n_y(n_x + n_y - 2)}{2}. \end{aligned}$$

3.2 NUMBER OF TERMS

We can also estimate the maximum number of quadratic terms that each quadratization method is expected to produce. For this, we should consider the five different formats of quadratic monomials that we expect to see in the final Hamiltonian. They are:

– xx,

– xy,

– yy,

– bx,

– and by,

where b are the auxiliary variables introduced by each method.

Since we know all the counts for each type of variable – meaning n_x for x variables, n_y for y variables and M_{Method} for b variables –, we can estimate the number of quadratic terms for both methods.

Let us first estimate the count for quadratic monomials of the forms xx, xy and yy. If we observe the Hamiltonian formula from Equation 2.13, we notice these three summations:

(a) $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq k \leq n_y}} [2^{2i+2k} + 2^{2i+k+1} + 2^{i+2k+1} - 2^{i+k+1}(N-2)]x_i y_k$,

(b) $\sum_{1 \leq i < j \leq n_x} 2^{i+j+1} x_i x_j$,

(c) and $\sum_{1 \leq k < l \leq n_y} 2^{k+l+1} y_k y_l$.

We argue that these three components (a), (b) and (c) already involve all possible monomials of the forms xx, xy and yy for a single Hamiltonian. Moreover, since these components are also carried over to the method-specific Hamiltonians from Equations 2.18 and 2.23, then we know they will be the same for both methods (except maybe for their coefficients, which we do not care about for the purpose of counting these monomials).

From this, we conclude that the number of monomials of the format xx can be expressed as the combination of two elements taken from a set of n_x elements. In other words, retrieving the notation we used before, we have:

$$\begin{aligned} \#(xx) &= C_2^{n_x} \\ &= \frac{n_x(n_x-1)}{2}. \end{aligned}$$

The same argument applies, of course, for monomials of the form xy and yy, where we have:

$$\begin{aligned} \#(xy) &= C_1^{n_x} * C_1^{n_y} \\ &= n_x n_y, \end{aligned}$$

and also:

$$\begin{aligned} \#(yy) &= C_2^{n_y} \\ &= \frac{n_y(n_y-1)}{2}. \end{aligned}$$

Further on, we should now address the count for monomials of the formats bx and by, which are specific to each method.

3.2.1 Ishikawa

We have already seen that Ishikawa's method introduces one auxiliary variable b for each monomial of degree three and four in the original Hamiltonian. We can clearly see in Equation 2.18 that, after applying the quadratization method, each b will be involved in three or four quadratic monomials, depending on whether it comes from a 3rd- or 4th-degree term.

From this, we can view each b variable as being connected to either 3 or 4 other variables (namely x's and y's). Since we also know that each b is associated to either a 3rd- or a 4th-degree monomial, then the number of quadratic monomials that involve b variables is given as follows:

$$\begin{aligned}
\#(bx)_{\text{Ishi}} &= 2 * \#(x_i x_j y_k) + 1 * \#(x_i y_k y_l) + 2 * \#(x_i x_j y_k y_l), \text{ and} \\
\#(by)_{\text{Ishi}} &= 1 * \#(x_i x_j y_k) + 2 * \#(x_i y_k y_l) + 2 * \#(x_i x_j y_k y_l).
\end{aligned}$$

But we already now the counts for the number of 3rd- and 4th-degree monomials, which we calculated in the previous section. So we can just substitute these values:

$$\begin{aligned}
\#(bx)_{\text{Ishi}} &= 2 * \#(x_i x_j y_k) + 1 * \#(x_i y_k y_l) + 2 * \#(x_i x_j y_k y_l) \\
&= \frac{2n_x n_y(n_x-1)}{2} + \frac{n_x n_y(n_y-1)}{2} + \frac{2n_x n_y(n_x-1)(n_y-1)}{4} \\
&= \frac{2n_x n_y(n_x-1) + n_x n_y(n_y-1) + n_x n_y(n_x-1)(n_y-1)}{2} \\
&= \frac{n_x n_y[2(n_x-1) + (n_y-1) + (n_x-1)(n_y-1)]}{2} \\
&= \frac{n_x n_y(2n_x - 2 + n_y - 1 + n_x n_y - n_x - n_y + 1)}{2} \\
&= \frac{n_x n_y(n_x n_y + n_x - 2)}{2},
\end{aligned}$$

and also:

$$\begin{aligned}
\#(by)_{\text{Ishi}} &= 1 * \#(x_i x_j y_k) + 2 * \#(x_i y_k y_l) + 2 * \#(x_i x_j y_k y_l) \\
&= \frac{n_x n_y (n_x - 1)}{2} + \frac{2 n_x n_y (n_y - 1)}{2} + \frac{2 n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{n_x n_y (n_x - 1) + 2 n_x n_y (n_y - 1) + n_x n_y (n_x - 1)(n_y - 1)}{2} \\
&= \frac{n_x n_y [(n_x - 1) + 2(n_y - 1) + (n_x - 1)(n_y - 1)]}{2} \\
&= \frac{n_x n_y (n_x - 1 + 2n_y - 2 + n_x n_y - n_x - n_y + 1)}{2} \\
&= \frac{n_x n_y (n_x n_y + n_y - 2)}{2}.
\end{aligned}$$

3.2.2 Dattani-Chau

For Dattani-Chau's method, however the count for bx and by monomials is simpler. Since we know this method introduces one variable for each 4th-degree monomial, we can simply estimate the number of resulting quadratic monomials directly from the following expressions:

$$\begin{aligned}
\#(bx)_{\text{Datt}} &= 2 * \#(x_i x_j y_k y_l) \\
&= 2 * \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{n_x n_y (n_x - 1)(n_y - 1)}{2} \\
&= \frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{2}, \\
\#(by)_{\text{Datt}} &= 2 * \#(x_i x_j y_k y_l) \\
&= 2 * \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{n_x n_y (n_x - 1)(n_y - 1)}{2} \\
&= \frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{2}.
\end{aligned}$$

As expected, given the symmetry of this quadratization method, we see that $\#(bx)_{\text{Datt}}$ and $\#(by)_{\text{Datt}}$ are in fact equal.

3.2.3 Comparison

We have compiled the results from the previous sections into Table 3.2, in order to better compare the two methods. These results show that even though the counts for monomials of the types xx, xy and yy are the same for both methods, the amount of monomials of the types bx and by are different – namely, Ishikawa's method presents more monomials of these two formats.

This result agrees with our findings from the previous section that stated that Ishikawa's Hamiltonian contains more b auxiliary qubits, so it makes sense for it to also contains more quadratic terms involving these variables.

Table 3.2 – Number of resulting quadratic terms for each method.

Monomial format	**Ishikawa's count**	**Dattani-Chau's count**
xx	$\frac{n_x(n_x-1)}{2}$	$\frac{n_x(n_x-1)}{2}$
xy	$n_x n_y$	$n_x n_y$
yy	$\frac{n_y(n_y-1)}{2}$	$\frac{n_y(n_y-1)}{2}$
bx	$\frac{n_x n_y(n_x n_y + n_x - 2)}{2}$	$\frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{2}$
by	$\frac{n_x n_y(n_x n_y + n_y - 2)}{2}$	$\frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{2}$

Source: developed by the author.

3.3 TOPOLOGY

Further on, we should also consider the connectivity aspects of the qubits in our Hamiltonian. To do so, let us first consider an example for a small value of N, so we can look at the expressions for each method.

Let us consider the case where $N = 85 = 5 * 17$. We expect, of course, that $x = 5$ and $y = 17$, since we previously defined that $x < y$. If we assume that we don't know these values from the start, we can use the formulas from Equations 2.4 to find an approximation for the length of each factor, in number of bits, thus finding n_x and n_y to be:

$$\begin{aligned}
n_x &= \lceil \log_2 \lfloor \sqrt{85} \rfloor_{odd} \rceil - 1 \\
&= \lceil \log_2 \lfloor 9.22 \rfloor_{odd} \rceil - 1 \\
&= \lceil \log_2 9 \rceil - 1 \\
&= \lceil 3.17 \rceil - 1 \\
&= 4 - 1 \\
&= 3,
\end{aligned}$$

$$\begin{aligned}
n_y &= \lceil \log_2 \lfloor \frac{85}{3} \rfloor \rceil - 1 \\
&= \lceil \log_2 \lfloor 28.33 \rfloor \rceil - 1 \\
&= \lceil \log_2 28 \rceil - 1 \\
&= \lceil 4.8 \rceil - 1 \\
&= 5 - 1 \\
&= 4.
\end{aligned}$$

Now, using our Equation 2.13, we reach the following Hamiltonian H_{85}:

$$\begin{aligned}
H_{85} = {} & 256x_1x_2y_1y_2 + 512x_1x_2y_1y_3 + 1024x_1x_2y_1y_4 \\
& + 1024x_1x_2y_2y_3 + 2048x_1x_2y_2y_4 + 4096x_1x_2y_3y_4 \\
& + 512x_1x_3y_1y_2 + 1024x_1x_3y_1y_3 + 2048x_1x_3y_1y_4 \\
& + 2048x_1x_3y_2y_3 + 4096x_1x_3y_2y_4 + 8192x_1x_3y_3y_4 \\
& + 1024x_2x_3y_1y_2 + 2048x_2x_3y_1y_3 + 4096x_2x_3y_1y_4 \\
& + 4096x_2x_3y_2y_3 + 8192x_2x_3y_2y_4 + 16384x_2x_3y_3y_4 \\
& + 128x_1x_2y_1 + 384x_1x_2y_2 + 1280x_1x_2y_3 + 4608x_1x_2y_4 \\
& + 256x_1x_3y_1 + 768x_1x_3y_2 + 2560x_1x_3y_3 + 9216x_1x_3y_4 \\
& + 128x_1y_1y_2 + 256x_1y_1y_3 + 512x_1y_1y_4 + 512x_1y_2y_3 \\
& + 1024x_1y_2y_4 + 2048x_1y_3y_4 + 512x_2x_3y_1 + 1536x_2x_3y_2 \\
& + 5120x_2x_3y_3 + 18432x_2x_3y_4 + 384x_2y_1y_2 + 768x_2y_1y_3 \\
& + 1536x_2y_1y_4 + 1536x_2y_2y_3 + 3072x_2y_2y_4 + 6144x_2y_3y_4 \\
& + 1280x_3y_1y_2 + 2560x_3y_1y_3 + 5120x_3y_1y_4 + 5120x_3y_2y_3 \\
& + 10240x_3y_2y_4 + 20480x_3y_3y_4 + 16x_1x_2 + 32x_1x_3 - 616x_1y_1 \\
& - 1168x_1y_2 - 2080x_1y_3 - 3136x_1y_4 + 64x_2x_3 - 1168x_2y_1 \\
& - 2144x_2y_2 - 3520x_2y_3 - 3968x_2y_4 - 2080x_3y_1 - 3520x_3y_2 \\
& - 4480x_3y_3 + 1280x_3y_4 + 16y_1y_2 + 32y_1y_3 + 64y_1y_4 \\
& + 64y_2y_3 + 128y_2y_4 + 256y_3y_4 - 332x_1 - 656x_2 - 1280x_3 \\
& - 332y_1 - 656y_2 - 1280y_3 - 2432y_4 + 7056.
\end{aligned}$$

Moreover, we can use our Equation 2.18 to skip ahead and find the expression for the same Hamiltonian H_{85}, but quadratized via the Ishikawa method, which we will denote as H_{85}^{Ishi}:

$$\begin{aligned}
H_{85}^{\text{Ishi}} = &- 512b_1(x_1 + x_2 + y_1 + y_2) + 768b_1 \\
&- 1024b_2(x_1 + x_2 + y_1 + y_3) + 1536b_2 \\
&- 2048b_3(x_1 + x_2 + y_1 + y_4) + 3072b_3 \\
&- 2048b_4(x_1 + x_2 + y_2 + y_3) + 3072b_4 \\
&- 4096b_5(x_1 + x_2 + y_2 + y_4) + 6144b_5 \\
&- 8192b_6(x_1 + x_2 + y_3 + y_4) + 12288b_6 \\
&- 1024b_7(x_1 + x_3 + y_1 + y_2) + 1536b_7 \\
&- 2048b_8(x_1 + x_3 + y_1 + y_3) + 3072b_8 \\
&- 4096b_9(x_1 + x_3 + y_1 + y_4) + 6144b_9 \\
&- 4096b_{10}(x_1 + x_3 + y_2 + y_3) + 6144b_{10} \\
&- 8192b_{11}(x_1 + x_3 + y_2 + y_4) + 12288b_{11} \\
&- 16384b_{12}(x_1 + x_3 + y_3 + y_4) + 24576b_{12} \\
&- 2048b_{13}(x_2 + x_3 + y_1 + y_2) + 3072b_{13} \\
&- 4096b_{14}(x_2 + x_3 + y_1 + y_3) + 6144b_{14} \\
&- 8192b_{15}(x_2 + x_3 + y_1 + y_4) + 12288b_{15} \\
&- 8192b_{16}(x_2 + x_3 + y_2 + y_3) + 12288b_{16} \\
&- 16384b_{17}(x_2 + x_3 + y_2 + y_4) + 24576b_{17} \\
&- 32768b_{18}(x_2 + x_3 + y_3 + y_4) + 49152b_{18} \\
&- 128b_{19}(x_1 + x_2 + y_1 - 1) - 384b_{20}(x_1 + x_2 + y_2 - 1) \\
&- 1280b_{21}(x_1 + x_2 + y_3 - 1) - 4608b_{22}(x_1 + x_2 + y_4 - 1) \\
&- 256b_{23}(x_1 + x_3 + y_1 - 1) - 768b_{24}(x_1 + x_3 + y_2 - 1) \\
&- 2560b_{25}(x_1 + x_3 + y_3 - 1) - 9216b_{26}(x_1 + x_3 + y_4 - 1) \\
&- 512b_{27}(x_2 + x_3 + y_1 - 1) - 1536b_{28}(x_2 + x_3 + y_2 - 1) \\
&- 5120b_{29}(x_2 + x_3 + y_3 - 1) - 18432b_{30}(x_2 + x_3 + y_4 - 1) \\
&- 128b_{31}(x_1 + y_1 + y_2 - 1) - 256b_{32}(x_1 + y_1 + y_3 - 1) \\
&- 512b_{33}(x_1 + y_1 + y_4 - 1) - 512b_{34}(x_1 + y_2 + y_3 - 1)
\end{aligned}$$

$$
\begin{aligned}
&- 1024b_{35}(x_1 + y_2 + y_4 - 1) - 2048b_{36}(x_1 + y_3 + y_4 - 1)\\
&- 384b_{37}(x_2 + y_1 + y_2 - 1) - 768b_{38}(x_2 + y_1 + y_3 - 1)\\
&- 1536b_{39}(x_2 + y_1 + y_4 - 1) - 1536b_{40}(x_2 + y_2 + y_3 - 1)\\
&- 3072b_{41}(x_2 + y_2 + y_4 - 1) - 6144b_{42}(x_2 + y_3 + y_4 - 1)\\
&- 1280b_{43}(x_3 + y_1 + y_2 - 1) - 2560b_{44}(x_3 + y_1 + y_3 - 1)\\
&- 5120b_{45}(x_3 + y_1 + y_4 - 1) - 5120b_{46}(x_3 + y_2 + y_3 - 1)\\
&- 10240b_{47}(x_3 + y_2 + y_4 - 1) - 20480b_{48}(x_3 + y_3 + y_4 - 1)\\
&+ 15376x_1x_2 + 30752x_1x_3 + 6040x_1y_1 + 11632x_1y_2\\
&+ 21472x_1y_3 + 35776x_1y_4 + 61504x_2x_3 + 11120x_2y_1\\
&+ 21408x_2y_2 + 39488x_2y_3 + 65664x_2y_4 + 18400x_3y_1\\
&+ 35392x_3y_2 + 65152x_3y_3 + 107776x_3y_4 + 3600y_1y_2\\
&+ 7200y_1y_3 + 14400y_1y_4 + 14400y_2y_3 + 28800y_2y_4\\
&+ 57600y_3y_4 - 332x_1 - 656x_2 - 1280x_3 - 332y_1\\
&- 656y_2 - 1280y_3 - 2432y_4 + 7056.
\end{aligned}
$$

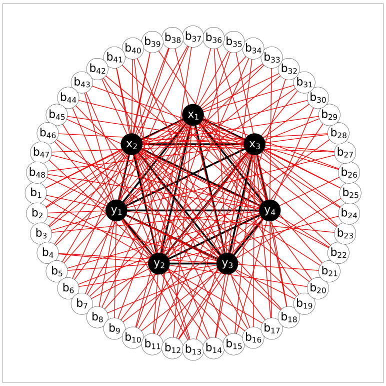
Additionally, we can calculate the expression for the Hamiltonian H_{85} quadratized via the Dattani-Chau method, which will be denoted as H_{85}^{Datt}. This is a direct application of Equation 2.23:

$$
\begin{aligned}
H_{85}^{\mathrm{Datt}} = {} & (1792 - 1152x_1 - 1408x_2 - 1152y_1 - 1408y_2)b_1 \\
& + (3840 - 2560x_1 - 3072x_2 - 2048y_1 - 3328y_3)b_2 \\
& + (9728 - 7168x_1 - 8192x_2 - 4096y_1 - 8704y_4)b_3 \\
& + (3840 - 2048x_1 - 3328x_3 - 2560y_1 - 3072y_2)b_4 \\
& + (8192 - 4608x_1 - 7168x_3 - 4608y_1 - 7168y_3)b_5 \\
& + (20480 - 13312x_1 - 18432x_3 - 9216y_1 - 18432y_4)b_6 \\
& + (5120 - 2560x_1 - 3584x_2 - 4096y_2 - 4096y_3)b_7 \\
& + (10240 - 5120x_1 - 7168x_2 - 8192y_2 - 8192y_4)b_8 \\
& + (20480 - 10240x_1 - 14336x_2 - 16384y_3 - 16384y_4)b_9 \\
& + (11264 - 4096x_1 - 9216x_3 - 9216y_2 - 9216y_3)b_{10} \\
& + (22528 - 8192x_1 - 18432x_3 - 18432y_2 - 18432y_4)b_{11} \\
& + (45056 - 16384x_1 - 36864x_3 - 36864y_3 - 36864y_4)b_{12} \\
& + (5120 - 4096x_2 - 4096x_3 - 2560y_1 - 3584y_2)b_{13} \\
& + (11264 - 9216x_2 - 9216x_3 - 4096y_1 - 9216y_3)b_{14} \\
& + (30720 - 26624x_2 - 26624x_3 - 8192y_1 - 26624y_4)b_{15} \\
& + (12288 - 8192x_2 - 8192x_3 - 8192y_2 - 8192y_3)b_{16} \\
& + (24576 - 16384x_2 - 16384x_3 - 16384y_2 - 16384y_4)b_{17} \\
& + (49152 - 32768x_2 - 32768x_3 - 32768y_3 - 32768y_4)b_{18} \\
& + 15376x_1x_2 + 30752x_1x_3 + 6040x_1y_1 + 11632x_1y_2 \\
& + 21472x_1y_3 + 35776x_1y_4 + 61504x_2x_3 + 11120x_2y_1 \\
& + 21408x_2y_2 + 39488x_2y_3 + 65664x_2y_4 + 18400x_3y_1 \\
& + 35392x_3y_2 + 65152x_3y_3 + 107776x_3y_4 + 3600y_1y_2 \\
& + 7200y_1y_3 + 14400y_1y_4 + 14400y_2y_3 + 28800y_2y_4 \\
& + 57600y_3y_4 - 332x_1 - 656x_2 - 1280x_3 - 332y_1 \\
& - 656y_2 - 1280y_3 - 2432y_4 + 7056.
\end{aligned}
$$

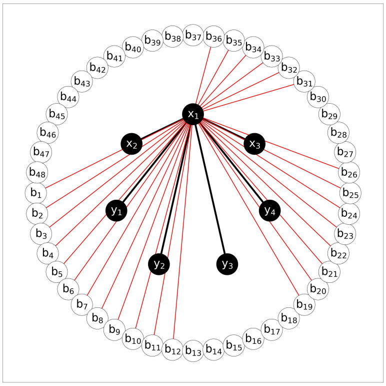
We can see from the polynomial expressions that H_{85}^{Ishi} involves more extra b_i variables than H_{85}^{Datt}, as we expected. Now, if we interpret each qubit of a Hamiltonian as a node, and each quadratic monomial as an undirected edge, then we can represent each quadratized Hamiltonian as an undirected graph. By doing this, we can view the quadratic polynomial expressions that describe each methods' Hamiltonian in a more visual manner, which may help us understand how each operator is organized in terms of its linear monomials (nodes) and quadratic monomials (edges).

These visualizations are given in Figure 3.1 for H_{85}^{Ishi} and in Figure 3.2 for H_{85}^{Datt}.

Figure 3.1 – Graph representation of Ishikawa's Hamiltonian.



(a) Graph representation of H_{85}^{Ishi}, previously given in polynomial form.



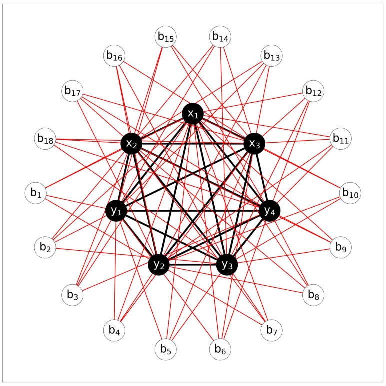
(b) Same graph representation of H_{85}^{Ishi}, but showing only edges connected to variable x_1.

Graph representation for the H_{85}^{Ishi} Hamiltonian, previously shown in its polynomial form. Variables are represented as nodes (usual x and y qubits in black, auxiliary b qubits added by the method in white). Black edges show the positive-coefficient connections between x and y variables. Red edges show negative-coefficient connections, which in this case always involve auxiliary variables.

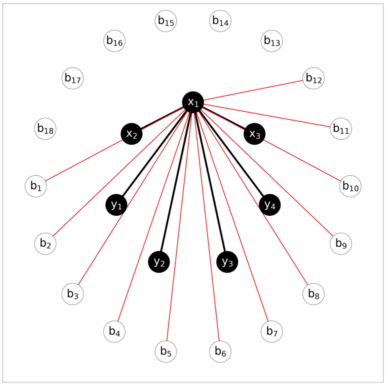
From these graphs, we can see more clearly that each Hamiltonian contains one fully connected core of x and y variables, represented in black, plus the set of b variables, which are more abundant, yet less connected. We note this arrangement has more to do with the way we chose to encode the factorization problem into the Hamiltonian, rather than to do with the quadratization methods.

Once again, we note that Ishikawa's method produces more auxiliary variables, and thus has more qubit connections overall, when compared to Dattani-Chau's formula.

Figure 3.2 – Graph representation of Dattani-Chau's Hamiltonian.



(a) Graph representation of H_{85}^{Datt}, previously given in polynomial form.



(b) Same graph representation of H_{85}^{Datt}, but showing only edges connected to variable x_1.

Graph representation for the H_{85}^{Datt} Hamiltonian, previously shown in its polynomial form. Variables are represented as nodes (usual x and y qubits in black, auxiliary b qubits added by the method in white). Black edges show the positive-coefficient connections between x and y variables. Red edges show negative-coefficient connections, which in this case always involve auxiliary variables.

3.4 CONNECTIVITY

Moreover, we can also estimate the connectivity of the qubits in each Hamiltonian, by which we mean the average number of connections per qubit. We can easily calculate this metric for both Hamiltonians if we view them as graphs, since the value we are after can be expressed as the number of connections (edges) divided by the number of qubits, or nodes. This value is sometimes called the graph's beta index, or simply β. The higher the graph's beta index, the more connected it is:

$$\beta = \frac{E}{V} = \frac{\text{"Number of edges"}}{\text{"Number of nodes"}}.$$

From what we have seen so far, we know the number of qubits (nodes) in our Hamiltonians is given by the sum of x and y variables, plus the number of b auxiliary variables, denoted by

M_{Method}, which is method-specific as per Theorems 3.1 and 3.2:

$$V = n_x + n_y + M_{\text{Method}}.$$

As for the number of connections, or edges, it is the same as the number of quadratic monomials in the quadratized Hamiltonian expression. Therefore, it can be expressed as the sum of the counts of each monomial format, previously shown in Table 3.2:

$$E = \#(xx) + \#(xy) + \#(yy) + \#(bx) + \#(by).$$

With this in mind, and using the values from Table 3.2, let us calculate the number of edges for Ishikawa's Hamiltonian, which we will refer to as E_{Ishi}:

$$\begin{aligned}
E_{\text{Ishi}} &= \frac{n_x(n_x-1)}{2} + n_x n_y + \frac{n_y(n_y-1)}{2} \\
&\quad + \frac{n_x n_y(n_x n_y + n_x - 2)}{2} + \frac{n_x n_y(n_x n_y + n_y - 2)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1)}{2} \\
&\quad + \frac{2n_x n_y + n_x n_y(n_x n_y + n_x - 2) + n_x n_y(n_x n_y + n_y - 2)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1)}{2} \\
&\quad + \frac{n_x n_y(n_x n_y + n_x n_y + n_y + n_x - 2 - 2 + 2)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1) + n_x n_y(2n_x n_y + n_y + n_x - 2)}{2}.
\end{aligned}$$

And we calculate the same for Dattani-Chau's Hamiltonian:

$$\begin{aligned}
E_{\text{Datt}} &= \frac{n_x(n_x-1)}{2} + n_x n_y + \frac{n_y(n_y-1)}{2} \\
&+ \frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{2} \\
&+ \frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1)}{2} \\
&+ \frac{2n_x n_y + n_x n_y(n_x n_y - n_x - n_y + 1)}{2} \\
&+ \frac{n_x n_y(n_x n_y - n_x - n_y + 1)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1)}{2} \\
&+ \frac{n_x n_y(n_x n_y + n_x n_y - n_x - n_y - n_x - n_y + 1 + 1 + 2)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1) + n_x n_y(2n_x n_y - 2n_x - 2n_y + 4)}{2} \\
&= \frac{n_x(n_x-1) + n_y(n_y-1) + 2n_x n_y(n_x n_y - n_x - n_y + 2)}{2}.
\end{aligned}$$

Finally, it is possible to calculate both β_{Ishi}, as shown below:

$$\begin{aligned}
\beta_{\text{Ishi}} &= \frac{E_{\text{Ishi}}}{V_{\text{Ishi}}} \\
&= \frac{1}{2}\frac{n_x(n_x-1) + n_y(n_y-1) + n_x n_y(2n_x n_y + n_y + n_x - 2)}{n_x + n_y + M_{\text{Ishi}}} \\
&= \frac{1}{2}\frac{n_x(n_x-1) + n_y(n_y-1) + n_x n_y(2n_x n_y + n_y + n_x - 2)}{n_x + n_y + \frac{n_x n_y(n_x n_y + n_x + n_y - 3)}{4}} \\
&= 2\frac{n_x(n_x-1) + n_y(n_y-1) + n_x n_y(2n_x n_y + n_y + n_x - 2)}{4n_x + 4n_y + n_x n_y(n_x n_y + n_x + n_y - 3)},
\end{aligned}$$

and also β_{Datt}:

$$\begin{aligned}\beta_{\text{Datt}} &= \frac{E_{\text{Datt}}}{V_{\text{Datt}}} \\ &= \frac{1}{2}\frac{n_x(n_x-1)+n_y(n_y-1)+2n_xn_y(n_xn_y-n_x-n_y+2)}{n_x+n_y+M_{\text{Datt}}} \\ &= \frac{1}{2}\frac{n_x(n_x-1)+n_y(n_y-1)+2n_xn_y(n_xn_y-n_x-n_y+2)}{n_x+n_y+\dfrac{n_xn_y(n_x-1)(n_y-1)}{4}} \\ &= 2\frac{n_x(n_x-1)+n_y(n_y-1)+2n_xn_y(n_xn_y-n_x-n_y+2)}{4n_x+4n_y+n_xn_y(n_xn_y-n_x-n_y+1)}.\end{aligned}$$

3.4.1 Comparison

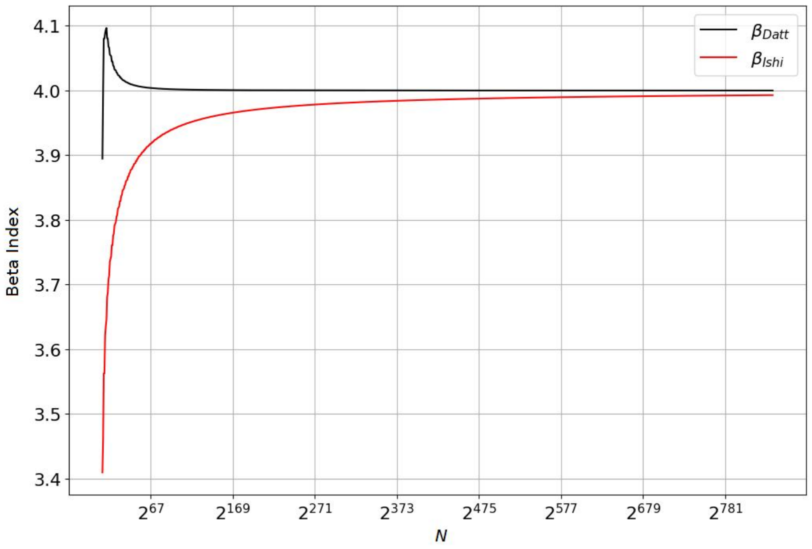
In this section we aim to compare the beta indexes of each method. First, we will look at the graph for both expressions. This is given in Figure 3.3. In the plot, we notice that for very large values of N (x-axis) both β_{Ishi} and β_{Datt} seem to approach the value 4.

The beta index for Ishikawa's Hamiltonian, however, presents a slight advantage, since its value is smaller than β_{Datt} for pretty much every value of N we have analysed. A way to interpret these values is to consider that, for the same value of N, Ishikawa's quadratized Hamiltonian presents more qubits overall (and consequently more connections), but its graph is slightly less connected, meaning the edge-node count ratio (i.e. the beta index) is smaller than that of Dattani-Chau's Hamiltonian.

This measure, however, does not mean Ishikawa's Hamiltonian is simpler or uses less connections than Dattani-Chau's. In fact, as we have seen in the example for Figures 3.1 and 3.2, the opposite is true.

In reality, this difference in beta indexes can be explained by once again considering that Ishikawa's quadratization method introduces b variables for both 3rd- and 4th-degree monomials, while Dattani-Chau's only introduces b variables for 4th-degree monomials. The fact that Ishikawa's method produces some auxiliary variables that only connect to three other variables (meaning those originating from the individual quadratization of 3rd-degree monomials) tends

Figure 3.3 – Growth of beta indexes.



Plot showing the growth of beta indexes β_{Ishi} (in red) and β_{Datt} (in black) in relation to the number N, in the x-axis. Note that the x-axis is presented in logarithmic scale.

to bring down the average of edges-per-node, in the graph, to a value less than 4.

But, as the plot in Figure 3.3 suggests, both of these values get closer and closer to 4 as N grows to become a very large number. This can be justified by looking at the limits for β_{Ishi} and β_{Datt} when N tends to infinity. In this context, both n_x and n_y also tend to infinity, so we can write:

$$\begin{aligned}&\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} \beta_{\text{Ishi}}\\ =\ &\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{n_x(n_x-1)+n_y(n_y-1)+n_xn_y(2n_xn_y+n_y+n_x-2)}{4n_x+4n_y+n_xn_y(n_xn_y+n_x+n_y-3)}.\end{aligned}$$

Analyzing this limit, we can see that most small terms will be overtaken by the larger terms, as the values tend to infinity. In this context, even if we are not being very rigorous, we can still cancel out most small terms, leaving only the ones highlighted in red in the equation below:

$$
\begin{aligned}
&= \lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{n_x(n_x-1)+n_y(n_y-1)+n_xn_y(2n_xn_y+n_y+n_x-2)}{4n_x+4n_y+n_xn_y(n_xn_y+n_x+n_y-3)}\\
&= \lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{n_xn_y(2n_xn_y)}{n_xn_y(n_xn_y)}\\
&= 2\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} \frac{2n_x^2n_y^2}{n_x^2n_y^2}\\
&= 2\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\\
&= 4.
\end{aligned}
$$

A similar argument can be made for the limit of β_{Datt} when N tends to infinity:

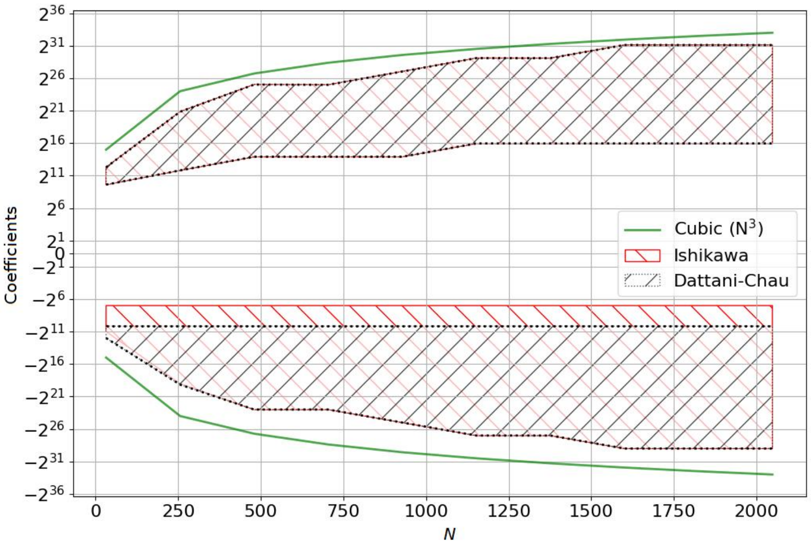
$$
\begin{aligned}
&\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} \beta_{\text{Datt}}\\
&= \lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{n_x(n_x-1)+n_y(n_y-1)+2n_xn_y(n_xn_y-n_x-n_y+2)}{4n_x+4n_y+n_xn_y(n_xn_y-n_x-n_y+1)}\\
&= \lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{n_x(n_x-1)+n_y(n_y-1)+2n_xn_y(n_xn_y-n_x-n_y+2)}{4n_x+4n_y+n_xn_y(n_xn_y-n_x-n_y+1)}\\
&= \lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\frac{2n_xn_y(n_xn_y)}{n_xn_y(n_xn_y)}\\
&= 2\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} \frac{2n_x^2n_y^2}{n_x^2n_y^2}\\
&= 2\lim_{\substack{n_x\to\infty\\ n_y\to\infty}} 2\\
&= 4.
\end{aligned}
$$

In the end, we see that both beta indexes do indeed approach 4 at infinity, which corroborates our initial intuition and suggests that the connectivity between qubits in each Hamiltonian is very similar for both methods.

3.5 RANGE OF COEFFICIENTS

Furthermore, we should also look at how the range of coefficients grows as the number N gets larger. In Figure 3.4 we show the curves for the highest and lowest monomial coefficient in the Hamiltonian polynomials. The x-axis denotes the values of N. In the y-axis, Ishikawa's expression is shown in red, while Dattani-Chau's Hamiltonian is represented by the black dotted lines.

Figure 3.4 – Range of coefficients.



Plot showing the growing range of coefficients for each Hamiltonian expression. The y-axis, in logarithmic scale, shows the coefficient range for both Ishikawa's (red) and Dattani-Chau's (black dotted) Hamiltonian. The x-axis shows values of N. The negative areas show the range of least-to-most **negative** coefficients for said value of N, while positive areas of course show the range of least-to-most **positive** coefficients for that same value. The green curves show positive and negative values of N^3 for comparison.

From this plot, we take away three things: first, even for relatively small values of N, such as $N = 2048$, the monomial coefficients reach the ranges of about $[-2^{31}, +2^{31}]$, which are considerably large numbers. Second, the ranges of coefficients for both methods are very similar, and there is not much difference between the two methods considering this aspect.

And third, we can see from the plot that up until $N = 2048$, the monomial coefficients for both methods stay bounded by the green curves, which represent the range $[-N^3, +N^3]$. Although we could not prove this characteristic for larger values of N, it does suggest a trend that says the absolute value of these coefficients tends to range around (possibly strictly below) N^3. This is important, because it indicates that these values don't blow up exponentially as N grows larger, but rather can be bounded by a polynomial in terms of the number N – not to be confused with the number of bits of N, or simply n.

3.6 ESTIMATE FOR LARGE RSA KEYS

In this section, we will provide an estimate for what our Hamiltonian would look like for large RSA keys, considering numbers with $n = 1024$ and $n = 2048$ bits, or $N = 2^{1024}$ and $N = 2^{2048}$, respectively.

Using our Equations 2.4 for n_x and n_y, we get:

$$\text{For } N = 2^{1024}:$$
$$n_x = 511,$$
$$n_y = 1022.$$

$$\text{For } N = 2^{2048}:$$
$$n_x = 1023,$$
$$n_y = 2046.$$

From this, we can estimate the number of auxiliary variables for each method, given by Theorems 3.1 and 3.2:

For $N = 2^{1024}$:

$$\begin{aligned}
M_{\text{Ishi}} &= \frac{n_x n_y (n_x n_y + n_x + n_y - 3)}{4} \\
&= \frac{511 * 1022(511 * 1022 + 511 + 1022 - 3)}{4}. \\
&= 68383934206 \\
&\approx 6.838 * 10^{10},
\end{aligned}$$

$$\begin{aligned}
M_{\text{Datt}} &= \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{511 * 1022(511 - 1)(1022 - 1)}{4} \\
&= 67984157955 \\
&\approx 6.798 * 10^{10}.
\end{aligned}$$

For $N = 2^{2048}$:

$$\begin{aligned}
M_{\text{Ishi}} &= \frac{n_x n_y (n_x n_y + n_x + n_y - 3)}{4} \\
&= \frac{1023 * 2046(1023 * 2046 + 1023 + 2046 - 3)}{4} \\
&= 1096827276798 \\
&\approx 1.097 * 10^{12},
\end{aligned}$$

$$\begin{aligned}
M_{\text{Datt}} &= \frac{n_x n_y (n_x - 1)(n_y - 1)}{4} \\
&= \frac{1023 * 2046(1023 - 1)(2046 - 1)}{4} \\
&= 1093617572355 \\
&\approx 1.094 * 10^{12}.
\end{aligned}$$

Finally, we compile these results in Table 3.3. Once again, we note that Dattani-Chau's Hamiltonian performs slightly better than Ishikawa's, considering it requires fewer extra variables. But, for these large numbers, the differences are already quite small in comparison.

Table 3.3 – Estimate for number of variables on large RSA keys.

	$\mathbf{n_x}$	$\mathbf{n_y}$	$\mathbf{M_{Ishi}}$	$\mathbf{M_{Datt}}$
$\mathbf{n = 1024}$ **bits**	511	1022	$6.838 * 10^{10}$	$6.798 * 10^{10}$
$\mathbf{n = 2048}$ **bits**	1023	2046	$1.097 * 10^{12}$	$1.094 * 10^{12}$

Source: developed by the author.

At this point, we should say that even though numbers with 1024 and 2048 bits are indeed very large and notably hard to factor, we have not identified anything about the internal structure of our Hamiltonian which would provide an improvement to integer factorization in general. This knowledge, combined with the number of variables estimated in Table 3.3, suggests it is not feasible to use this approach to factor large numbers, since it would require billions (or even trillions) of auxiliary variables to interact with just a few thousand x and y variables.

In reality, we should not expect this arrangement of variables to be possible to implement, since the number of connections to the x-y-core seen in the graphs from Figures 3.1 and 3.2 would be too great to physically arrange.

We argue this limitation is mostly due to the structure of the cost function – meaning $f_N(x, y) = (N - xy)^2$ – that we chose in the beginning of this study. Nonetheless, this formulation was useful for evaluating the two quadratization methods that we sought out to compare. In future works, however, we wish to explore different configurations for the cost function and different methods for encoding the factorization problem into the Hamiltonian operator.

3.7 DISCUSSION

Throughout this chapter, we have discussed different metrics for evaluating each method's quadratized Hamiltonian. Finally, we have compiled the overall results of our analysis, which are presented in Table 3.4.

Table 3.4 – Comparison of quadratization methods.

Feature	**Ishikawa**	**Dattani-Chau**
(1) Lower number of extra variables See Theorems 3.1 and 3.2.	×	✓
(2) Lower number of quadratic terms See Table 3.2.	×	✓
(3) Lower connectivity (beta index) See Figure 3.3.	✓	×
(4) Smaller range of coefficients See Figure 3.4.	×	×

Source: developed by the author.

From these results, we can conclude that Dattani-Chau's method is better at quadratizing the original Hamiltonian using the least amount of extra variables (item 1). As a consequence of this, the resulting Hamiltonian also contains less quadratic monomials (item 2) than the one from Ishikawa's method.

Further on, we have also looked at the topology and connectivity regarding each method's Hamiltonian. Previously, we have shown that Ishikawa's Hamiltonian always presents a smaller beta index (item 3) than Dattani-Chau's. In this context, the former method seems to have an advantage over the latter. However, we have also explained that this feature holds little importance, because Ishikawa's Hamiltonian, when interpreted as a graph, still utilizes more nodes and edges than the other method, and can only be considered the lesser-connected Hamiltonian (meaning the one with the smaller beta index) precisely because it uses more nodes (or qubits) to do so.

Moreover, we have also estimated how the range of coefficients (item 4) in the Hamiltonian grows for each method. We have shown that, at least for reasonably small values of N, the Hamiltonian coefficients tend to stay inside the interval $[-N^3, +N^3]$, and thus seem to be polynomially bounded in relation to the value of N. Other than that, we were not able to identify any significant difference between the two methods' expected range of coefficients, and therefore we have considered this feature a draw.

Finally, with all that was presented, we conclude that Dattani-Chau's quadratization method is overall better than Ishikawa's, considering the Hamiltonian that was the object of analysis and the scope of this study.

Conclusions and Future Work

The main goal of this work was to study how different quadratization methods – specifically the ones proposed by Ishikawa (2011) and Dattani and Chau (2019) –, can affect the Hamiltonian for the integer factorization problem. As of yet, we have provided a general formula for this Hamiltonian, which includes simplifications regarding monomials with repeated binary variables. We have also provided:

1. A second general formula for the factorization Hamiltonian, quadratized via the method proposed by Ishikawa (2011).
2. And a third general formula for the same Hamiltonian, but instead quadratized through the method proposed by Dattani and Chau (2019).

Moreover, we have evaluated different metrics for both quadratization methods, which were then used to compare each methods' cost and effectiveness in simplifying the original Hamiltonian operator. For each case, we have explored:

– How the number of auxiliary variables grows.

– What is the number of resulting quadratic terms in the final expression.

– What is the connectivity between qubits of the Hamiltonian.

– What is the range of coefficients (i.e. coupling strengths) required for the computation.

We have shown that the approach proposed by Dattani and Chau (2019) is better suited for quadratizing our Hamiltonian, since it outperforms Ishikawa's method in almost every aspect we have analysed.

On top of that, we have also made estimates for what our Hamiltonian operator would look like if it were employed to factor large RSA numbers, with 1024 and 2048 bits. We have concluded that our approach would not be physically possible to implement, but that it still serves as a framework for comparing the aforementioned quadratization methods.

In future works, this study wishes to explore alternative ways of encoding the factorization problem into the Hamiltonian operator, aiming at approaches with better scalability that are based in different cost functions, or different multiplication algorithms.

We also intend to explore other quadratization techniques, when applicable, in order to better understand the impacts that any quadratization process has on the Hamiltonian operators that are related to the factorization problem.

Finally, we also wish to perform actual examples of factorization via adiabatic quantum computing, so we can compare how each quadratization method affects the overall performance of the computation. Preferably, this should be done in an actual quantum computer, such as the ones provided by *D-Wave Systems*[1]. Depending on availability, however, this may also be performed using a quantum simulator.

[1] See `https://www.dwavesys.com/`.

References

AGRAWAL, Manindra; KAYAL, Neeraj; SAXENA, Nitin. PRIMES is in P. **Annals of Mathematics**, Annals of Mathematics, v. 160, n. 2, p. 781–793, Sept. 2004. DOI: 10.4007/annals.2004.160.781. Available from: <https://doi.org/10.4007/annals.2004.160.781>. Cited 1 time on page 34.

ALBASH, Tameem; LIDAR, Daniel A. Adiabatic quantum computation. **Reviews of Modern Physics**, American Physical Society, v. 90, 1 Jan. 2018. ISSN 15390756. DOI: 10.1103/RevModPhys.90.015002. Available from: <https://arxiv.org/abs/1611.04471>. Visited on: 13 Jan. 2022. Cited 6 times on pages 23, 26, 30, 31, 95.

BARKER, Elaine B.; DANG, Quynh H. **Recommendation for Key Management Part 3: Application-Specific Key Management Guidance**. [S.l.], Jan. 2015. DOI: 10.6028/nist.sp.800-57pt3r1. Available from: <https://doi.org/10.6028/nist.sp.800-57pt3r1>. Cited 1 time on page 18.

BORN, M.; FOCK, V. Beweis des Adiabatensatzes. **Zeitschrift für Physik**, v. 51, n. 3, p. 165–180, Mar. 1928. ISSN 0044-3328. DOI: 10.1007/BF01343193. Available from: <https://doi.org/10.1007/BF01343193>. Cited 1 time on page 26.

BOROS, Endre; HAMMER, Peter L. Pseudo-Boolean optimization. **Discrete Applied Mathematics**, v. 123, n. 1, p. 155–225, 2002. ISSN 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00341-9.

Available from: <https://www.sciencedirect.com/science/article/pii/S0166218X01003419>. Cited 1 time on page 20.

BOUDOT, F. et al. **Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment**. [S.l.: s.n.], 2020. Cryptology ePrint Archive, Report 2020/697. Available from: <https://ia.cr/2020/697>. Cited 1 time on page 18.

DATTANI, Nike; CHAU, Hou Tin. All 4-variable functions can be perfectly quadratized with only 1 auxiliary variable, Oct. 2019. Available from: <http://arxiv.org/abs/1910.13583>. Cited 14 times on pages 20, 21, 33, 42, 48, 49, 54, 57, 60, 85, 86.

DATTANI, Nikesh S. Quadratization in discrete optimization and quantum mechanics. **ArXiv**, 2019. Available from: <https://arxiv.org/abs/1901.04405>. Cited 2 times on pages 20, 42.

DATTANI, Nikesh S.; BRYANS, Nathaniel. Quantum factorization of 56153 with only 4 qubits. **ArXiv**, 2014. Available from: <https://arxiv.org/abs/1411.6758>. Cited 1 time on page 20.

ELGART, Alexander; HAGEDORN, George A. A note on the switching adiabatic theorem. **Journal of Mathematical Physics**, v. 53, p. 102202, 10 Oct. 2012. ISSN 0022-2488. DOI: 10.1063/1.4748968. Available from: <http://aip.scitation.org/doi/10.1063/1.4748968>. Cited 2 times on pages 30, 31.

FARHI, Edward et al. **Quantum Computation by Adiabatic Evolution**. [S.l.], 2000. arXiv: quant-ph/0001106 [quant-ph]. Cited 1 time on page 20.

FREEDMAN, D.; DRINEAS, P. Energy Minimization via Graph Cuts: Settling What is Possible. IEEE, 2005. DOI: 10.1109/cvpr.2005.143. Available from:

<https://doi.org/10.1109/cvpr.2005.143>. Cited 3 times on pages 20, 43.

GRIFFITHS, David J. **Introduction to Quantum Mechanics (2nd Edition)**. 2nd. [S.l.]: Pearson Prentice Hall, Apr. 2004. Hardcover. ISBN 0131118927. Available from: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0131118927>. Cited 3 times on pages 24, 26, 27.

HEGADE, Narendra N. et al. Digitized Adiabatic Quantum Factorization, May 2021. ISSN 2469-9926. DOI: 10.1103/PhysRevA.104.L050403. Available from: <http://arxiv.org/abs/2105.09480>. Cited 2 times on page 36.

ISHIKAWA, Hiroshi. Transformation of general binary mrf minimization to the first-order case. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 33, p. 1234–1249, 6 2011. ISSN 01628828. DOI: 10.1109/TPAMI.2010.91. Cited 13 times on pages 20, 21, 33, 42, 43, 49, 57, 59, 85.

KATO, Tosio. On the Adiabatic Theorem of Quantum Mechanics. **Journal of the Physical Society of Japan**, v. 5, n. 6, p. 435–439, 1950. DOI: 10.1143/JPSJ.5.435. eprint: https://doi.org/10.1143/JPSJ.5.435. Available from: <https://doi.org/10.1143/JPSJ.5.435>. Cited 1 time on page 26.

KNUTH, Donald E. **The Art of Computer Programming, Volume 2: Seminumerical Algorithms**. Third. Boston: Addison-Wesley, 1997. ISBN 0201896842. Cited 1 time on page 18.

LENSTRA, A. K. et al. The number field sieve. In: LECTURE Notes in Mathematics. [S.l.]: Springer Berlin Heidelberg, 1993. P. 11–42. DOI: 10.1007/bfb0091537. Available from: <https://doi.org/10.1007/bfb0091537>. Cited 1 time on page 17.

MARTÍN-LÓPEZ, Enrique et al. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. **Nature Photonics**, Springer Science and Business Media LLC, v. 6, n. 11, p. 773–776, Oct. 2012. ISSN 1749-4893. DOI: 10.1038/nphoton.2012.259. Available from: <http://dx.doi.org/10.1038/nphoton.2012.259>. Cited 1 time on page 19.

MOSCA, Michele; VERSCHOOR, Sebastian R. Factoring semi-primes with (quantum) SAT-solvers. **CoRR**, abs/1902.01448, 2019. arXiv: 1902.01448. Available from: <http://arxiv.org/abs/1902.01448>. Cited 1 time on page 20.

NIELSEN, Michael A.; CHUANG, Isaac L. **Quantum computation and quantum information**. [S.l.]: Cambridge University Press, 2010. P. 676. ISBN 9781107002173. Cited 10 times on pages 18, 19, 24, 25, 95, 96, 98, 101.

PENG, Xinhua et al. Quantum Adiabatic Algorithm for Factorization and Its Experimental Implementation. **Physical Review Letters**, American Physical Society (APS), v. 101, n. 22, Nov. 2008. DOI: 10.1103/physrevlett.101.220405. Available from: <https://doi.org/10.1103/physrevlett.101.220405>. Cited 3 times on pages 20, 36.

SHOR, P.W. Algorithms for quantum computation: discrete logarithms and factoring. In: p. 124–134. DOI: 10.1109/sfcs.1994.365700. Cited 1 time on page 19.

STALLINGS, William. **Cryptography and Network Security: Principles and Practice**. 6th. USA: Prentice Hall Press, 2013. ISBN 0133354695. Cited 2 times on pages 18, 19.

XU, Nanyang et al. Erratum: Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System [Phys. Rev. Lett.108, 130501 (2012)]. **Physical Review Letters**, American

Physical Society (APS), v. 109, n. 26, Dec. 2012. DOI: 10.1103/physrevlett.109.269902. Available from: <https://doi.org/10.1103/physrevlett.109.269902>. Cited 1 time on page 20.

YIN, Juan et al. Satellite-Based Entanglement Distribution Over 1200 kilometers. **Science**, v. 356, p. 1140–1144, June 2017. DOI: 10.1126/science.aan3211. Cited 1 time on page 106.

Appendix

APPENDIX A

Quantum Information

The idea behind using quantum interactions as an alternative – perhaps more powerful – universal computational model, has been around since the 1980s. Nowadays, the circuit-based model of quantum computing has evolved a lot, and is considered the standard model for quantum computers (ALBASH; LIDAR, 2018).

This appendix serves as an introduction to the standard model of quantum computing, and aims to explain the more basic concepts involved in quantum information. Note that this is done from the perspective of classical computing, so the more fundamental differences between the two models can be explored.

A.1 QUANTUM BITS

In classical computing, a binary digit, or simply a *bit*, is the smallest unit of information, and can be found in exactly one of two binary states: 0 or 1. Similarly, in quantum computing there is the concept of a quantum bit, or *qubit*. The state of a qubit is represented by a two-dimensional unit vector inside the vector space $\mathbb{C}^2$, which is a Hilbert space (NIELSEN; CHUANG, 2010, p. 13).

Just like the classical bit, a quantum bit can be in the binary states represented by the vectors $|0\rangle$ and $|1\rangle$. However, the key difference is that the qubit can also be in states which are linear combinations – or superpositions – of the vectors $|0\rangle$ and $|1\rangle$. In general, the state $|\psi\rangle$ of a qubit can be written as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers, and $|\alpha|^2 + |\beta|^2 = 1$. This is possible because the vectors $|0\rangle$ and $|1\rangle$ correspond to a basis in the vector space $\mathbb{C}^2$, known as the computational basis. In practice, any pair of linearly independent unit vectors serves as a basis in $\mathbb{C}^2$. For simplicity, the computational basis is established as a convention.

A.1.1 Dirac Notation

Conventionally, quantum computing borrows a certain mathematical notation from quantum mechanics. This is called the Dirac Notation, or the *bra-ket* notation, and it is used for algebraically representing quantum states. Here, line vectors are represented by a *bra* $\langle\psi|$, while column vectors are represented by a *ket* $|\psi\rangle$ (NIELSEN; CHUANG, 2010, p. 13).

This precise notation was used in the previous section to write the kets $|0\rangle$ and $|1\rangle$. Their expansions in the usual column vector form are presented next:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, \qquad\qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}.$$

In the same way, we can write any state $|\psi\rangle$ of a single qubit in its usual column form, as can be seen below:

$$|\psi\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}.$$

Moreover, we sometimes need to convert a line vector (bra) into a column vector (ket) or vice-versa. This is achieved via the following relation:

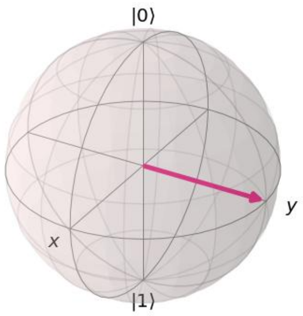
$$\langle\psi| = |\psi\rangle^{\dagger},$$

where the symbol $\dagger$ stands for the complex conjugation and transposition of the vector.

A.1.2 Bloch Sphere

Since the state of a qubit can always be represented by a unit vector in the $\mathbb{C}^2$ space, it is possible to visualize these states as points on a unit sphere, called the Bloch sphere. The computational basis states, $|0\rangle$ and $|1\rangle$, are located at opposite poles of the sphere, as shown in Figure A.1.

Figure A.1 – Bloch Sphere.



Bloch sphere showing an arbitrary vector state. Source: adapted from Qiskit[1].

This graphical representation is interesting, because it allows for operations on a qubit to be represented as rotations around the axes of the sphere.

A.1.3 Measurement and Collapse

An intrinsic property of quantum systems is the uncertainty regarding the actual state of the system – here, quantum system means a collection of one or more qubits. It is not possible to know the exact state of a qubit before measuring it.

The measurement of a qubit is always associated with a collection of *measurement operators*. For instance, we can define the measurement operators $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$, such that they relate to the computational basis. In this sense, it is also possible to say that the action of measuring a qubit is associated with

[1] Available at: `https://qiskit.org/documentation/_images/qiskit.visualization.plot_bloch_vector_0_0.png`. Accessed on: January 2022.

a certain basis – usually the computational basis –, of the vector space, as long as the measurement operators are chosen accordingly (NIELSEN; CHUANG, 2010, p. 84-85).

It is important to note that, after a measurement, the state of the qubit will collapse to another state, related to the operators that were used for measuring it. This means the actual state of a qubit is lost after being measured.

For instance, let $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$ be the current state of a single qubit. In practice, if this qubit were to be measured using the computational basis, its state $|\psi\rangle$ would collapse to either $|0\rangle$ or $|1\rangle$. Therefore, the measurement can be interpreted as an operation which yields a classical bit (0 or 1), depending on which state $|\psi\rangle$ collapses to (NIELSEN; CHUANG, 2010, p. 85).

Moreover, the probability of the measurement resulting in 0 would be $|\alpha|^2$, while the probability of it yielding 1 would be $|\beta|^2$. Note that this relates to the restriction that $|\alpha|^2 + |\beta|^2 = 1$. Either way, after said qubit was measured – and assuming no other operations were applied to it –, all subsequent measurements would yield the same result as the first measurement, since by then the state $|\psi\rangle$ would have already collapsed to either $|0\rangle$ or $|1\rangle$.

A.1.4 Tensor Product

From a computational perspective, a quantum system with a single qubit is not very useful. Consequently, there needs to be a way to represent systems involving many qubits.

The state of a quantum system with many qubits is commonly represented algebraically through the use of the tensor product of individual qubit states. This operation is shown below:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} a_0 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}.$$

Additionally, the tensor product between two or more kets can be abbreviated to the juxtaposition of the states, as shown below – not to be confused with the internal product operation, usually denoted by a dot:

$$|\psi_1\rangle \otimes |\psi_2\rangle = |\psi_1\rangle\,|\psi_2\rangle = |\psi_1\psi_2\rangle\,.$$

Example A.1.1. Consider the tensor products between the states of the computational basis.

$$|0\rangle \otimes |0\rangle = |00\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix}1\\0\end{bmatrix} \\ 0\begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix},$$

$$|0\rangle \otimes |1\rangle = |01\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix}0\\1\end{bmatrix} \\ 0\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix},$$

$$|1\rangle \otimes |0\rangle = |10\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix} 0\begin{bmatrix}1\\0\end{bmatrix} \\ 1\begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix},$$

$$|1\rangle \otimes |1\rangle = |11\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} 0\begin{bmatrix}0\\1\end{bmatrix} \\ 1\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}.$$

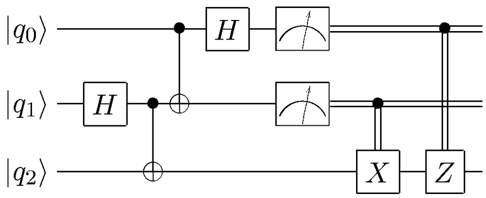
Notice each state $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ is a unit vector in the $(\mathbb{C}^2 \otimes \mathbb{C}^2)$ vector space. In addition to that, notice these states also form a basis – namely, the computational basis for systems of two qubits.

A.2 QUANTUM CIRCUITS

Analogous to logical circuits from classical computing, the standard model for quantum computing is based around quantum circuits, which are built from quantum gates representing operations over one or more qubits. In general, these circuits are described graphically, much like in Figure A.2 and are read left to right.

Each qubit is represented by a single horizontal line, while classical bits are drawn as double horizontal lines. On the other hand, quantum gates are usually drawn as small boxes with symbols, indicating what operation is being performed – although there are some exceptions. Lastly, boxes with a semicircle and an arrow indicate the operation of measuring a qubit, which is then treated as a classical bit – a double horizontal line – that represents the output of the measurement.

Figure A.2 – Example of a quantum circuit.



Example of a quantum circuit with three qubits (labeled q_0, q_1 and q_2) and a few quantum gates, as well as some measurement operations. Source: Imgur[2].

Mathematically, a quantum gate can be described by a unitary matrix, usually called a *unitary operator*.

Definition A.2.1 (Unitary Operator)**.** A matrix U is called unitary if and only if $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ stands for the complex conjugate of the transpose of U, and I is simply the identity matrix.

It is very important to note that many classical logic gates do not have directly corresponding quantum gates. This is due to the fact that the most usual logic gates, such as AND and OR, perform irreversible operations over their inputs – meaning their input can't always be derived entirely from their output. This is a big difference when compared to quantum gates, which are always necessarily reversible by their nature.

In practice, all operations involved in a quantum computation are reversible, except for the measuring of a qubit, which causes the state to collapse. This reversibility restriction implies that all

[2] Available at: https://i.stack.imgur.com/AYfEA.png. Accessed on: January 2022.

quantum gates must have the same number of inputs and outputs, or else they simply could not be reversible.

In addition to that, another restriction imposed by quantum mechanics is that an unknown qubit state can never be copied over to another qubit – i.e., duplicated. This is supported by the *No-cloning Theorem* (NIELSEN; CHUANG, 2010, p. 134).

A.2.1 Single Qubit Quantum Gates

Single qubit quantum gates are represented by 2×2 unitary matrices. This section aims to present a few of the most usual gates from this category.

A.2.1.1 Pauli Matrices

Among the most common quantum gates are the so-called Pauli matrices. They are represented by the letters X, Y and Z, as shown below:

$$X = \sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

$$Y = \sigma^y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix},$$

$$Z = \sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Regarding the Bloch sphere, discussed before, the effect a Pauli matrix has over a qubit state can be view as an 180-degree rotation around a particular axis. For instance, the matrix X, when applied to a given state $|\psi\rangle$, rotates the state vector 180 degrees around the x-axis of the Bloch sphere. The same applies for the Y and Z matrices, in regard to their respective axes.

Furthermore, it is easy to show algebraically the effect of an X gate over the states $|0\rangle$ and $|1\rangle$. This gate is sometimes called a quantum NOT gate, because its effect on the computational basis states is equivalent to a classical logic NOT gate.

$$\begin{aligned} X\left|0\right\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \left|1\right\rangle, \\ X\left|1\right\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \left|0\right\rangle. \end{aligned}$$

In the same way, we can define the behavior of the Y and Z operators over the computational basis states.

$$\begin{aligned} Y\left|0\right\rangle &= i\left|1\right\rangle, \\ Y\left|1\right\rangle &= -i\left|0\right\rangle, \end{aligned}$$

$$\begin{aligned} Z\left|0\right\rangle &= \left|0\right\rangle, \\ Z\left|1\right\rangle &= -\left|1\right\rangle. \end{aligned}$$

A.2.1.2 Hadamard Gate

Another very important gate is the Hadamard gate, represented by the letter H:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

This gate is equivalent to a change of basis from the computational basis $\{\left|0\right\rangle, \left|1\right\rangle\}$ to the $\{\left|+\right\rangle, \left|-\right\rangle\}$ basis, and vice-versa.

$$\begin{aligned} H\left|0\right\rangle &= \left|+\right\rangle, & H\left|+\right\rangle &= \left|0\right\rangle, \\ H\left|1\right\rangle &= \left|-\right\rangle, & H\left|-\right\rangle &= \left|1\right\rangle. \end{aligned}$$

Additionally, the states $\left|+\right\rangle$ and $\left|-\right\rangle$ are also presented below:

$$\begin{aligned} \left|+\right\rangle &= \frac{\left|0\right\rangle + \left|1\right\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\ \left|-\right\rangle &= \frac{\left|0\right\rangle - \left|1\right\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \end{aligned}$$

A.2.2 Two Qubit Quantum Gates

Double qubit quantum gates are represented by 4×4 unitary matrices. This section describes some common gates in this group.

A.2.2.1 CNOT Gate

The controlled NOT gate, or simply CNOT gate, acts on two qubits. Conventionally, the first qubit is said to be the control, whilst the second qubit is the target. The behavior of this gate depends on the value of the control qubit. If it is $|0\rangle$, nothing happens. But, if it is $|1\rangle$, the X Pauli matrix is applied to the target qubit.

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Where we have:

$$\begin{aligned} \mathrm{CNOT}\,|0\rangle\,|0\rangle &= \mathrm{CNOT}\,|00\rangle = |00\rangle\,, \\ \mathrm{CNOT}\,|0\rangle\,|1\rangle &= \mathrm{CNOT}\,|01\rangle = |01\rangle\,, \\ \mathrm{CNOT}\,|1\rangle\,|0\rangle &= \mathrm{CNOT}\,|10\rangle = |11\rangle\,, \\ \mathrm{CNOT}\,|1\rangle\,|1\rangle &= \mathrm{CNOT}\,|11\rangle = |10\rangle\,. \end{aligned}$$

The concept of a gate being controlled by one or more qubits is well-known, and widely used across quantum circuits, because it can be applied to virtually any quantum gate.

A.2.2.2 SWAP Gate

Another two qubit gate that is commonly used is the SWAP gate, which swaps around two qubits. This gate is used implicitly in classical logic circuits, just by rearranging the electric wires that carry each classical bit:

$$\mathrm{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Where we have:

$$\begin{aligned}
\text{SWAP}\,|0\rangle\,|0\rangle &= \text{SWAP}\,|00\rangle = |00\rangle\,, \\
\text{SWAP}\,|0\rangle\,|1\rangle &= \text{SWAP}\,|01\rangle = |10\rangle\,, \\
\text{SWAP}\,|1\rangle\,|0\rangle &= \text{SWAP}\,|10\rangle = |01\rangle\,, \\
\text{SWAP}\,|1\rangle\,|1\rangle &= \text{SWAP}\,|11\rangle = |11\rangle\,.
\end{aligned}$$

A.3 ENTANGLEMENT

So far, we have only talked about multiple qubit states that are written as linear combinations of states from a certain basis – usually the computational basis. For instance, imagine a quantum system with two qubits $|\psi\rangle$ and $|\phi\rangle$, such that:

$$\begin{aligned}
|\psi\rangle &= \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \\
|\phi\rangle &= \frac{1}{\sqrt{3}}\,|0\rangle + \frac{2}{\sqrt{3}}\,|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{3}} \end{bmatrix},
\end{aligned}$$

then, the state of the whole system can be described as the tensor product between $|\psi\rangle$ and $|\phi\rangle$, like this:

$$|\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{3}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{3}} \end{bmatrix} \\ \frac{1}{\sqrt{2}} & \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{3}} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{6}} \end{bmatrix}.$$

In this case, the state of the system is said to be separable, for it can be decomposed into the tensor product between the individual states of each qubit in the system – namely $|\psi\rangle$ and $|\phi\rangle$ in the above example.

However, there are certain situations in which the state of the whole system simply cannot be separated nicely into individual single qubit states. For example, let $|\psi\rangle$ be the state of a system with two qubits, such that:

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} .$$

Well, if $|\psi\rangle$ were to be separable, then it could be written as follows:

$$\begin{aligned} |\psi\rangle &= (a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle) \\ &= ac\,|00\rangle + ad\,|01\rangle + bc\,|10\rangle + bd\,|11\rangle \,, \end{aligned}$$

which would imply the following system of equations has a solution.

$$\begin{cases} ac &= \frac{1}{\sqrt{2}} \,, & \text{(A.1)} \\ ad &= 0 \,, & \text{(A.2)} \\ bc &= 0 \,, & \text{(A.3)} \\ bd &= \frac{1}{\sqrt{2}} \,. & \text{(A.4)} \end{cases}$$

Yet, it is easy to show that such solution does not exist. Note that, in Equation A.2, at least one of the coefficients a and d must equal zero. Either way, this would contradict one (or both) of the Equations A.1 and A.4.

In that sense, in the example above, $|\psi\rangle$ is called an entangled state. Entanglement is a property that is exclusive to quantum mechanics, and does not have a corresponding feature in classical computing.

Practically speaking, when two (or more) qubits are entangled, their states depend on one another. This means that if one entangled qubit is measured and collapses, the other (others) will simultaneously collapse as well. This instantaneous update is believed to always take place, no matter how far apart the qubits are from each other – recent experiments demonstrated entanglement between pairs of qubits over 1200 kilometers apart, which was achieved with the help of a satellite in orbit (YIN et al., 2017).

APPENDIX B

Proof for the equality of Hamiltonian formulas

This appendix provides a complete proof for the equality of the Hamiltonians from Equations 2.16 and 2.22. Let us refer to the former as the original Hamiltonian, or simply H_N^{O}, and to the latter as the alternate Hamiltonian formula, or H_N^{A}. Thus, we want to show that, for every N greater or equal to some constant K, the following equation holds:

$$H_N^{\mathrm{O}} = H_N^{\mathrm{A}}, \text{ where } N \geq K \text{ for some } K, \tag{B.1}$$

in which, of course, we have:

$$\begin{aligned} H_N^{\mathrm{O}} = & \sum_{\substack{1 \leq i < j \leq n_x \\ 1 \leq k < l \leq n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l \\ & + \sum_{\substack{1 \leq i < j \leq n_x \\ 1 \leq k \leq n_y}} (2^{i+j+2k+1} + 2^{i+j+k+2}) x_i x_j y_k \\ & + \sum_{\substack{1 \leq i \leq n_x \\ 1 \leq k < l \leq n_y}} (2^{2i+k+l+1} + 2^{i+k+l+2}) x_i y_k y_l \\ & + (\textit{original terms of degree less than 3}), \end{aligned} \tag{B.2}$$

and:

$$\begin{aligned} H_N^{\mathrm{A}} &= f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \\ &\quad + (\textit{original terms of degree less than 3}), \end{aligned} \tag{B.3}$$

where each f_i is defined as described in Table 2.1.

This equality will be proven by construction. First, let us define three expressions for the variables A, B and C as follows:

$$\begin{aligned} A_{i,j,k,l} &= 2^{i+j+k+l+2} x_i x_j y_k y_l \\ B_{i,j,k} &= (2^{i+j+2k+1} + 2^{i+j+k+2}) x_i x_j y_k \\ C_{i,k,l} &= (2^{2i+k+l+1} + 2^{i+k+l+2}) x_i y_k y_l, \end{aligned}$$

such that we can rewrite H_N^{O} from Equation B.2 as simply:

$$\begin{aligned} H_N^{\mathrm{O}} = &\sum_{\substack{1 \le i < j \le n_x \\ 1 \le k < l \le n_y}} A_{i,j,k,l} + \sum_{\substack{1 \le i < j \le n_x \\ 1 \le k \le n_y}} B_{i,j,k} + \sum_{\substack{1 \le i \le n_x \\ 1 \le k < l \le n_y}} C_{i,k,l} \\ &+ (\textit{original terms of degree less than 3}). \end{aligned} \tag{B.4}$$

Now, let us also rewrite each function f_i from Equation B.3 in terms of A, B and C, following the same groupings that were established in Table 2.1:

$$\begin{aligned} f_1 &= A_{1,2,1,2} + B_{1,2,1} + B_{1,2,2} + C_{1,1,2} + C_{2,1,2} \\ f_2 &= \sum_{3 \le l \le n_y} A_{1,2,1,l} + B_{1,2,l} + C_{1,1,l} + C_{2,1,l} \\ f_3 &= \sum_{3 \le j \le n_x} A_{1,j,1,2} + B_{1,j,1} + B_{1,j,2} + C_{j,1,2} \\ f_4 &= \sum_{\substack{3 \le j \le n_x \\ 3 \le l \le n_y}} A_{1,j,1,l} + B_{1,j,l} + C_{j,1,l} \\ f_5 &= \sum_{2 \le k < l \le n_y} A_{1,2,k,l} + C_{1,k,l} + C_{2,k,l} \\ f_6 &= \sum_{2 \le i < j \le n_x} A_{i,j,1,2} + B_{i,j,1} + B_{i,j,2} \end{aligned}$$

$$f_7 = \sum_{\substack{3 \le j \le n_x \\ 2 \le k < l \le n_y}} A_{1,j,k,l} + C_{j,k,l}$$

$$f_8 = \sum_{\substack{2 \le i < j \le n_x \\ 3 \le l \le n_y}} A_{i,j,1,l} + B_{i,j,l}$$

$$f_9 = \sum_{\substack{2 \le i < j \le n_x \\ 2 \le k < l \le n_y}} A_{i,j,k,l}.$$

From this, it is possible to prove Equation B.1 by simply rearranging the terms from the right side of the equation, in order for them to match the left side. To achieve this, let us first sum all the functions f_i explicitly:

$$\begin{aligned}
H_N^{\mathrm{A}} &= f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \\
&\quad + (\textit{original terms of degree less than 3}) \\
H_N^{\mathrm{A}} &= A_{1,2,1,2} + B_{1,2,1} + B_{1,2,2} + C_{1,1,2} + C_{2,1,2} \\
&\quad + \sum_{3 \le l \le n_y} A_{1,2,1,l} + B_{1,2,l} + C_{1,1,l} + C_{2,1,l} \\
&\quad + \sum_{3 \le j \le n_x} A_{1,j,1,2} + B_{1,j,1} + B_{1,j,2} + C_{j,1,2} \\
&\quad + \sum_{\substack{3 \le j \le n_x \\ 3 \le l \le n_y}} A_{1,j,1,l} + B_{1,j,l} + C_{j,1,l} \\
&\quad + \sum_{2 \le k < l \le n_y} A_{1,2,k,l} + C_{1,k,l} + C_{2,k,l} \\
&\quad + \sum_{2 \le i < j \le n_x} A_{i,j,1,2} + B_{i,j,1} + B_{i,j,2} \\
&\quad + \sum_{\substack{3 \le j \le n_x \\ 2 \le k < l \le n_y}} A_{1,j,k,l} + C_{j,k,l} \\
&\quad + \sum_{\substack{2 \le i < j \le n_x \\ 3 \le l \le n_y}} A_{i,j,1,l} + B_{i,j,l} \\
&\quad + \sum_{\substack{2 \le i < j \le n_x \\ 2 \le k < l \le n_y}} A_{i,j,k,l} \\
&\quad + (\textit{original terms of degree less than 3}).
\end{aligned} \tag{B.5}$$

Moreover, let us also rearrange its terms in order to separate the As, Bs and Cs from each other, while respecting the summations and their intervals:

$$\begin{aligned}
H_N^{\mathrm{A}} = A_{1,2,1,2} &+ \sum_{3\leq l\leq n_y} A_{1,2,1,l} + \sum_{3\leq j\leq n_x} A_{1,j,1,2} \\
&+ \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} A_{1,j,1,l} + \sum_{2\leq k<l\leq n_y} A_{1,2,k,l} + \sum_{2\leq i<j\leq n_x} A_{i,j,1,2} \\
&+ \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} A_{1,j,k,l} + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} A_{i,j,1,l} + \sum_{\substack{2\leq i<j\leq n_x\\ 2\leq k<l\leq n_y}} A_{i,j,k,l} \\
&+ B_{1,2,1} + B_{1,2,2} + \sum_{3\leq l\leq n_y} B_{1,2,l} + \sum_{3\leq j\leq n_x} (B_{1,j,1} + B_{1,j,2}) \\
&+ \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} B_{1,j,l} + \sum_{2\leq i<j\leq n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l} \\
&+ C_{1,1,2} + C_{2,1,2} + \sum_{3\leq l\leq n_y} (C_{1,1,l} + C_{2,1,l}) + \sum_{3\leq j\leq n_x} C_{j,1,2} \\
&+ \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l} + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l} \\
&+ (\textit{original terms of degree less than 3}).
\end{aligned} \tag{B.6}$$

Further on, we can now deal with each collection of As, Bs and Cs separately, whilst trying to write them in a more compact expression – similar to what can be seen in Equation B.4. To do so, let us once again rewrite Equation B.6, such that:

$$\begin{aligned}
H_N^{\mathrm{A}} &= \Lambda_A + \Lambda_B + \Lambda_C \\
&\qquad + (\textit{original terms of degree less than 3}),
\end{aligned} \tag{B.7}$$

where:

$$\begin{aligned}
\Lambda_A = {} & A_{1,2,1,2} + \sum_{3\le l\le n_y} A_{1,2,1,l} + \sum_{3\le j\le n_x} A_{1,j,1,2} \\
& + \sum_{\substack{3\le j\le n_x\\ 3\le l\le n_y}} A_{1,j,1,l} + \sum_{2\le k<l\le n_y} A_{1,2,k,l} + \sum_{2\le i<j\le n_x} A_{i,j,1,2} \\
& + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l} + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l} + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l} \\
\Lambda_B = {} & B_{1,2,1} + B_{1,2,2} + \sum_{3\le l\le n_y} B_{1,2,l} + \sum_{3\le j\le n_x} (B_{1,j,1} + B_{1,j,2}) \\
& + \sum_{\substack{3\le j\le n_x\\ 3\le l\le n_y}} B_{1,j,l} + \sum_{2\le i<j\le n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} B_{i,j,l} \\
\Lambda_C = {} & C_{1,1,2} + C_{2,1,2} + \sum_{3\le l\le n_y} (C_{1,1,l} + C_{2,1,l}) + \sum_{3\le j\le n_x} C_{j,1,2} \\
& + \sum_{\substack{3\le j\le n_x\\ 3\le l\le n_y}} C_{j,1,l} + \sum_{2\le k<l\le n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} C_{j,k,l}.
\end{aligned}$$

Notice how Λ_A only contains A terms and so on. Let us now focus on each Λ individually, in order to concatenate their summations intervals, thus rewriting each of them with a single summation. Let us begin with Λ_A:

$$\begin{aligned}
\Lambda_A = {} & \boxed{A_{1,2,1,2} + \sum_{3\le l\le n_y} A_{1,2,1,l}} + \sum_{3\le j\le n_x} A_{1,j,1,2} \\
& + \sum_{\substack{3\le j\le n_x\\ 3\le l\le n_y}} A_{1,j,1,l} + \sum_{2\le k<l\le n_y} A_{1,2,k,l} + \sum_{2\le i<j\le n_x} A_{i,j,1,2} \\
& + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l} + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l} + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l}
\end{aligned}$$

$$\begin{aligned}\Lambda_A = {} & \boxed{\sum_{2\le l\le n_y} A_{1,2,1,l}} + \boxed{\sum_{3\le j\le n_x} A_{1,j,1,2} + \sum_{\substack{3\le j\le n_x\\ 3\le l\le n_y}} A_{1,j,1,l}} \\ & + \sum_{2\le k<l\le n_y} A_{1,2,k,l} + \sum_{2\le i<j\le n_x} A_{i,j,1,2} + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l} \\ & + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l} + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_A = {} & \boxed{\sum_{2\le l\le n_y} A_{1,2,1,l} + \boxed{\sum_{\substack{3\le j\le n_x\\ 2\le l\le n_y}} A_{1,j,1,l}}} + \sum_{2\le k<l\le n_y} A_{1,2,k,l} \\ & + \sum_{2\le i<j\le n_x} A_{i,j,1,2} + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l} + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l} \\ & + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_A = {} & \boxed{\sum_{\substack{2\le j\le n_x\\ 2\le l\le n_y}} A_{1,j,1,l}} + \boxed{\sum_{2\le k<l\le n_y} A_{1,2,k,l}} + \sum_{2\le i<j\le n_x} A_{i,j,1,2} \\ & + \boxed{\sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l}} + \sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l} + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_A = {} & \sum_{\substack{2\le j\le n_x\\ 2\le l\le n_y}} A_{1,j,1,l} + \boxed{\sum_{2\le i<j\le n_x} A_{i,j,1,2}} + \boxed{\sum_{\substack{2\le j\le n_x\\ 2\le k<l\le n_y}} A_{1,j,k,l}} \\ & + \boxed{\sum_{\substack{2\le i<j\le n_x\\ 3\le l\le n_y}} A_{i,j,1,l}} + \sum_{\substack{2\le i<j\le n_x\\ 2\le k<l\le n_y}} A_{i,j,k,l}\end{aligned}$$

$$\Lambda_A = \boxed{\sum_{\substack{2\le j\le n_x\\2\le l\le n_y}} A_{1,j,1,l}} + \boxed{\sum_{\substack{2\le j\le n_x\\2\le k<l\le n_y}} A_{1,j,k,l}} + \boxed{\boxed{\sum_{\substack{2\le i<j\le n_x\\2\le l\le n_y}} A_{i,j,1,l}}}$$
$$+ \sum_{\substack{2\le i<j\le n_x\\2\le k<l\le n_y}} A_{i,j,k,l}$$

$$\Lambda_A = \boxed{\sum_{\substack{i=1\\2\le j\le n_x\\k=1\\2\le l\le n_y}} A_{i,j,k,l}} + \boxed{\sum_{\substack{i=1\\2\le j\le n_x\\2\le k<l\le n_y}} A_{i,j,k,l}} + \boxed{\sum_{\substack{2\le i<j\le n_x\\k=1\\2\le l\le n_y}} A_{i,j,k,l}}$$
$$+ \sum_{\substack{2\le i<j\le n_x\\2\le k<l\le n_y}} A_{i,j,k,l}$$

$$\Lambda_A = \sum_{\substack{1\le i<j\le n_x\\1\le k<l\le n_y}} A_{i,j,k,l}.$$

This way we have condensed the long form of Λ_A into a more concise expression. The same can be achieved for Λ_B, in the following manner:

$$\Lambda_B = \boxed{B_{1,2,1} + B_{1,2,2}} + \sum_{3\le l\le n_y} B_{1,2,l} + \sum_{3\le j\le n_x} (B_{1,j,1} + B_{1,j,2})$$
$$+ \sum_{\substack{3\le j\le n_x\\3\le l\le n_y}} B_{1,j,l} + \sum_{2\le i<j\le n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\le i<j\le n_x\\3\le l\le n_y}} B_{i,j,l}$$

$$\Lambda_B = \boxed{\boxed{\sum_{1\le l\le 2} B_{1,2,l}} + \sum_{3\le l\le n_y} B_{1,2,l}} + \sum_{3\le j\le n_x} (B_{1,j,1} + B_{1,j,2})$$
$$+ \sum_{\substack{3\le j\le n_x\\3\le l\le n_y}} B_{1,j,l} + \sum_{2\le i<j\le n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\le i<j\le n_x\\3\le l\le n_y}} B_{i,j,l}$$

$$\begin{aligned}\Lambda_B = {} & \boxed{\sum_{1\leq l\leq n_y} B_{1,2,l}} + \boxed{\sum_{3\leq j\leq n_x} (B_{1,j,1} + B_{1,j,2})} + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} B_{1,j,l} \\ & + \sum_{2\leq i<j\leq n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l}\end{aligned}$$

$$\begin{aligned}\Lambda_B = {} & \sum_{1\leq l\leq n_y} B_{1,2,l} + \boxed{\boxed{\sum_{\substack{3\leq j\leq n_x\\ 1\leq l\leq 2}} B_{1,j,l}} + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} B_{1,j,l}} \\ & + \sum_{2\leq i<j\leq n_x} (B_{i,j,1} + B_{i,j,2}) + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l}\end{aligned}$$

$$\begin{aligned}\Lambda_B = {} & \boxed{\sum_{1\leq l\leq n_y} B_{1,2,l} + \boxed{\sum_{\substack{3\leq j\leq n_x\\ 1\leq l\leq n_y}} B_{1,j,l}}} + \sum_{2\leq i<j\leq n_x} (B_{i,j,1} + B_{i,j,2}) \\ & + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l}\end{aligned}$$

$$\Lambda_B = \boxed{\sum_{\substack{2\leq j\leq n_x\\ 1\leq l\leq n_y}} B_{1,j,l}} + \boxed{\sum_{2\leq i<j\leq n_x} (B_{i,j,1} + B_{i,j,2})} + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l}$$

$$\Lambda_B = \sum_{\substack{2\leq j\leq n_x\\ 1\leq l\leq n_y}} B_{1,j,l} + \boxed{\boxed{\sum_{\substack{2\leq i<j\leq n_x\\ 1\leq l\leq 2}} B_{i,j,l}} + \sum_{\substack{2\leq i<j\leq n_x\\ 3\leq l\leq n_y}} B_{i,j,l}}$$

$$\Lambda_B = \boxed{\sum_{\substack{2\leq j\leq n_x\\ 1\leq l\leq n_y}} B_{1,j,l}} + \boxed{\sum_{\substack{2\leq i<j\leq n_x\\ 1\leq l\leq n_y}} B_{i,j,l}}$$

$$\Lambda_B = \boxed{\sum_{\substack{i=1\\ 2\leq j\leq n_x\\ 1\leq l\leq n_y}} B_{i,j,l}} + \sum_{\substack{2\leq i<j\leq n_x\\ 1\leq l\leq n_y}} B_{i,j,l}$$

$$\Lambda_B = \sum_{\substack{1\leq i<j\leq n_x\\ 1\leq l\leq n_y}} B_{i,j,l}.$$

At last, this process can also be done to Λ_C, in a very similar way, such that we can rewrite it like so:

$$\begin{aligned}\Lambda_C = {} & \boxed{C_{1,1,2} + C_{2,1,2}} + \sum_{3\leq l\leq n_y} (C_{1,1,l} + C_{2,1,l}) + \sum_{3\leq j\leq n_x} C_{j,1,2} \\ & + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l} + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_C = {} & \boxed{\boxed{\sum_{1\leq j\leq 2} C_{j,1,2}}} + \sum_{3\leq l\leq n_y} (C_{1,1,l} + C_{2,1,l}) + \boxed{\sum_{3\leq j\leq n_x} C_{j,1,2}} \\ & + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l} + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_C = {} & \boxed{\sum_{1\leq j\leq n_x} C_{j,1,2}} + \boxed{\sum_{3\leq l\leq n_y} (C_{1,1,l} + C_{2,1,l})} + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l} \\ & + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_C = {} & \sum_{1\leq j\leq n_x} C_{j,1,2} + \boxed{\boxed{\sum_{\substack{1\leq j\leq 2\\ 3\leq l\leq n_y}} C_{j,1,l}} + \sum_{\substack{3\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l}} \\ & + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}\end{aligned}$$

$$\begin{aligned}\Lambda_C = {} & \boxed{\sum_{1\leq j\leq n_x} C_{j,1,2} + \boxed{\sum_{\substack{1\leq j\leq n_x\\ 3\leq l\leq n_y}} C_{j,1,l}}} + \sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l}) \\ & + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}\end{aligned}$$

$$\Lambda_C = \boxed{\sum_{\substack{1\leq j\leq n_x\\ 2\leq l\leq n_y}} C_{j,1,l}} + \boxed{\sum_{2\leq k<l\leq n_y} (C_{1,k,l} + C_{2,k,l})} + \sum_{\substack{3\leq j\leq n_x\\ 2\leq k<l\leq n_y}} C_{j,k,l}$$

$$\Lambda_C = \sum_{\substack{1\le j\le n_x\\ 2\le l\le n_y}} C_{j,1,l} + \boxed{\boxed{\sum_{\substack{1\le j\le 2\\ 2\le k<l\le n_y}} C_{j,k,l}} + \sum_{\substack{3\le j\le n_x\\ 2\le k<l\le n_y}} C_{j,k,l}}$$

$$\Lambda_C = \boxed{\sum_{\substack{1\le j\le n_x\\ 2\le l\le n_y}} C_{j,1,l}} + \boxed{\sum_{\substack{1\le j\le n_x\\ 2\le k<l\le n_y}} C_{j,k,l}}$$

$$\Lambda_C = \boxed{\sum_{\substack{1\le j\le n_x\\ k=1\\ 2\le l\le n_y}} C_{j,k,l}} + \sum_{\substack{1\le j\le n_x\\ 2\le k<l\le n_y}} C_{j,k,l}$$

$$\Lambda_C = \sum_{\substack{1\le j\le n_x\\ 1\le k<l\le n_y}} C_{j,k,l}.$$

Finally, we have shown how to condense each Λ into a single summation of terms A, B and C. Now, we can rewrite Equation B.7 using what we have established:

$$\begin{aligned} H_N^{\mathrm{A}} &= \Lambda_A + \Lambda_B + \Lambda_C \\ &\quad + (\textit{original terms of degree less than 3}) \\ H_N^{\mathrm{A}} &= \sum_{\substack{1\le i<j\le n_x\\ 1\le k<l\le n_y}} A_{i,j,k,l} + \sum_{\substack{1\le i<j\le n_x\\ 1\le l\le n_y}} B_{i,j,l} + \sum_{\substack{1\le j\le n_x\\ 1\le k<l\le n_y}} C_{j,k,l} \qquad \text{(B.8)} \\ &\quad + (\textit{original terms of degree less than 3}). \end{aligned}$$

But notice that Equation B.8 contains exactly the same expression as Equation B.4 – except for the name of some indexes, which could easily be renamed. This means that H_N^{A} does in fact equal H_N^{O}, which proves our conjecture from Equation B.1.

By now we should address the requisite that N be greater or equal to some constant K. We know from Equation 2.22 that the formula for H_N^{A} only makes sense when the Hamiltonian involves at least the variables x_1, x_2, y_1 and y_2. In other words, we should only use this formula if $n_x \ge 2$ and $n_y \ge 2$. From our previous definition of n_x and n_y in Equations 2.4, we recall that both of these values depend on N. Moreover, it is not hard to check that $N = 25$ is the first integer for which both n_x and n_y are at least equal to two:

$$\begin{aligned}
\text{Let } N = 25&: \\
n_x &= \lceil \log_2 \lfloor \sqrt{N} \rfloor_{odd} \rceil - 1 \\
&= \lceil \log_2 \lfloor \sqrt{25} \rfloor_{odd} \rceil - 1 \\
&= \lceil \log_2 \lfloor 5 \rfloor_{odd} \rceil - 1 \\
&= \lceil \log_2 (5) \rceil - 1 \\
&= \lceil 2.32 \rceil - 1 \\
&= 3 - 1 \\
&= 2,
\end{aligned}$$

$$\begin{aligned}
n_y &= \lceil \log_2 \lfloor \frac{N}{3} \rfloor \rceil - 1 \\
&= \lceil \log_2 \lfloor \frac{25}{3} \rfloor \rceil - 1 \\
&= \lceil \log_2 \lfloor 8.33 \rfloor \rceil - 1 \\
&= \lceil \log_2 (8) \rceil - 1 \\
&= \lceil 3 \rceil - 1 \\
&= 3 - 1 \\
&= 2.
\end{aligned}$$

Therefore, we fix our constant $K = 25$ and thus we expect our conjecture from Equation B.1 to be true as long as $N \geq 25$.

This completes our proof.

Annex

ANNEX A

Article about this study

Comparison of quadratization methods for integer factorization via adiabatic quantum computing

Gilson T. Magro[1]

[1] Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Postcode 88040-370 – Florianópolis – SC – Brazil

gilson.magro@grad.ufsc.br

Abstract. *Adiabatic quantum computing (AQC) has been studied as an alternative for the circuit-based quantum computing model, specially regarding optimization problems. Recent studies have shown the applicability of AQC in the context of integer factorization. We have provided a general formula for a Hamiltonian operator, which encodes the solution to the integer factorization problem. This formula includes Boolean simplifications, followed by a Hamiltonian quadratization via two different methods. We also present a comparison between these two quadratization methods, considering metrics such as the number of auxiliary variables required and the range of coefficients in the resulting Hamiltonians.*

Keywords: *Integer factorization. Adiabatic quantum computing. Hamiltonian quadratization. Pseudo-Boolean functions.*

1. Introduction

The integer factorization problem poses the question: given a positive integer N, what are its prime factors? At first, this might seem like an easy question to answer. Naively, one may simply attempt to check every number between 1 and N, and discover which of them divides N perfectly. In fact, this is fine for small numbers, since we don't have to check many values before finding a prime factor for N.

The problem arises when N is a very large number, with hundreds, if not thousands of bits. It is easy to see that, as the number of bits n increases[1], the search space grows at a rate of about 2^n. Thus, the naive approach of checking every number becomes rapidly intractable. On the other hand, if we take a less naive path, one might consider using the General Number Field Sieve (GNFS) algorithm, proposed by [Lenstra et al. 1993], which is the best known classical algorithm for factoring integers. Still, this algorithm only provides a sub-exponential runtime.

Surprisingly, though, and despite its importance, the exact complexity of the integer factorization problem remains unknown [Knuth 1997]. It is therefore possible – although unlikely – that an efficient classical algorithm, capable of factoring integers in polynomial time, is still waiting to be discovered.

[1]Throughout this study, we will always refer to N as the number being factored, and n as the number of bits of N. Notice that $n \approx \log_2 N$.

For now, though, since no one has ever found an efficient classical algorithm for factorization, it is believed that finding the factors of a number is a hard problem for classical computers to solve, in general. It also turns out that this assumed difficulty on trying to factor large numbers can be taken advantage of by cryptography schemes, such as the RSA (Rivest-Shamir-Adleman) cryptosystem. The RSA cryptosystem is a public-key cryptography scheme widely used around the world, for secure data transmission and digital signatures [Nielsen and Chuang 2010, p. 11].

The RSA scheme involves a pair of public and private keys. These keys are generated from a large number N, which is the product of two large primes p and q, of about the same size. The security provided by RSA comes from the fact that it is computationally very difficult to derive a private key from someone's public key, unless one knows the factorization of $N = pq$. Thus, the advent of a new, more powerful factoring algorithm is of great interest, because it poses a threat to such cryptosystems [Stallings 2013, p. 272-275].

Usually, minor advances in factorization can be overcome by RSA, by simply choosing larger key-sizes (i.e., choosing a larger N). Since 2015, the *National Institute of Standards and Technology* (NIST) recommends key-sizes of at least 2048 bits for RSA, which are considered secure nowadays [Barker and Dang 2015]. Currently, the world record for largest RSA number ever factored is known as RSA-250, which has 250 decimal digits (or 829 bits). It was factored in February 2020 using the GNFS algorithm, and it took researchers almost 2700 CPU core-years[2] of processing in a classical computer [Boudot et al. 2020].

Unfortunately, as the current size of transistors approaches a physical limit, and quantum effects start interfering with the behavior of conventional electronic circuits, the growth rate of classical computing power predicted by Moore's Law[3] has decreased. Therefore, it seems unlikely for the near future that RSA schemes should be threatened solely by classical computers.

In that sense, quantum computing is being studied as a model of computation alternative to classical computing. It is believed to be impossible for a classical computer to efficiently simulate a quantum computer. Thus, the quantum paradigm seems to offer a welcome advantage when compared to classical computing. In fact, there are certain quantum algorithms whose efficiency overcomes that of any known classical algorithm. Perhaps the most famous example of this is the algorithm proposed by [Shor 1994], which is capable of factoring a number N in a polynomial number of steps[4], using a (circuit-based) quantum computer. Unfortunately, the current technology of quantum circuits does not allow for any meaningful, practical implementation of Shor's algorithm, because today's noisy intermediate-scale quantum (NISQ) computers simply do not offer a significant amount of error-corrected qubits.

The current record for largest number ever factored using Shor's algorithm, in an actual quantum computer, was achieved in 2012. The number factored in that occasion

[2]Using the 2.1GHz Intel Xeon Gold 6130 CPU as a reference.

[3]Moore's Law, formulated in 1965 by Gordon Moore, stated that the computational power of classical computers would double regularly every two years. According to [Nielsen and Chuang 2010], this prediction remained approximately correct from the 1960s until the early 2000s.

[4]Regarding the number of bits n of the input number.

was $N = 21$, which is obviously very small [Martín-López et al. 2012]. Nonetheless, this result goes to show that efficient factorization through quantum computing may one day be a reality. This is a relevant topic, because, as mentioned before, being able to efficiently factor large integers makes it possible to efficiently break certain types of encryption. Hence, it is speculated that the dawn of a powerful quantum computer could lead to Internet security issues in the future [Stallings 2013].

Meanwhile, as the notion of circuit-based quantum computers grows in popularity – so much as to have become the standard model of quantum computation –, other approaches to quantum computing are also being developed. An example of that is the idea of using adiabatic evolution as the means for achieving quantum computation. This paradigm is commonly referred to as adiabatic quantum computing, or AQC for short.

Initially proposed by [Farhi et al. 2000], in the context of quantum computing via adiabatic evolution, a given problem is encoded in the construction of a quantum operator, called the Hamiltonian operator. The quantum system is then evolved adiabatically – that is, slowly – to match the ground state of said Hamiltonian, which in turn yields the solution to the problem.

Several examples of integer factorization achieved using AQC – and also using quantum annealing, which is a similar technique – can be found in recent literature. For instance, [Peng et al. 2008] showed the factorization of the number 21 via adiabatic quantum computing. Some time later, [Xu et al. 2012] successfully factored the number 143 in a quantum device. In 2014, however, [Dattani and Bryans 2014] explained that the same Hamiltonian employed by the previous paper is sufficient to factor larger numbers, such as 3599, 11663, and 56153. They also demonstrate the factorization of the number 175, which is the product of three primes (i.e., the product of $5 \times 5 \times 7$).

Although targeted with some criticism[5] for only considering hand-picked, easy instances of the integer factorization problem, these studies present promising results for factorization through adiabatic evolution. Additionally, most of the proposed Hamiltonians seem to require some type of quadratization method, that transforms 3- and 4-body interactions into 2-body interactions, due to physical limitations in current quantum devices.

Consequently, different quadratization methods for pseudo-Boolean functions have been proposed in recent years[6]. Following some new developments[7], it is clear that different quadratization methods can affect the adiabatic computation in different ways – these include the number of qubits needed and the range of coupling strengths between those qubits. For these reasons, the present study aims at exploring the effects that different quadratization methods have on the Hamiltonian for integer factorization. We will specifically address the methods proposed by [Ishikawa 2011] and [Dattani and Chau 2019].

The present work aims at studying the complexity of integer factorization, using the adiabatic quantum computing model as a background. The main goal is to study how different quadratization methods affect the Hamiltonian for the integer factorization

[5]See [Mosca and Verschoor 2019].

[6]See [Freedman and Drineas 2005]; [Ishikawa 2011]; [Dattani and Chau 2019].

[7]See [Dattani 2019].

problem. This study hopes to attain the following objectives: (1) study different ways to encode the integer factorization problem into a Hamiltonian operator, (2) present a general formula for the Hamiltonian of integer factorization, (3) implement the quadratization methods proposed by [Ishikawa 2011] and [Dattani and Chau 2019], on top of the integer factorization Hamiltonian, and explore their effects on the final Hamiltonian, and (4) provide a comparison between the quadratization methods proposed by [Ishikawa 2011] and [Dattani and Chau 2019].

2. Adiabatic quantum computing

In physics, an observable is any physical quantity that can be measured, such as the position or momentum of a body, for example. In classical physics, these observables are described by real-valued functions. In quantum systems, however, they are described by Hermitian operators, as per [Nielsen and Chuang 2010, Postulate 3, p. 85] and [Griffiths 2004, p. 97].

A Hermitian operator is a matrix H that obeys the relation $H = H^\dagger$, meaning it is self-adjoint. In particular, the result of measuring a quantum observable is always an eigenvalue of the associated Hermitian operator.

The Hermitian operator associated with the *total energy* of a quantum system receives a special name: it is called the Hamiltonian operator, or just Hamiltonian. Also, the *ground state* of a quantum system is the state with the lowest energy level out of all the possible states described by the system's Hamiltonian operator.

The idea behind the paradigm of adiabatic quantum computing (AQC) is to encode the solution to a problem into the ground state of a Hamiltonian H_f, and then evolve a quantum system from the ground state a known generic Hamiltonian H_i into the ground state of this specific Hamiltonian H_f. If no transition of state occurs during the evolution, then in principle the quantum system is guaranteed to arrive at the ground state of H_f, which by design encodes the solution to the initial problem. This model of computation serves as an alternative to standard circuit-based quantum computing, and yet both are provably computationally equivalent (meaning one can simulate the other). Thus, AQC is proven to be universal for quantum computing [Albash and Lidar 2018].

The difficult part in AQC is, in fact, ensuring no transition of state takes place. This is handled by a theorem from quantum mechanics called the adiabatic theorem. The word adiabatic, in this case, refers to the notion of adiabatic processes, which according to [Griffiths 2004, p. 368] are defined by a gradual change of external conditions. Below, we give a definition for this theorem inspired by the one given in [Albash and Lidar 2018]:

Theorem 2.1 (The Adiabatic Theorem) *A system prepared in the ground state $|\psi_0(0)\rangle$ of a time-dependent Hamiltonian $H(t)$ will remain in the instantaneous ground state $|\psi_0(t)\rangle$ of $H(t)$, provided that:*

1. *$H(t)$ varies slowly and gradually enough.*
2. *The energy gap between the ground state energy level E_0 and the first excited state energy level E_1 is sufficiently large.*

In other words, if a quantum system, prepared in its ground state $|\psi_0(t)\rangle$, suffers small interferences from the outside world, causing its Hamiltonian $H(t)$ to change very gradually, the system should still remain in its ground state. Although the ground state *per se* at a time $t = t_0$ may be different from the ground state at the initial time $t = 0$ (because it is time-dependent and thus varies with time), the system itself will be in a state which is the ground state at said time $t = t_0$.

This notion of the current ground state of an evolving system is usually called the instantaneous ground state, meaning the ground state of $H(t)$ at a fixed time $t = t_0$, or rather the ground state of $H(t = t_0)$.

Additionally, we can also describe the time-dependent Hamiltonian $H(t)$ through an equation, which marks the interpolation between H_i and H_f. To do so, let T be the total time for a computation. Let $t \in [0, T]$ mark the evolution of time inside the system. It can be convenient to work with a normalized interval, so let us define $s = t/T$ such that $s \in [0, 1]$. Thus, we say the computation starts at time $s = 0$ and ends at time $s = 1$.

Instead of working with $H(t)$, let us consider $H(s)$ without any loss of generality. It is possible to write the time-dependent Hamiltonian as follows:

$$H(s) = f_0(s)H_i + f_1(s)H_f,$$

where $f_0, f_1 : [0, 1] \to \mathbb{R}$ are interpolation functions.

The simplest example of such an interpolation is to choose $f_0(s) = (1 - s)$ and $f_1(s) = s$, yielding the Hamiltonian:

$$H(s) = (1 - s)H_i + (s)H_f.$$

3. Hamiltonian for integer factorization

In order to simplify the process of factoring an integer N, we impose certain conditions over said number, which can help to better study the problem. Namely, we expect N to be a positive, odd, composite integer with exactly two non-trivial factors, such that it can be written as $N = pq$.

In order to solve factorization via AQC, we must first find a way to construct a Hamiltonian H from an instance of factorization, such that the ground state of H encodes the solution to the problem. To do so, we define a cost function $f_N : \mathbb{Z}^2 \to \mathbb{Z}$ that attributes more energy to configurations of factors that do not multiply to give N, and less energy to the ones that do.

$$f_N(x, y) = (N - xy)^2. \tag{1}$$

Notice that f_N is always non-negative, and that $f_N = 0$ if and only if $N = xy$. For any other values of x, y that do not multiply to give N, $f_N(x, y)$ will be evaluated to a value larger than zero. Thus, intuitively, minimizing f_N for some value of x, y is enough to solve the problem of factoring N.

Now suppose we write x and y – which we have established must be odd since N is also odd – in their binary expansions, such that each Boolean variable x_i with coefficient 2^i represents the i-th bit of x, and the same with y_k for y. If we replace them in Equation 1, we get the following pseudo-Boolean function F_N:

$$F_N(x_1, ..., x_{n_x}, y_1, ..., y_{n_y}) = \left[N - \left(\sum_{i=1}^{n_x} 2^i x_i + 1\right)\left(\sum_{k=1}^{n_y} 2^k y_k + 1\right)\right]^2, \tag{2}$$

where n_x and n_y are the lengths (in bits) for x and y, respectively. Note that these lengths can be estimated via a result from [Peng et al. 2008], cited by [Hegade et al. 2021], where we impose, without loss of generality, the following conditions:

1. $x \leq y$,
2. and $3 \leq x \leq \sqrt{N}$,
3. and $\sqrt{N} \leq y \leq N/3$.

Then, provided these assumptions are met, the authors define the length bounds as follows[8]:

$$n_x = \lceil \log_2 \lfloor \sqrt{N} \rfloor_{odd} \rceil - 1, \quad \text{and} \quad n_y = \lceil \log_2 \lfloor \frac{N}{3} \rfloor \rceil - 1. \tag{3}$$

Further on, from Equation 2 it is easy to construct a Hamiltonian simply by replacing each Boolean variable x_i, y_k with the appropriate quantum operator $\hat{x_i}, \hat{y_k}$. A very natural choice for these operators is given below:

$$\hat{x_i} = \frac{I - \sigma_i^z}{2}, \quad \text{and} \quad \hat{y_k} = \frac{I - \sigma_k^z}{2},$$

where σ^z is the Z Pauli matrix and I is simply the identity matrix. For now, however, we will continue treating our Hamiltonian simply as a pseudo-Boolean function, so that the next steps are less convoluted than if we were dealing with matrices.

Thus, finally, we arrive at Equation 4, which describes the Hamiltonian for the problem, such as the one presented by [Hegade et al. 2021].

$$H_N = \left[N - \left(\sum_{i=1}^{n_x} 2^i x_i + 1 \right) \left(\sum_{k=1}^{n_y} 2^k y_k + 1 \right) \right]^2. \tag{4}$$

Furthermore, if we expand the xy product in Equation 4, we can simplify the Hamiltonian a bit further by noticing that any Boolean variable squared equals itself. We

[8]Here, $\lfloor \cdot \rfloor_{odd}$ means the largest odd integer not larger than $(\cdot)$.

then arrive at Equation 5, which is our general simplified Hamiltonian for factorization:

$$
\begin{aligned}
H_N = &\sum_{\substack{1\le i<j\le n_x\\1\le k<l\le n_y}} 2^{i+j+k+l+2} x_i x_j y_k y_l \\
&+\sum_{\substack{1\le i<j\le n_x\\1\le k\le n_y}} (2^{i+j+2k+1}+2^{i+j+k+2}) x_i x_j y_k \\
&+\sum_{\substack{1\le i\le n_x\\1\le k<l\le n_y}} (2^{2i+k+l+1}+2^{i+k+l+2}) x_i y_k y_l \\
&+\sum_{\substack{1\le i\le n_x\\1\le k\le n_y}} [2^{2i+2k}+2^{2i+k+1}+2^{i+2k+1}-2^{i+k+1}(N-2)] x_i y_k \\
&+\sum_{1\le i<j\le n_x} 2^{i+j+1} x_i x_j + \sum_{1\le k<l\le n_y} 2^{k+l+1} y_k y_l \\
&+\sum_{1\le i\le n_x} [2^{2i}-2^{i+1}(N-1)] x_i + \sum_{1\le k\le n_y} [2^{2k}-2^{k+1}(N-1)] y_k \\
&+(N-1)^2.
\end{aligned} \tag{5}
$$

4. Hamiltonian quadratization

The process of quadratizing a Hamiltonian – or any pseudo-Boolean function, for that matter – consists of transforming all its terms of degree $d > 2$ into quadratic terms (i.e., $d = 2$). Usually, this is done at the expense of adding auxiliary binary variables to the initial expression.

In the polynomial expression for the Hamiltonian in Equation 5 we have some 3rd- and 4th-degree monomials, which represent 3- and 4-body interactions between qubits, respectively. It is well known that two-body physical interactions occur more naturally than interactions involving many (more than 2) bodies [Dattani 2019]. Thus, it is generally easier to implement Hamiltonians with at most two-body interactions, than it is to implement those with cubic and quartic terms, for example.

For this reason, we have explored two general methods for quadratization. The first one was proposed by [Ishikawa 2011] and the second, by [Dattani and Chau 2019]. We have looked at them both in the context of the Hamiltonian for factorization defined in the previous section, in Equation 5.

4.1. Ishikawa's quadratization method

The author of [Ishikawa 2011] presents a general method for quadratizing a monomial of degree d, which is an extension of another quadratization method proposed by [Freedman and Drineas 2005]. For our case, we only need to apply these two substitutions, which include an extra Boolean variable b:

$$
\begin{aligned}
\alpha x_1x_2x_3 &= \alpha \min_{b\in\mathbb{B}} b(1-x_1-x_2-x_3) + \alpha(x_1x_2+x_1x_3+x_2x_3), \text{ and} \\
\alpha x_1x_2x_3x_4 &= \alpha \min_{b\in\mathbb{B}} b(3-2x_1-2x_2-2x_3-2x_4) \\
&\quad + \alpha(x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4).
\end{aligned}
$$

And since we are already trying to minimize the energy level via our cost function encoded into the Hamiltonian operator, we can ignore the restriction to minimize the extra b variable, simply by minimizing the overall value of our expression over all x, y and b variables.

4.2. Dattani-Chau's quadratization method

On the other hand, the method proposed by [Dattani and Chau 2019] involves the substitution of a 4-variable pseudo-Boolean in Equation 6 function by the expression given in Equation 7. Both are shown below:

$$\begin{aligned} f(x_1,x_2,x_3,x_4) = {} & \alpha_{1234}x_1x_2x_3x_4 + \alpha_{123}x_1x_2x_3 + \alpha_{124}x_1x_2x_4 \\ & + \alpha_{134}x_1x_3x_4 + \alpha_{234}x_2x_3x_4. \end{aligned} \tag{6}$$

$$\begin{aligned} f(x_1,x_2,x_3,x_4) = \min_{b\in\mathbb{B}} & \left(3\alpha_{1234} + \sum_{ijk}\alpha_{ijk}\right)b + \sum_{ij}\left(\alpha_{1234} + \sum_{k\notin ij}\alpha_{ijk}\right)x_ix_j \\ & - \sum_i\left(2\alpha_{1234} + \sum_{jk;i\notin jk}\alpha_{ijk}\right)bx_i, \end{aligned} \tag{7}$$

In order to format our Hamiltonian from Equation 5 into the form of Equation 6, we have proposed the following set of monomial groupings, which we have shown covers the whole Hamiltonian expression and has no overlaps:

#	Quartic monomial	Cubic monomials	Intervals
1	$x_1x_2y_1y_2$	$x_1x_2y_1$ $x_1x_2y_2$ $x_1y_1y_2$ $x_2y_1y_2$	-
2	$x_1x_2y_1y_l$	$x_1x_2y_l$ $x_1y_1y_l$ $x_2y_1y_l$	$l \geq 3$
3	$x_1x_jy_1y_2$	$x_1x_jy_1$ $x_1x_jy_2$ $x_jy_1y_2$	$j \geq 3$
4	$x_1x_jy_1y_l$	$x_1x_jy_l$ $x_jy_1y_l$	$j \geq 3,\ l \geq 3$
5	$x_1x_2y_ky_l$	$x_1y_ky_l$ $x_2y_ky_l$	$k \geq 2,\ l > k$
6	$x_ix_jy_1y_2$	$x_ix_jy_1$ $x_ix_jy_2$	$i \geq 2,\ j > i$
7	$x_1x_jy_ky_l$	$x_jy_ky_l$	$j \geq 3,\ k \geq 2,\ l > k$
8	$x_ix_jy_1y_l$	$x_ix_jy_l$	$i \geq 2,\ j > i,\ l \geq 3$
9	$x_ix_jy_ky_l$	-	$i \geq 2,\ j > i,\ k \geq 2,\ l > k$

Table 1. Nine non-overlapping monomial groupings.

5. Results

After applying the quadratization methods presented, we have calculated metrics in order to compare how each method affects the factorization Hamiltonian. First, we estimate how many auxiliary b variables each method introduces. This result if shown in Table 2.

Method	Number of extra variables
Ishikawa's count	$\frac{n_x n_y (n_x n_y + n_x + n_y - 3)}{4}$
Dattani-Chau's count	$\frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{4}$

Table 2. Number of extra variables introduced by each method.

Second, we estimated the number of result quadratic monomials for each method. This result is presented in Table 3.

Monomial format	Ishikawa's count	Dattani-Chau's count
xx	$\frac{n_x(n_x - 1)}{2}$	$\frac{n_x(n_x - 1)}{2}$
xy	$n_x n_y$	$n_x n_y$
yy	$\frac{n_y(n_y - 1)}{2}$	$\frac{n_y(n_y - 1)}{2}$
bx	$\frac{n_x n_y (n_x n_y + n_x - 2)}{2}$	$\frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{2}$
by	$\frac{n_x n_y (n_x n_y + n_y - 2)}{2}$	$\frac{n_x n_y (n_x n_y - n_x - n_y + 1)}{2}$

Table 3. Number of resulting quadratic terms for each method.

Moreover, if we interpret each variable (qubit) as a node and each quadratic monomial as an edge, we can interpret the whole Hamiltonian as an undirected weighted graph. From this, we can estimate how connected the Hamiltonian, which can be expressed as the number of connections (edges) divided by the number of qubits, or nodes. This value is sometimes called the graph's beta index, or simply β. The higher the graph's beta index, the more connected it is:

$$\beta = \frac{E}{V} = \frac{\text{"Number of edges"}}{\text{"Number of nodes"}}.$$

The behavior of beta indexes as N grows to be a very large number is shown in Figure 1. We have also proven that, in the limit as N tends to infinity, both β_{Ishi} and β_{Datt} tend to the value 4.

And finally, we have also estimated another important metric, which is the range of coefficients of the quadratic monomials that appear in the Hamiltonian expression. Particularly, we are interested in how this range grows as N becomes very large. We have calculated these ranges up to $N = 2048$ and the results are presented in Figure 2.

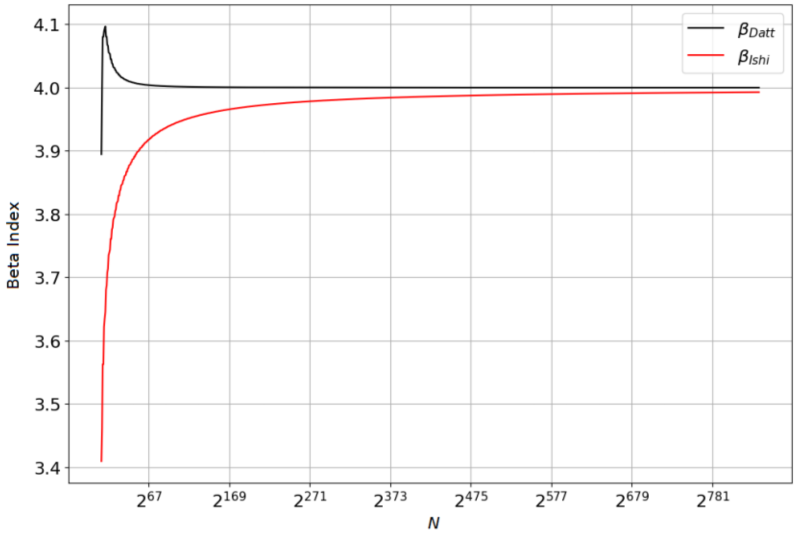


Figure 1. Growth of beta indexes. Plot showing the growth of beta indexes β_{Ishi} (in red) and β_{Datt} (in black) in relation to the number *N*, in the x-axis. Note that the x-axis is presented in logarithmic scale.

We can see from the plot that up until $N = 2048$, the monomial coefficients for both methods stay bounded by the green curves, which represent the range $[-N^3, +N^3]$. Although we could not prove this characteristic for larger values of N, it does suggest a trend that says the absolute value of these coefficients tends to range around (possibly strictly below) N^3. This is important, because it indicates that these values don't blow up exponentially as N grows larger, but rather can be bounded by a polynomial.

5.1. Estimate for large RSA keys

Furthermore, we provide an estimate for what our Hamiltonian would look like for large RSA keys, considering numbers with $n = 1024$ and $n = 2048$ bits, or $N = 2^{1024}$ and $N = 2^{2048}$, respectively. The results are presented in table 4.

	$\mathbf{n_x}$	$\mathbf{n_y}$	$\mathbf{M_{Ishi}}$	$\mathbf{M_{Datt}}$
$\mathbf{n = 1024}$ **bits**	511	1022	$6.838 * 10^{10}$	$6.798 * 10^{10}$
$\mathbf{n = 2048}$ **bits**	1023	2046	$1.097 * 10^{12}$	$1.094 * 10^{12}$

Table 4. Estimate for number of variables on large RSA keys.

At this point, we should say that even though numbers with 1024 and 2048 bits are indeed very large and notably hard to factor, we have not identified anything about the

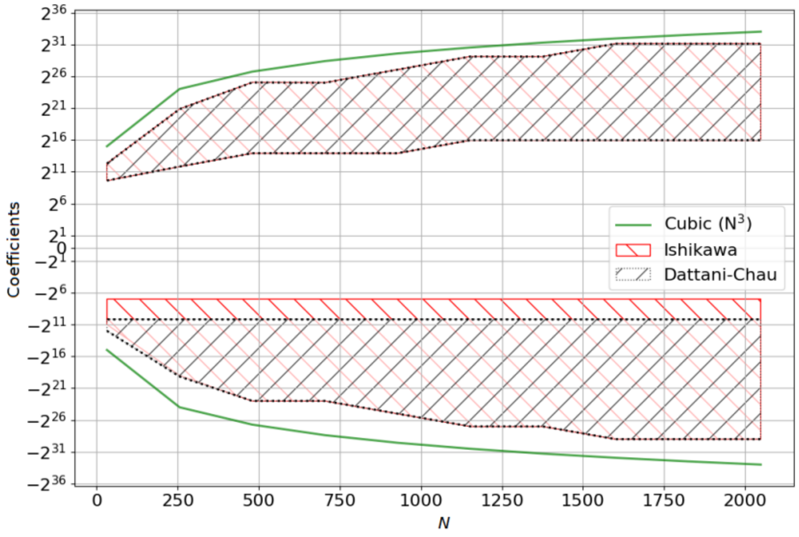


Figure 2. Plot showing the growing range of coefficients for each Hamiltonian expression. The y-axis, in logarithmic scale, shows the coefficient range for both Ishikawa's (red) and Dattani-Chau's (black dotted) Hamiltonian. The x-axis shows values of *N*. The negative areas show the range of least-to-most negative coefficients for said value of *N*, while positive areas of course show the range of least-to-most positive coefficient for that same value. The green curves show positive and negative values of N^3 for comparison.

internal structure of our Hamiltonian which would provide an improvement to integer factorization in general. This knowledge, combined with the number of variables estimated in Table 4, suggests it is not feasible to use this approach to factor large numbers, since it would require billions (or even trillions) of auxiliary variables to interact with just a few thousand x and y variables. In reality, we should not expect this arrangement of variables to be possible to implement.

We argue this limitation is mostly due to the structure of the cost function – meaning $f_N(x, y) = (N - xy)^2$ – that we chose in the beginning of this study. Nonetheless, this formulation was useful for evaluating the two quadratization methods that we sought out to compare. In future works, however, we wish to explore different configurations for the cost function and different methods for encoding the factorization problem into the Hamiltonian operator.

5.2. Discussion

Throughout this chapter, we have discussed different metrics for evaluating each method's quadratized Hamiltonian. Finally, we have compiled the overall results of our analysis, which are presented in Table 5.

From these results, we can conclude that Dattani-Chau's method is better at

Feature	**Ishikawa**	**Dattani-Chau**
(1) Lower number of extra variables	✗	✓
(2) Lower number of quadratic terms	✗	✓
(3) Lower connectivity (beta index)	✓	✗
(4) Smaller range of coefficients	✗	✗

Table 5. Comparison of quadratization methods.

quadratizing the original Hamiltonian using the least amount of extra variables (item 1). As a consequence of this, the resulting Hamiltonian also contains less quadratic monomials (item 2) than the one from Ishikawa's method.

Further on, we have also looked at the connectivity regarding each method's Hamiltonian. Previously, we have shown that Ishikawa's Hamiltonian always presents a smaller beta index (item 3) than Dattani-Chau's. In this context, the former method seems to have an advantage over the latter. However, we have also explained that this feature holds little importance, because Ishikawa's Hamiltonian, when interpreted as a graph, still utilizes more nodes and edges than the other method, and can only be considered the lesser-connected Hamiltonian (meaning the one with the smaller beta index) precisely because it uses more nodes (or qubits) to do so.

Moreover, we have also estimated how the range of coefficients (item 4) in the Hamiltonian grows for each method. We have shown that, at least for reasonably small values of N, the Hamiltonian coefficients tend to stay inside the interval $[-N^3, +N^3]$, and thus seem to be polynomially bounded in relation to the value of N. Other than that, we were not able to identify any significant difference between the two methods' expected range of coefficients, and therefore we have considered this feature a draw.

Finally, with all that was presented, we conclude that Dattani-Chau's quadratization method is overall better than Ishikawa's, considering the Hamiltonian that was the object of analysis and the scope of this study.

6. Conclusion

The main goal of this work was to study how different quadratization methods – specifically the ones proposed by [Ishikawa 2011] and [Dattani and Chau 2019] –, can affect the Hamiltonian for the integer factorization problem. As of yet, we have provided a general formula for this Hamiltonian, which includes simplifications regarding monomials with repeated binary variables. We have also provided a second general formula for the factorization Hamiltonian, quadratized via the method proposed by [Ishikawa 2011], and a third general formula for the same Hamiltonian, but instead quadratized through the method proposed by [Dattani and Chau 2019].

Moreover, we have evaluated different metrics for both quadratization methods, which were then used to compare each methods' cost and effectiveness in simplifying the original Hamiltonian operator. For each case, we have explored how the number of auxiliary variables grows, what is the connectivity between qubits of the Hamiltonian, and what is the range of coefficients (i.e. coupling strengths) required for the computation.

We have shown that the approach proposed by [Dattani and Chau 2019] is better suited for quadratizing our Hamiltonian, since it outperforms Ishikawa's method in almost every aspect we have analysed.

On top of that, we have also made estimates for what our Hamiltonian operator would look like if it were employed to factor large RSA numbers, with 1024 and 2048 bits. We have concluded that our approach would not be physically possible to implement, but that it still serves as a framework for comparing the aforementioned quadratization methods.

In future works, this study wishes to explore alternative ways of encoding the factorization problem into the Hamiltonian operator, aiming at approaches with better scalability that are based in different cost functions, or different multiplication algorithms. We also intend to explore other quadratization techniques, when applicable, in order to better understand the impacts that any quadratization process has on the Hamiltonian operators that are related to the factorization problem.

Finally, we also wish to perform actual examples of factorization via adiabatic quantum computing, so we can compare how each quadratization method affects the overall performance of the computation. Preferably, this should be done in an actual quantum computer, such as the ones provided by *D-Wave Systems*[9]. Depending on availability, however, this may also be performed using a quantum simulator.

References

Albash, T. and Lidar, D. A. (2018). Adiabatic quantum computation. *Reviews of Modern Physics*, 90.

Barker, E. B. and Dang, Q. H. (2015). Recommendation for key management part 3: Application-specific key management guidance. Technical report.

Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., and Zimmermann, P. (2020). Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. Cryptology ePrint Archive, Report 2020/697.

Dattani, N. and Chau, H. T. (2019). All 4-variable functions can be perfectly quadratized with only 1 auxiliary variable.

Dattani, N. S. (2019). Quadratization in discrete optimization and quantum mechanics. *ArXiv*.

Dattani, N. S. and Bryans, N. (2014). Quantum factorization of 56153 with only 4 qubits. *ArXiv*.

Farhi, E., Goldstone, J., Gutmann, S., and Sipser, M. (2000). Quantum computation by adiabatic evolution.

Freedman, D. and Drineas, P. (2005). Energy minimization via graph cuts: Settling what is possible.

Griffiths, D. J. (2004). *Introduction to Quantum Mechanics (2nd Edition)*. Pearson Prentice Hall, 2nd edition.

[9]See `https://www.dwavesys.com/`.

Hegade, N. N., Paul, K., Albarrán-Arriagada, F., Chen, X., and Solano, E. (2021). Digitized adiabatic quantum factorization.

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

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Boston, third edition.

Lenstra, A. K., Lenstra, H. W., Manasse, M. S., and Pollard, J. M. (1993). The number field sieve. In *Lecture Notes in Mathematics*, pages 11–42. Springer Berlin Heidelberg.

Martín-López, E., Laing, A., Lawson, T., Alvarez, R., Zhou, X.-Q., and O'Brien, J. L. (2012). Experimental realization of shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776.

Mosca, M. and Verschoor, S. R. (2019). Factoring semi-primes with (quantum) sat-solvers. *CoRR*, abs/1902.01448.

Nielsen, M. A. and Chuang, I. L. (2010). *Quantum computation and quantum information*. Cambridge University Press.

Peng, X., Liao, Z., Xu, N., Qin, G., Zhou, X., Suter, D., and Du, J. (2008). Quantum adiabatic algorithm for factorization and its experimental implementation. *Physical Review Letters*, 101(22).

Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. pages 124–134. Institute of Electrical and Electronics Engineers (IEEE).

Stallings, W. (2013). *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, USA, 6th edition.

Xu, N., Zhu, J., Lu, D., Zhou, X., Peng, X., and Du, J. (2012). Erratum: Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system [phys. rev. lett.108, 130501 (2012)]. *Physical Review Letters*, 109(26).