ABSTRACT

Title of Dissertation: QUANTUM COMBINATORIAL OPTIMIZATION ALGORITHMS FOR PACKING PROBLEMS IN CLASSICAL COMPUTING AND NETWORKING

Cem M. Unsal
Doctor of Philosophy, 2023

Dissertation Directed by: Professor Yavuz A. Oruc
Department of Electrical and Computer Engineering

In Computer Engineering, packing problems play a central role in many aspects of hardware control. The field aims to maximize computer processing speed, network throughput, and dependability in industry applications. Many of these constrained maximization problems can be expressed as packing problems in integer programming when working with restrictions such as latency, memory size, race conditions, power, and component availability. Some of the most crucial of these integer programming problems are NP-hard for the global optimum. Therefore, real-world applications heavily rely on heuristics and meta-heuristics to find good solutions. With recent developments in quantum meta-heuristic methods and promising results in experimental quantum computing systems, quantum computing is rapidly becoming more and more relevant for complex real-world combinatorial optimization tasks. This thesis is about applications of quantum combinatorial optimization algorithms in classical computer engineering problems. These include novel

quantum computing techniques that respect the constraints of state-of-the-art experimental quantum systems. This thesis includes five projects.

Faster Quantum Concentration via Grover's Search

One of the most important challenges in information networks is to gather data from a larger set of nodes to a smaller set of nodes. This can be done via the use of a concentrator architecture in the connection topology. This chapter is a proof-of-concept that demonstrates a quantum-based controller in large interconnection networks can asymptotically perform this task faster. We specifically present quantum algorithms for routing concentration assignments on full-capacity fat-and-slim concentrators, bounded fat-and-slim concentrators, and regular fat-and-slim concentrators. Classically, the concentration assignment takes $O(n)$ time on all these concentrators, where n is the number of inputs. Powered by Grover's quantum search algorithm, our algorithms take $O(\sqrt{nc}\ln c)$ time, where c is the capacity of the concentrator. Thus, our quantum algorithms are asymptotically faster than their classical counterparts when $c\ln^2 c = o(n)$. In general, $c = n^{\mu}$, satisfies $c\ln^2 c = o(n)$, implying a time complexity of $O(n^{0.5(1+\mu)}\ln n)$, for any $\mu, 0 < \mu < 1$.

Quantum Adversarial Learning in Emulation of Monte-Carlo Methods for Max-cut Approximation: QAOA is not Optimal

One of the leading candidates for near-term quantum advantage is the class of Variational Quantum Algorithms. However, these algorithms suffer from classical difficulty in optimizing the variational parameters as the number of parameters increases. Therefore,

it is important to understand the expressibility and power of various ansätze to produce target states and distributions. To this end, we apply notions of emulation to Variational Quantum Annealing and the Quantum Approximate Optimization Algorithm (QAOA) to show that variational annealing schedules with equivalent numbers of parameters outperform QAOA. Our Variational Quantum Annealing schedule is based on a novel polynomial parameterization that can be optimized in a similar gradient-free way as QAOA, using the same physical ingredients. In order to compare the performance of ansätze types, we have developed statistical notions of Monte-Carlo methods. Monte-Carlo methods are computer programs that generate random variables that approximate a target number that is computationally hard to calculate exactly. While the most well-known Monte-Carlo method is Monte-Carlo integration (e.g., Diffusion Monte-Carlo or path-integral quantum Monte-Carlo), QAOA is itself a Monte-Carlo method that finds good solutions to NP-complete problems such as Max-cut. We apply these statistical Monte-Carlo notions to further elucidate the theoretical framework around these quantum algorithms.

Scheduling Jobs in a Shared High-Performance Computer with a NISQ Computer

Several quantum approximation algorithms for NP-hard optimization problems have been described in the literature. The properties of quantum approximation algorithms have been well-explored for optimization problems of Ising type with 2-local Hamiltonians. A wide range of optimization problems can be mapped to Ising problems. However, the mapping overhead of many problem instances puts them out of the reach of Noisy

Intermediate-scale Quantum (NISQ) devices. In this chapter, we develop a way of mapping constrained optimization problems to higher-order spin interactions to put a larger set of problem instances within reach of spin interaction devices with potential NISQ applications. We demonstrate the growth in the practicable set of problem instances by comparing the resource requirements as a function of coupling. As an example, we have demonstrated our techniques on the problem of scheduling jobs in a high-performance computer queue with limited memory and CPUs.

Protein Structures with Oscillating QPacker

A significant challenge in designing proteins for therapeutic purposes is determining the structure of a protein to find the sidechain identities given a protein backbone. This problem can be easily and efficiently encoded as a quadratic binary optimization problem. There has been a significant effort to find ways to solve these problems in the field of quantum information, both exactly and approximately. An important initiative has applied experimental quantum annealing platforms to solve this problem and got promising results. This project is about optimizing the annealing schedule for the sidechain identity problem, inspired by cutting-edge developments in the algorithmic theory of quantum annealing.

On the Complexity of Generalized Discrete Logarithm Problem

Generalized Discrete Logarithm Problem (GDLP) is an extension of the Discrete Logarithm Problem where the goal is to find $x \in \mathbb{Z}_s$ such $g^x \mod s = y$ for a given $g, y \in \mathbb{Z}_s$. The generalized discrete logarithm is similar, but instead of a single base element,

it uses a number of base elements that do not necessarily commute. In this chapter, we prove that GDLP is NP-hard for symmetric groups. The lower-bound complexity of GDLP has been an open question since GDLP was defined in 2008 until our proof. Furthermore, we prove that GDLP remains NP-hard even when the base elements are permutations of at most three elements. Lastly, we discuss the implications and possible implications of our proofs in classical and quantum complexity theory.

QUANTUM COMBINATORIAL OPTIMIZATION ALGORITHMS FOR PACKING PROBLEMS IN CLASSICAL COMPUTING AND NETWORKING

by

Cem Mehmet Unsal

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Yavuz A. Oruc, Chair/Advisor
Dr. Lucas T. Brady
Professor Eric V. Slud
Professor Maria K. Cameron
Professor Michelle Girvan

Foreword

This thesis includes writings from my previous works in part for some and as a whole for others. Three of them are jointly authored completed writings. These are "Faster Quantum Concentration via Grover's Search" which was advised by Dr. Yavuz Oruc, "On the Complexity of Generalized Discrete Logarithm Problem" which Dr. Rasit O. Topaloglu advised, and "Quantum Adversarial Learning in Emulation of Monte-Carlo Methods for Max-cut Approximation: QAOA is not optimal" which was advised by Dr. Lucas T Brady. I am also currently working on a project titled "Quantum Annealing with Higher-Order Magnetic Coupling Hamiltonian and Its Job Scheduling Applications" with Dr Lucas T Brady which is included in this thesis. In all these projects, I was the first author and the primary contributor. I was the only graduate student in all these projects, and the previously mentioned people were the only co-authors. These are cited with regular in-text citations. Three of the writings that are individually authored by me are also included. These are my candidacy prospectus titled "Quantum Algorithms for Switching Networks", my dissertation prospectus, "Quantum Computing in Combinatorial Optimization for Telecommunication Networks", and a report I submitted in a private email to Dr. Vikram Mulligan at the end of my internship for Menten AI, Inc. For the internship report, I asked and was granted permission to reuse this material. These three individually authored writings are not publicly available and, therefore, cannot be provided with a citation to point the reader to. This disclosure is part of the requirements stated in "Steps

& Deadlines for a Spring 2023 Graduation".

Acknowledgments

I would like to thank all the people that helped me and supported me along the way. They are:

- All my professors from the classes I have taken and audited for their diligence in teaching.
- Department of Mathematics for financial assistance by giving me the opportunity for the teaching assistantship position.
- Howard Elman and Jessica Saddler for helping me understand the policies and processes of my program.
- Michael Perrone for introducing me to quantum computing during the freshmen year of my undergraduate education.
- Alexandra Gurel and Ellie Nielsen for helping me become a better communicator.
- All the scientists whose work made my work possible.
- Yusuf Alnawakta, Eric Slud, Eleanor Rieffel, and Aniruddha Bapat valuable conversations and sharing their expertise.
- Elliott Lehrer and Drew Risinger for formative discussions that led me to some of my research topics.

- Yesim Mercan for helping me develop a deep understanding of the nuances of professional communication, networking, and confidence.
- Eddie Schoute for leading an earlier project I have co-authored (not included in this thesis).
- Andrew Childs for advising that project.
- Rasit Onur Topaloglu for advising the "On the Complexity of Generalized Discrete Logarithm Problem" project, where we answered a more than a decade-old open research question.
- Vikram Mulligan and Gavin Crooks for advising my internship at Menten AI Inc.
- Hans Melo, Tamas Gorbe, and Menten AI Inc for giving me the internship opportunity.
- KBR Inc (Prime Contract No. 80ARC020D0010) and NASA Quantum Artificial Intelligence Laboratory (QuAIL) for giving me an internship opportunity.

There are two people whom I would like to emphasize my gratitude to especially. The first one is Lucas Brady, who shared his expertise for one project, advised another completed project, mentored QuAIL internship, and is currently advising another project. Thank you Lucas for everything!

Last but not least, I would like to declare my gratitude to my Advisor, Yavuz Oruc, for guiding me throughout my journey, teaching me critical technical skills, teaching me critical research skills, helping me become the scientist I am today, advising a project, countless

valuable discussions in others, all the major and minor comments on my writings, letting me explore, always keeping a critical perspective, detailed questions that helped me catch what I miss, and countless other ways in which he helped me.

Table of Contents

List of Figures

List of Abbreviations

AKA	Also Known As
AQC	Adiabatic Quantum Computing
BAB	Bang-Anneal-Bang Schedule
BQP	Bounded-Error Quantum Polynomial-Time
CBS	Computational Basis State
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
DH	Diffie–Hellman key exchange protocol
DLP	Discrete Logarithm Problem
GDLP	Generalized Discrete Logarithm Problem
HSP	Hidden Subgroup Problem
KBR	Kellogg Brown & Root, Inc
NASA	National Aeronautics and Space Administration
NISQ	Noisy Intermediate-scale Quantum
NP	Nondeterministic Polynomial-Time
PDF	probability distribution/density/mass function
QA	Quantum Annealing
QAOA	Quantum Approximate Optimization Algorithm
QuAIL	NASA Quantum Artificial Intelligence Laboratory
SAT	Boolean Satisfiability Problem

Chapter 1: Introduction

The promise of quantum computing in speeding up computations continues to attract research into exploring quantum algorithms for problems that arise in computer science, mathematics, and other scientific fields beyond searching and factorization [2]. Indeed, several quantum algorithms have been reported for a wide range of computational problems extending from algebraic topics to pattern matching and many problems in graph theory. See [5] and [6] for a survey of quantum algorithms for Abelian and non-Abelian discrete Fourier transform, hidden subgroup problem, computing discrete logarithms, and several other problems in number theory, cryptography, and group theory. However, despite the vast potential, the effort to develop quantum algorithms for problems that appear in classical computation and networks has been sparse outside of the work included in this thesis [7,8]. That being said, the literature has considered ways to fuse classical and quantum computations as a way to augment the capacity of the classical circuits. "The power of small-depth quantum circuits" conjecture states that any quantum circuit of polynomial size can be simulated with a polylogarithmic depth quantum circuit and a polynomial depth classical circuit [9]. If true, this is expected to be the single most important fact in the field of quantum information, as it makes it much easier to build a quantum computer compared to the known methods. Another way to state this conjecture is that classical computers,

when aided by short quantum circuits, can be extremely powerful in terms of the types of computational problems they can solve. In classical literature for computer engineering, there are already well-defined problems relating to resource allocation where improvements that can find better solutions to these problems are known to increase the power of classical hardware, albeit it is not known if better resource allocation of classical computers can be as powerful as quantum computers. Nevertheless, just like in "the power of small-depth quantum circuits" conjecture, it is possible to greatly amplify the computational power of a hybrid quantum-classical computer if the quantum part is deployed in a way that makes the classical part use its resources more efficiently. This thesis is the pursuit of that vision.

The rest of this thesis is organized as follows. Chapter 2 describes how quantum algorithms can be used to time-efficiently compute the optimal routing paths in several topologies used in classical networking. Chapters 3, 4, and 4 are about approximate quantum algorithms. Chapter 3 describes a novel quantum combinatorial optimization algorithm that is less resource intensive than the renowned Quantum Approximate Optimization Algorithm (QAOA). Chapter 4 describes how quantum algorithms can be used to schedule jobs in a queue of a supercomputer. Chapter 4 is about the implementation of approximate quantum algorithms for a different but qualitatively similar problem. It describes a quantum algorithm that is used to estimate the structure of proteins. Chapter 5 proves how a slight generalization of the discrete logarithm problem makes it NP-hard. The proof of NP-hardness of this problem might have considerable implications in the field of quantum algorithms since the discrete logarithm problem is one of the small number of problems that can be efficiently solved with a quantum computer but is almost universally assumed by cryptographers to not be feasibly solvable with classical computers. The last chapter

summarizes the main takeaways from this thesis and identifies important future directions.

1.1 Quantum Information Background Concepts

Qubits represent quantum information, with each qubit being in a superposition of the states $|0\rangle$ and $|1\rangle$ [10]. The state of a qubit, $|\psi\rangle$, is determined by probability amplitude angle θ and relative phase ϕ: $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$. This is a unit vector in the 2-dimensional Hilbert Space. The number of computational basis states (CBS) doubles with each additional qubit, where k qubits provide $N = 2^k$ states. These basis states are denoted with $|i\rangle$ for $i \in \mathbb{Z}_N$ that represents the unit vector in ith dimension. Each of these additional states has its own relative phase and probability amplitude. Quantum Computing is the field of Computer Science that explores computation using quantum logic as opposed to classical logic. Quantum logic can do operations that are based on Quantum Physics in addition to all the operations in classical logic. These operations allow for more efficient algorithms than classical computers in terms of resource use. The most commonly considered resources are time, memory, hardware size, communication capabilities, and query count. The goal of quantum computing is to formulate decreased resource use compared to classical systems [10]. Evaluations are usually specified in terms of Landau's asymptotic families of functions [11,12] i.e. big-O, small-O, big-Omega, small-Omega and big-theta.

The state of a quantum system evolves according to deterministic rules, but measuring a quantum system causes the superposition of the state to collapse to a single state in a randomized manner. The probability of observing a CBS upon measurement is the square

of its magnitude corresponding to the entry in the vector that represents the quantum state. Due to this innate randomness, the decreased resource use is compared to classical probabilistic Turing machines [13]. Just like with classical probabilistic algorithms, the goal with quantum algorithms is to increase the probability of observing the correct solution to the computational problem, and quantum computing has some additional knobs which are not present in probabilistic Turing machines that can be used such as the interference of relative phases that can be used to for more efficient computation.

1.1.1 The Time-Evolution of Quantum States

Quantum states evolve according to the Schrodinger equation:

$$H(t)|\psi(t)\rangle = i\hbar\frac{\partial|\psi(t)\rangle}{\partial t}$$

where $H(t)$ refers to the Hamiltonian, $\hbar$ is the Reduced Planck's constant and $|\psi(t)\rangle$ refers to a complex vector representing the state [10,14]. When Hamiltonian is constant, $H(t) = H$, this gives rise to dynamics that are governed by the following unitary transformations:

$$|\psi(t)\rangle = \exp\left(\frac{-iHt}{\hbar}\right)|\psi(0)\rangle.$$

The term "Quantum Gate" without any qualifiers[1] usually refers to the unitary matrices $\exp\left(\frac{-i\mathfrak{H}\mathfrak{T}}{\hbar}\right)$ where $\mathfrak{H}$ is a constant Hamiltonian matrix and $\mathfrak{T}$ is a constant amount of time. The most general form of gate-based quantum computing is a Solovay–Kitaev universal

[1] This is as opposed to the term "Parameterized Quantum Gate" where the time or the Hamiltonian may depend on some variable.

quantum computer [10]. Solovay–Kitaev universality refers to a quantum computer that can entangle qubits and approximately perform any single-qubit unitary operations to arbitrary accuracy. A set of quantum gates that fit this definition are called Solovay–Kitaev universal. Another model of quantum computing is analog quantum computing, such as Quantum Annealing. This thesis makes use of several important quantum algorithms in both categories that already exist in the literature for comparison, inspiration, and as a subroutine. These are Grover's Search Algorithm [15], Quantum Approximate Optimization Algorithm [16], and Quantum Annealing [17].

1.1.2 Grover's Search Algorithm

Quantum computers are known for their ability to solve certain problems faster than classical computers, such as finding the index of the unique marked element in a database, which is known as an unordered database search problem [15]. Mathematically, this problem is described as follows. Let S be a database of N elements such that there is a unique element, S_v, that satisfies a Boolean function $B(S_v) = 1$, and none of the other elements satisfies this Boolean function, where S_i denotes the ith element in the database for $0 \leq i \leq N-1$. The satisfying element S_v is known as a 'solution'. The goal of the unordered database search problem is to find the index v of this solution.

The time that a classical computer takes to solve this problem is proportional to the database size, which is $O(N)$ [15]. However, quantum computers can do this faster via Grover's search, which only takes $O(\sqrt{N})$ time. Grover's search uses a unitary operation that identifies the marked item in the list by a relative phase shift followed by another

unitary operation that amplifies the probability amplitude of the identified element. This process is repeated to find the index of the unique marked item in $O(\sqrt{N})$ time. Grover's seminal work was extended to the case of M solutions in the database, where $M > 1$ [18]. The time to find one of the solutions using this version of Grover's search is $O(\sqrt{N/M})$ as compared to $O(N/M)$, the average time required by classical computation.

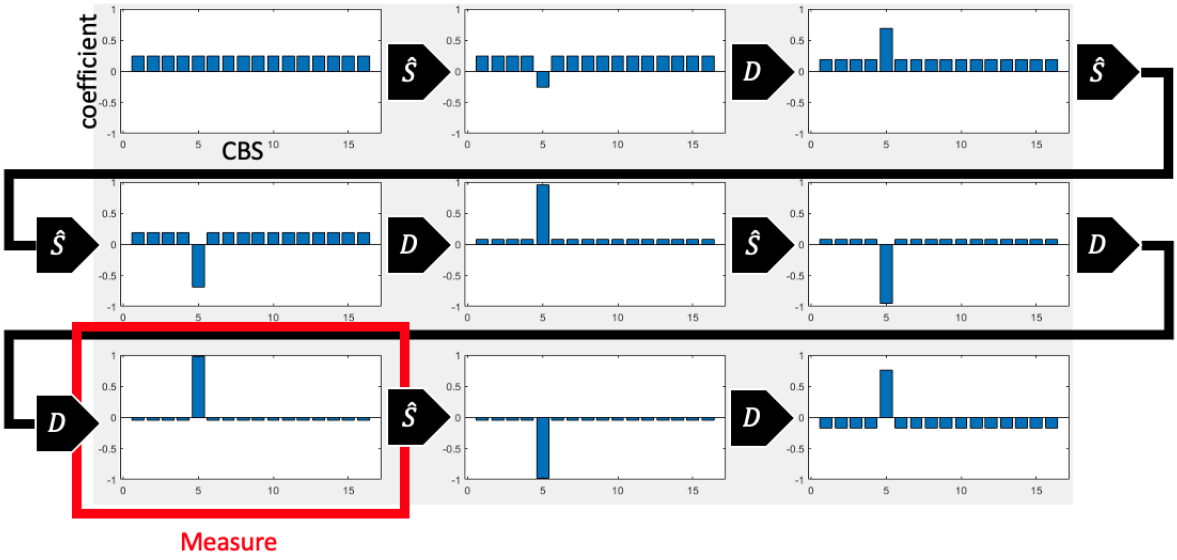


Figure 1.1: The step-by-step evolution of the state in Grover's search of 16 elements where the fifth element from the left is the solution. The best step in the iteration to perform the measurement is indicated with the red rectangle.

This algorithm starts by transforming the classical zero state to the quantum uniform superposition state, i.e. the vector with all $1/\sqrt{N}$ entries [15]. This is efficiently performed using a quantum logic operation known as "Haddamard Gates", i.e.

$$H^{\otimes n}|0\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} = \sum_{i=0}^{2^n-1} \frac{|i\rangle}{\sqrt{2^n}}$$

$$\text{where } H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} / \sqrt{2}, n = \lceil \log_2 N \rceil,$$

and $\otimes$ is the tensor product. Next, the state evolves under two different energy landscapes that periodically alternate. These energy landscapes are described by their respective Hamiltonian matrix. The first Hamiltonian is used to rotate the phase of the basis state corresponding to the solution's index. The unitary that this Hamiltonian creates is a diagonal matrix $\hat{S}$ with

$$\hat{S}_{i,i} = \begin{cases} -1 & \text{if } i = v \\ 1 & \text{otherwise} \end{cases}$$

where v is the index of the marked element. This effect is known as "selective rotation". The second Hamiltonian is used to increase the magnitude of the vector for rotated basis states and decreases the magnitude of other elements in the state vector. The unitary that this Hamiltonian creates is matrix D with

$$D_{i,j} = \begin{cases} -1 + 2/N & \text{if } i = j \\ 2/N & \text{otherwise.} \end{cases}$$

The effect of the second Hamiltonian is known as "diffusion". While doing diffusion, the phase of the negative entries is set back to 0, meaning that to keep increasing the magnitude, we must once again do selective rotation. These two transformations are applied $O(\sqrt{N})$ times (or $O(N/M)$ for the case of multiple solutions). These steps are demonstrated in the example in Figure 1.1.

Grover's search is widely applicable. Therefore, it is one of the most important algorithms used to prove a theoretical speedup over classical algorithms [2, 19, 20], but it is not easy to implement it experimentally. Even so, the method of using just two

Hamiltonians to perform quantum computation is still relevant to modern experimental setups, especially for Variational Quantum Algorithms.

1.1.3 Variational Quantum Algorithms [1]

Variational Quantum Algorithms have emerged as one of the most popular classes of near-term quantum algorithms, finding use cases in optimization [16], chemistry [21], imaginary time evolution [22], quantum machine learning [23], and many other applications. This class of algorithms relies on parameterized quantum circuits and a variational classical outer loop that optimizes the quantum circuit parameters based on measurements of quantum observables. Two of the big appeals of these variational algorithms are:

- they are slightly robust to noise since the variational loop can account for this, and
- they allow for potential quantum advantage through their variational nature rather than meticulously crafted algorithmic design.

While these variational algorithms can be parameterized using any available quantum circuit, variational parameters that do not incorporate enough information about the problem, often run into so-called barren plateaus where there is an insufficient direction to the optimization process [24]. A way to avoid this is to use as much information about the problem instance as possible in the design.

One key example of this is the Quantum Approximate Optimization Algorithm (QAOA). Unlike Grover's search, which can be used to find the global minimum, QAOA is an approximate method that finds good solutions to combinatorial optimization problems, as the name suggests. However, just like Grover's search, it mainly consists of evolving the

quantum state under two alternating Hamiltonians. This is known as bang-bang control or 2-step control. The evolution is

$$e^{-i\hat{B}\beta_{p-1}}e^{-i\hat{C}\gamma_{p-1}}\dots e^{-i\hat{B}\beta_0}e^{-i\hat{C}\gamma_0}|\psi_0\rangle$$

where β_is and γ_is are variational parameters that determine the time between the alternations. The two Hamiltonians are a simple driver Hamiltonian, $\hat{B}$, such as a transverse field on qubits, and the problem Hamiltonian, $\hat{C}$, which encodes the energy landscape of the target problem [16]. By starting at the ground state of $\hat{B}$, we can apply the bang-bang control, trying to find a state that minimizes its energy with respect to $\hat{C}$, treating the time that the Hamiltonian is applied (lengths of the bangs) as variational parameters. Unlike Grover's search, QAOA does not evaluate states in an all-or-nothing manner rather, each state is gauged in a way that depends on how well it minimizes $\hat{C}$. QAOA was inspired by Quantum Annealing [25, 26], which has the same goal and uses the same Hamiltonians and initial state. While QAOA is a variational bang-bang algorithm, Quantum Annealing relies on a continuous ramp from the driver to the problem Hamiltonian. For long enough runtimes, the quantum adiabatic theorem [27] ensures that such an annealing procedure will transform a system, initially in the ground state of $\hat{B}$, into the ground state of $\hat{C}$ with high probability, minimizing the energy. While quantum annealing was originally proposed to be adiabatic, many non-adiabatic or diabatic variants and usages of Quantum Annealing have been proposed [28] with numeric success in some specific problems but without a general analytic framework.

Both QAOA and Annealing Ramp fall into a class of analog quantum algorithms

which can be described by the Hamiltonian evolution

$$\hat{H}(t) = u(t)\hat{B} + (1 - u(t))\hat{C}, \tag{1.1}$$

where $u(t) \in [0, 1]$ is the generalized annealing schedule that takes on a bang-bang form in QAOA and a decreasing ramp form in annealing. It would of course be possible to generalize this further by replacing $(1 - u(t)) \to v(t)$ and having independent controls. Whether independent controls are implementable is dependent on the quantum system at hand, and these two models have equivalent computational power under polynomial-time reductions. Restricting the system to a single control function, $u(t)$, is often easier from a theoretical point of view, and will be the only model we consider.

Some work has gone into the connection between QAOA, and Quantum Annealing [16, 29–32], and it has been shown that for constrained annealing time, optimizing for energy expectation and searching on the space of all annealing schedule functions, the optimal solution to Eq. (1.1) approaches a smooth, adiabatic like annealing ramp in the long runtime limit. Furthermore, it has been observed that this optimal schedule oscillates at a frequency which empirically matches the optimal bang lengths in QAOA [32]. This correspondence to an underlying optimal curve provides a link and justification for the practice of QAOA bootstrapping.

It has been noted in several studies [30, 33] that QAOA parameters asymptotically form smooth curves that are independent of the number of bang-bang layers, p. In other words, for a $p = 10$ layer QAOA procedure, the 5th layer's parameters will look similar to the 10th layer's parameters from a $p = 20$ QAOA procedure on the same problem. This

indicates [32] that the QAOA optimization is tapping into some underlying procedure, such as the above-cited optimal bang-anneal-bang procedure, and this behavior provides a useful way to bootstrap up from low p QAOA to higher p, improving the performance.

Chapter 3 again utilizes this connection between QAOA and the optimal procedure by creating an alternative parameterization that is able to capture more information from the optimal procedure. We still parameterize $u(t)$, using $2p$ parameters, but instead of having those parameters be bang-bang lengths, these parameters will instead be coefficients of a higher-order polynomial. We discuss the details of this parameterization and its theoretical underpinnings in Sec. 3.3. The concept of customizing a quantum annealing schedule to a problem dates back to Roland & Cerf's optimized adiabatic schedule for unstructured search [34] and its subsequent generalization into the Quantum Adiabatic Brachistochrone [35]. Most attempts at variational optimization of an annealing schedule have worked within the context of an adiabatic path, meaning that most attempts at variational schedules still have a ramp from an initial to a final schedule. For instance, recently VanQver and its derivatives [36–38] have applied variational techniques on a third catalyst Hamiltonian that is only present in the middle of the anneal. There has been some work on variationally implementing counterdiabatic schedules [39–42], but these still seek to create an adiabatic path. The work in chapter 3 chapter is novel in that it attempts a variational annealing schedule with no reference to adiabaticity at all, except for a few edge cases.

This thesis draws on both exact and approximate quantum algorithms. Chapters 3, 4, and 4 are about variational quantum algorithms for approximate optimization. Chapter 3 is about how the variational degrees of freedom can be effectively used. Chapter 4 is about how variational quantum algorithms can be used for constrained optimization problems

such as job scheduling. Chapter 4 is about implementing variational quantum algorithms for the protein design problem. On the other hand chapter 2, is about exact quantum algorithms that are based on Grover's search.

Chapter 2: Faster Quantum Concentration via Grover's Search [2]

Graph theoretical problems have been a key application of quantum algorithms since the 2000s. Algorithms like Grover's search algorithm and quantum walk have been used to solve matching and transportation network problems [43–45] and graph traversals [46]. In this chapter, we focus our attention on a different direction, namely the application of quantum computing to developing fast quantum algorithms to realize connection requests in concentrators. Concentration is a fundamental operation in interconnection. Thus, we think that developing fast quantum algorithms for realizing connection requests in concentrators will likely have a significant impact on routing in most interconnection networks. It is important to underscore that our goal in this chapter is not to develop new techniques in quantum algorithms but rather to prove that for certain network connection models, a quantum controller is asymptotically better than a classical one. As mentioned above, the field of quantum algorithms, using quantum data processing elements to solve computational problems, is an area that has rich literature and is an active area of research with enthusiastic interest behind it. However, we believe that the power of quantum algorithms goes beyond that. In this work, we extend and adapt Grover's seminal work to network controllers. In particular, we focus on the identification of active inputs in concentration assignments.

2.1 Sparse Crossbar Concentrators

A concentrator is a telecommunications device that has more inputs than it has outputs [47, 48]. The goal of this device is to identify the active inputs and route them to the outputs in a process called concentration. Pinsker established that concentrators with n inputs, m outputs, and at most $29n$ edges exist for all $m \leq n$ in [49]. Since Pinsker's seminal result, quite a few concentrator designs have been reported in the literature, see for example [47, 50–58]. The earlier designs in [50–52] provide non-explicit solutions and can be viewed as upper-bound results. The first explicit concentrator was reported in [53] using a binomial sparse crossbar design with $O(n^{1.5})$ edges. The design given in [54] falls outside the focus of our work as it is based on binary comparators and sorters. This chapter is on designing quantum algorithms for certain families of concentrators that can be constructed with crossbar switches (see Figure 2.1). The goal of these algorithms is to compute the control bits of these switches that realize the given routing assignment. These families of concentrators were introduced in [47] and developed in [55–58]. We note that concentrators are further refined to study a family of bipartite graphs, widely known as expanders [59–63]. Expanders provide bounded capacity concentration with $O(n)$ edges. In this chapter, we give classical algorithms to realize concentration assignments in sparse crossbar concentrators and use Grover's search to transform them into quantum algorithms, decreasing their time complexity. The rest of the chapter is organized as follows. The next section provides the preliminary concepts needed to describe our results. In Section 2.2, we state our assumptions of classical and quantum models of computation. Sections 2.3 and 2.4 present our main results. The chapter is concluded in Section 2.5 with a discussion of

our results and possible directions for future research.

Concentrators are defined by three positive integers: the number of inputs, the number of outputs, and the capacity. An (n, m, c)-concentrator refers to a concentrator with n inputs, m outputs, and a capacity of c. As mentioned in the introduction, there are more inputs than outputs, thus $n > m$. Formally, a c capacity concentrator is a graph where a set of vertices represents the inputs, another set of vertices represents the outputs, and non-overlapping paths exist between every size c subset of inputs and at least one size c subset of the outputs[1]. The existence of non-overlapping paths is needed to allow for simultaneous connections to be created between the active inputs and the outputs. A concentrator is referred to as a bipartite or sparse concentrator if all paths are of length one, i.e., each path consists of two vertices and an edge that is represented by a crosspoint in the fat-and-slim crossbar model [47]. If $c < m$, then an (n, m, c)-concentrator is said to have bounded capacity, and if $c = m$, it is said to have full capacity. Henceforth, we will refer to the latter as an (n, m)-concentrator. A concentrator is called explicit if all of its edges (crosspoints) are specified, inexplicit if all of its edges exist with a non-vanishing probability other than 1, and semi-explicit if only a proper subset of its edges is explicitly specified.

Numerous explicit, semi-explicit, and inexplicit concentrators have been described in the literature, as mentioned above. In this chapter, we focus on concentrators that can be constructed with crossbar switches (see Figure 2.1) called crossbar concentrators. Specifically, explicit crossbar concentrator designs, two of which were introduced in [47]

[1]In other words, the outputs are assumed to be interchangeable in the routing assignments. Therefore, any subset of the inputs with less than c elements can form connections through non-overlapping paths with different output vertices but cannot specify the exact output vertices.

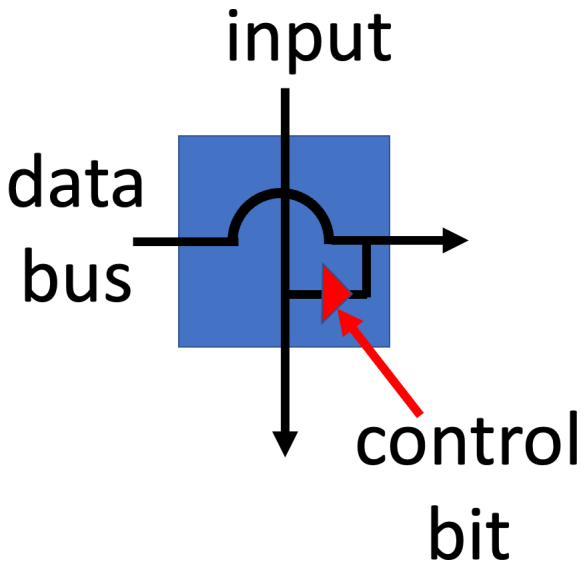


Figure 2.1: The internal structure of a crossbar switch. When the control bit is 1, the input is connected to the data bus. When the control bit is 0, the input is not connected to the data bus. For the rest of the figures in this chapter, the data bus is used as the network output and represented with horizontal lines in the figures. The inputs are represented with vertical lines. The crossbar switches are represented with blue squares. Control bits are not included in the figures but are implied to be there for each of the blue squares.

and one introduced in [56]. The first among these is a bipartite concentrator, called a full-capacity fat-and-slim crossbar, in which inputs are divided into two subsets of size $n - m$ and m. Each of the $n - m$ inputs is connected to all the outputs and forms the fat part of the concentrator, whereas each of the m inputs is connected to exactly one output and forms its slim part. These concentrators are often diagrammed using a sparse crossbar representation, as shown in Figure 2.2. As described, the left-hand side forms the fat part, and the diagonal line on the right-hand side forms the slim part of the concentrator. Adding all the crosspoints (edges) together, we find that a fat-and-slim (n, m)-concentrator consists of $(n - m)m + m = (n - m + 1)m$ crosspoints, and it was established in [47] that every bipartite (n, m)-concentrator must have at least $(n - m + 1)m$ crosspoints. Therefore, the full-capacity fat-and-slim concentrator is minimal with respect to the number of crosspoints.

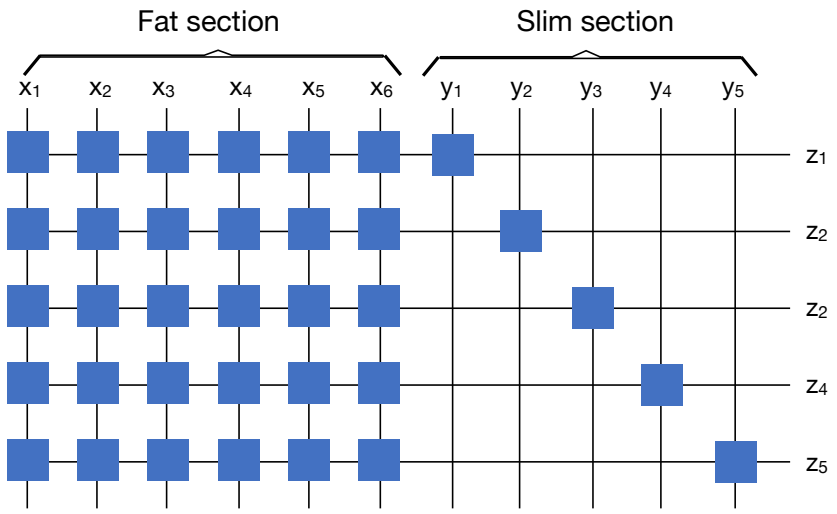


Figure 2.2: An (11, 5) full-capacity, fat-and-slim concentrator.

The second concentrator is also a sparse crossbar concentrator similar in concept to a full-capacity fat-and-slim concentrator but with a bounded capacity c as illustrated in Figure 2.3 for $n = 9$, $m = 7$, $c = 2$. In this construction, the left-hand side forms the slim part, whereas the right-hand side is added to provide a minimum capacity of c. This is ensured by requiring $c \leq m/c$ or $c \leq \sqrt{m}$. The key idea is to provide a sufficient number of crosspoints (edges) to each input in the right part so that if it gets blocked by as many as $c-1$ inputs from the slim part, it can still find an idle output to match with. Requiring $c \leq m/c$ secures this as each input in the right part is connected to $\lfloor m/c \rfloor$ outputs. The constriction also assumes that $n - m \leq c$. This assumption ensures that for every output, there is a crossbar switch at the slim section that connects to it. Otherwise, there would be outputs that no input connects, making such an output fruitless. It was shown in [47] that the crosspoint complexity of this construction remains with a factor of two of a lower bound of $\lfloor \frac{(n-c+1)m}{m-c+1} \rfloor$ crosspoints. We refer the reader to [47] for a more in-depth account of these two concentrator constructions, while we note that the capacity of the second sparse

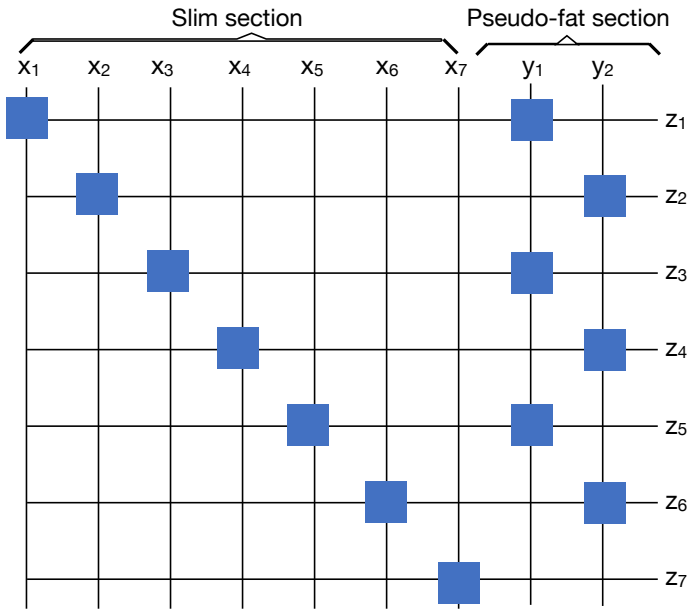


Figure 2.3: A bounded capacity fat-and-slim $(9, 7, 2)$-concentrator, which is also a $(9, 7, 3)$-concentrator by our extension of its capacity.

crossbar concentrator is actually larger than c when $m/c > c$. In fact, the capacity of this crossbar is $\lfloor m/c \rfloor$ as each input in the fact section is connected to at least $\lfloor m/c \rfloor$ outputs. We adjust our notation to highlight this observation by replacing c for the width of the inputs in the fat-section by q, and letting $n - m \leq q \leq m$, and $c = \lfloor m/q \rfloor$. Therefore, $c \leq \sqrt{m}$ is no longer required as described in [47]. For the rest of this chapter, this updated construction will be referred to as "bounded capacity fat-and-slim concentrator". Thus, we revise the capacity of the fat-and-slim crossbar in Figure 2.3 to 3.

We will also consider a third sparse concentrator that has a more regular structure [56]. This concentrator uses $n = pm$ inputs and m outputs, and it is derived from the full-capacity fat-and-slim concentrator. Each output is connected to $n - m + 1$ inputs as in the full-capacity fat-and-slim concentrator. On the other hand, each input is connected to between $m - \lfloor m/p \rfloor$ and $m - \lfloor m/p \rfloor + 1$ outputs, making this (pm, m)-concentrator nearly

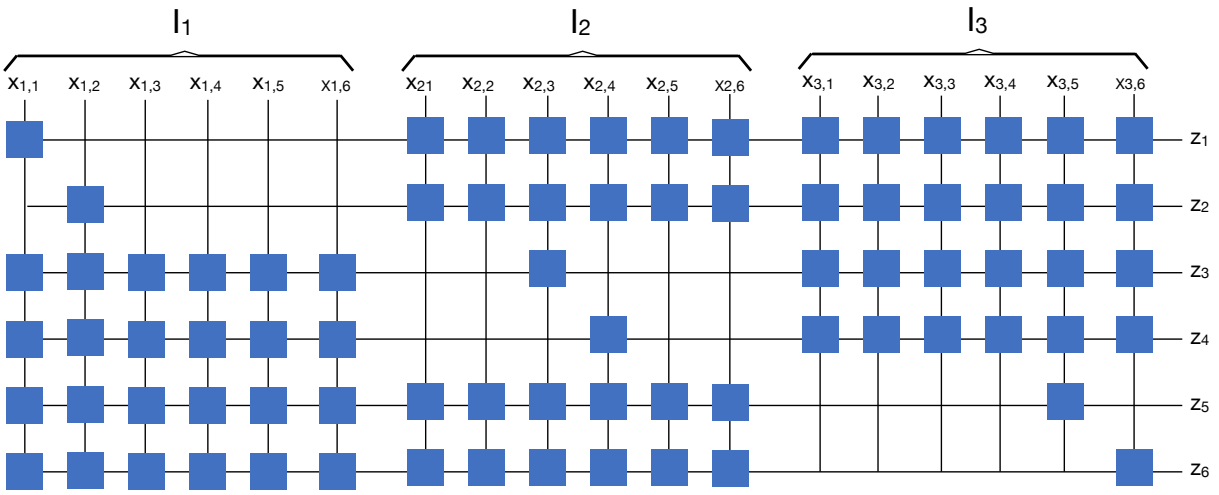


Figure 2.4: A regular (18,6)-fat-and-slim concentrator.

regular in terms of its in-degree (fan-in) as well. In this chapter, we will consider the case when p divides m.

Additionally, we only consider cases when $p \geq 3$. Figure 2.4 illustrates this construction for $p = 3$, and $m = 6$. It is seen that the out-degree of the construction falls between 4 and 5. Effectively, this construction is obtained from the full-capacity fat-and-slim concentrator by (i) dividing $n = pm$ columns of crosspoints into p sections, (ii) chopping the diagonal of m crosspoints in the slim part into p groups of m/p columns, and (iii) swapping each of those m/p columns with an equal number of columns in one of the remaining $p-1$ sections on the left. Using the notation in [56], we denote the sets of inputs in the p sections by $I_j, 1 \leq j \leq p$, and let $I_j = \{x_{j,1}, x_{j,2}, \cdots, x_{j,m}\}$. We further let $V_j = \cap_{k=1}^{m-1} N(x_{j,k})$, $U_j = O \backslash V_j$, and W_j denote the set of inputs that are connected to the outputs in U_j, where $N(x_{j,k})$ denotes the neighbor set of input $x_{j,k}, 1 \leq j \leq p, 1 \leq k \leq m$. In Figure 2.4, $V_1 = \{z_3, z_4, z_5, z_6\}, V_2 = \{z_1, z_2, z_5, z_6\}, V_3 = \{z_1, z_2, z_3, z_4\}, U_1 = \{z_1, z_2\}, U_2 = \{z_3, z_4\}, U_3 = \{z_5, z_6\}, W_1 = \{x_{1,1}, x_{1,2}\}, W_2 = \{x_{2,3}, x_{2,4}\}, W_3 = \{x_{3,5}, x_{3,6}\}$. The crux of

this construction is a transformation theorem proved in [56].

As we describe in the latter part of Section 2.3, the construction of this concentrator makes the design of a classical routing algorithm for it more involved, but we establish that the time complexity of such an algorithm remains $O(m)$ once active inputs are located.

2.2 Classical Versus Quantum Routing Model

The algorithms presented in this chapter will be run on two models of computation: (2.2.1) classical model and (2.2.2) quantum model. It is important to highlight the differences between these two models to make a justifiable comparison of their execution times.

2.2.1 The Classical Routing Model

In the classical model, we assume that quantities of interest are represented using classical bits of 0 and 1. For example, we use $\log_2 n$ bits and $\log_2 m$ bits to identify the inputs and outputs of an (n, m)-concentrator, respectively. We further assume that the bits within the representation of each input and/or output can be processed in parallel. For example, we can inspect all of the bits within a representation of any input (output) in parallel to determine if it is a particular input we seek, or we can compare the bits in representations of any two inputs (outputs) or a constant number of inputs (outputs) in parallel to see if they are the same in $O(1)$ time. Such bit-level operations can generally be carried out by logic circuits that consist of elementary logic components such as OR, AND, XOR, XNOR, and NOT gates that we assume have $O(1)$ computation time, and have a constant fan-in and fan-out, i.e., they have a constant number of inputs and outputs. We note that two or

a constant number of $\lg n$-bit numbers can be added, subtracted or compared using $O(\lg n)$ 2-input, 2-output logic gates in $O(\lg \lg n)$ time using a prefix-adder as described in [64]. This $O(\lg \lg n)$ time will be suppressed in our time complexity formulas, leaving us with $O(1)$ time. It will also be assumed that $O(\lg n)$-bit operands can be written and read in and out of a random access memory in $O(1)$ time. We will be using an array named *in* to specify if a given input is active. We will assume that *in* is an array that is stored in a random access memory with $O(1)$ access time. It is possible to relax $O(1)$ access time complexity. However, since such a non-constant access time complexity would be a multiplicative coefficient in the time complexity of the bottleneck step of both classical and quantum algorithms, we will omit this factor from our complexity calculations, effectively assume that the access time of *in* is $O(1)$.

2.2.2 The Quantum Routing Model

In the quantum routing model, classical bits are replaced by quantum bits (qubits), and classical logic gates are replaced by those that represent unitary transformations. We assume that unitary gates have one or two inputs. All together, we allow $O(\lg n)$ qubits in the quantum routing model. This allows us to work with possibly up to n states in parallel. We further allow any combination of quantum gates to be used on these $O(\lg n)$ qubits, but with the restriction that each qubit can source only one quantum gate at a time. This last restriction is a corollary of the no-cloning theorem [65]. This fanout restriction of one in the quantum routing model is not imposed on the bits in the classical routing model. However, we still assume that the fanout of each input is $O(1)$ in the classical model as well.

Therefore, the two models can be viewed to be analogous, where any quantum operation on mutually exclusive $O(\lg n)$ qubits takes $O(1)$ time much the same way each operation on $O(\lg n)$ classical bits takes $O(1)$ time. The difference lies in the amount of parallelism afforded by the two models: in the classical model, we assume that the parallelism is limited to operations on any given pattern of $O(\lg n)$ bits, whereas in the quantum routing model, the parallelism transcends any particular or fixed pattern as quantum operations are applied to the totality of the quantum state that includes all $O(n)$ binary patterns of $O(\lg n)$ qubits. This vast amount of parallelism inherently present in quantum mechanical systems resulted in Shor's quantum prime number factorization algorithm [66], which is exponentially faster than the best-known classical algorithm. Another key quantum algorithm, Grover's quantum search [15] provides a speed-up of $O(\sqrt{n})$ over a sequential algorithm to search an element in an unordered list of n elements. The latter algorithm will be used in Section 2.4 to reduce the routing time of concentration assignment on fat-and-slim concentrators using quantum algorithms. Before we describe these algorithms, we provide their classical analogs in the next section.

Routing an assignment on a sparse crossbar concentrator amounts to constructing a matching between any given subset of inputs and some subset of outputs of equal cardinality. The particular topologies of the three concentrators given in Section 2.1 guide the design of a routing algorithm for each concentrator as we describe next.

2.3 Classical Routing On Concentrators

2.3.1 Full-Capacity Fat-Slim (F-S) Concentrator Routing

Algorithm 1 Classical Full-Capacity F-S Concentration

```
1: function Classical Full FS Route(in, n, m)
   //in: n-bit array that marks up to m active inputs
   //n: number of inputs.
   //m: number of outputs.
2:     L ← list()
3:     for i ← 1 : m do //slim section
4:         if in[m − n + i] == 1 then
5:             pair(y_i, z_i);
6:         else
7:             L.insert(z_i);
8:         end if
9:     end for
10:    for i ← 1 : n − m do //fat section
11:        if in[i] == 1 then
12:            {z = L.remove(); pair(x_i, z);}
13:        end if
14:    end for
15: end function
```

Our first classical algorithm, Algorithm 1, is a restatement of the algorithm that was originally described in [48] for a full-capacity fat-and-slim concentrator. We recall from Section 2.1 that the set of inputs is partitioned into two sets: those in the fat section: $X = \{x_1, x_2, \cdots, x_{n-m}\}$ and those in the slim section: $Y = \{y_1, y_2, \cdots, y_m\}$. The set of outputs of the concentrator is denoted by $Z = \{z_1, z_2, \cdots, z_m\}$.

As we stated in the earlier section, the active inputs in a routing request, i.e., those to be concentrated, are specified by '1' entries in an n-bit array, named in. A routing request with k active inputs is first completed to a request with m inputs by combining the leftmost unused $m - k$ inputs in in with the given k active inputs. Moreover, the leftmost $m - k$

unused inputs in in can be determined in $O(m)$ time by examining the leftmost m inputs of in.

Effectively, the first for loop assigns each active input in the slim section to the output with the same index value, while also inserting each unused output z_i into a list L of m elements as it checks if input $y_i, 1 \leq i \leq m$ is active. The second for loop then assigns the active inputs from left to right in the fat section to the unused outputs, i.e., those that are inserted into L from top to bottom.

The time complexity of Algorithm 1 is easily seen to be $O(n)$ steps, given that (i) each iteration in the first loop involves checking if a bit in the list in is '1', pairing an input in the slim-section with an output or inserting a value into a list of m elements, and (ii) each iteration in the second loop involves checking a bit, pairing an input in the fat-section with an output, and removing an output from the list L. Checking if $in[i] == 1$ clearly takes $O(1)$ time. It is further assumed that pairing an input and output as well as inserting or removing a value in and out of a list also takes $O(1)$ time. This is a reasonable assumption, considering that all three operations involve no more than a basic memory read or write operation.

As an example, let in=$[0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0]$, $n = 11$ and $m = 5$. This implies that set of active inputs are $\{x_2, x_4, y_1, y_2, y_4\}$. On line 5, pairings are $\{(y_1, z_1), (y_2, z_2), (y_4, z_4)\}$. This gives us $L = \{z_3, z_5\}$ at the end of line 9. On line 12, pairings are $\{(x_2, z_3), (x_4, y_5)\}$.

Algorithm 2 Classical Bounded Capacity F-S Concentration

```
1: function Classical Bounded F-S Route(in, n, m, q)
   //in: n-bit array that marks up to c active inputs.
   //n: number of inputs.
   //m: number of outputs.
   //q: the width of the fat section.
   //c = ⌊m/q⌋: capacity.
   //out: m-bit array that marks available outputs (initialized to all 1's).
2:     for i ← 1 : n − q do //slim section
3:         if in[i] == 1 then
4:             pair(x_i, z_i); out[i] ← 0;
5:         end if
6:     end for
7:     L ← list() //pseudo-fat section
8:     for i ← 1 : q do
9:         if in[i + m] == 1 then
10:            L.insert(i);
11:        end if
12:    end for
13:    while i ← L.remove() do
14:        for j ← 0 : c − 1 do
15:            if out[i + qj] == 1 then
16:                pair(y_i, z_{i+qj});
17:                break
18:            end if
19:        end for
20:    end while
21: end function
```

2.3.2 Bounded Capacity Fat-Slim Concentration Routing

Algorithm 2 extends the main idea of Algorithm 1 to routing active inputs in a bounded capacity fat-and-slim concentrator[2]. As in Algorithm 1, the inputs are divided into fat and slim sections, but this time, the slim section is placed on the left, and the pseudo-fat section is placed on the right in Figure 2.3. An m-bit array, named out, is added to mark the outputs. The first for loop pairs the active inputs in the slim section with outputs whose indices coincide with those of the active inputs while marking those outputs by entering '0's into out in their index positions. The second for loop finds the active inputs in the pseudo-fat section on the right. The last nested loop pairs the active inputs in the pseudo-fat section on the right by searching for an available output among the set of outputs to which each active input is connected by a crosspoint.

Algorithm 2 has an execution time of $O(n) = O(m)$.

1. Lines 2-6 take $O(m)$ time as they involve steps to check if $in[i] = 1$, pair x_i with z_i, and clear a bit in the *out* array.

2. Lines 7-12 take $O(q)$ time as they involve checking q array elements and adding them to a list.

3. Lines 13-20 consist of a nested loop that takes $O(c)$ time. To see this, let $c', 0 \leq c' \leq c$, be the number of active inputs found in lines 2-6. In line 17, we exit the interior for loop. This happens when available output is found for that input. The number of

[2]Even though we refer to this construction as a bounded capacity, fat-and-slim concentrator, the fat section on the right is not completely filled with crosspoints as in the case of the full-capacity fat-and-slim concentrator.

unsuccessful checks for an available output can at most be c', and successful checks for such an output can at most be c. Since the number of elements in the list L is also bounded by c, this nested loop takes $O(c)$ time.

As an example, let $in = [0, 1, 0, 1, 0, 0, 0, 0, 1]$, $n = 9$, $m = 7$ and $c = 3$. This implies that set of active inputs is $\{x_2, x_4, y_2\}$. On line 4, pairings are $\{(x_2, z_2), (x_4, x_4)\}$. This leaves us with only 1 unpaired active input, y_2. y_2 is connected to z_2, z_4 and z_6. On line 4 outputs z_2 are z_4 used. Therefore on line 16 the paring is (y_2, z_6).

2.3.3 Regular Fat-Slim Concentration Routing

Our third classical algorithm restates the one given in [48]. However, for some steps of the algorithm, we prove tighter time complexities than provided in [48], which are useful when we introduce the quantum counterpart. For this sparse crossbar concentrator, we have a construction with $n = pm$ inputs and m outputs as described in Section 2.1, where we assume that p divides m. A routing request with k active inputs is first completed to a request with m inputs by combining the first unused $m - k$ inputs in I_1 with the given k active inputs. This is always possible since if all $k \leq m$ active inputs belong to I_1; then its remaining $m - k$ inputs can be combined with the k active inputs to obtain an m-request. On the other hand, if only some $k' < k \leq m$ of the k active inputs belong to I_1, then I_1 must have $m - k' > m - k$ unused inputs, the first $m - k$ of which can be combined with the given k active inputs to obtain an m-request. Moreover, the first $m - k$ unused inputs in I_1 can be determined in $O(m)$ time by examining all m inputs in I_1 in the worst case[3].

[3]The selection of I_1 is arbitrary and simplifies our description for the completion of a k-assignment to an m-assignment. The algorithm will work regardless of which set of inputs is selected.

Therefore, we will assume that a k-request is completed to an m-request and specified by an array of n-bits named in, in which 0 and 1 bits represent the absence and presence of an active input, respectively. At the beginning of the algorithm, the locations of these active inputs are searched, and active inputs in $I_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,m}\}$ are placed in a linked list $R_j, 1 \leq i \leq p$.

Algorithm 3 routes an m-request by considering two distinct cases: (a) One of the R_j's has more than $m - m/p$ active inputs. (b) None of R_j's has more than $m - m/p$ active inputs. That there can be no other case for all $p \geq 2$ is shown as follows. Suppose there exists R_j with more than $m - m/p$ active inputs for some j. Then there remain less than $m - (m - m/p) = m/p$ active inputs in the union of all the remaining $p - 1$ sets of m inputs. Therefore, another m-bit array, i.e., $R_{j'}$ for some $j' \neq j, 1 \leq j' \leq p$ with more than $m - m/p$ active inputs exists only if $m/p > m - m/p$, which implies $p < 2$ or $p = 1$, contradicting our assumption that $p \geq 2$ as stated in Section 2.1.

Now, continuing with Algorithm 3, lines 1 through 8 construct the linked lists in $O(n)$ time, assuming that we have m active inputs as described above and determine the number of entries in each linked list by incrementing a size field each time an active input is found. Next, we see that the for statement in line 9 is iterated at most p times, which occurs either when none of $R_j, 1 \leq j \leq p$ has more than $m - m/p$ active inputs or R_p does. During the jth iteration, the if statement in line 10 checks if R_j has more than $m - m/p$ 1's. The size of each set can be queried in $O(1)$ time with the help of the size field that has been computed in line 5.

Case (a): If there exists such an R_j then line 11 pairs all the active inputs in $R_j \cap W_j$, if any, and the outputs in U_j to which those active inputs are connected by crosspoints.

Algorithm 3 Classical Regular F-S Concentration

1: function Classical Regular F-S Route(in, p, m)
   //in: n-bit array that marks up to m active inputs.
   //$n = m * p$: number of inputs.
   //$R_j, 1 \leq j \leq p$: linked lists of active inputs.
   //m: number of outputs.
   //The size fields of $R_j, 1 \leq j \leq p$ are cleared.
2: for $j \leftarrow 1 : p$ do
3: for $i \leftarrow 1 : m$ do //find active elements
4: if $in[(j-1)m + i] == 1$ then
5: $R_j.insert(i); R_j.size{+}{+}$
6: end if
7: end for
8: end for
9: for $j \leftarrow 1 : p$ do
10: if $R_j.size > m - m/p$ then //case (a)
11: Pair inputs in $R_j \cap W_j$ with outputs in U_j;
12: Pair inputs in $\cup_{i \neq j} R_i$ with unpaired outputs in U_j;
13: Pair unpaired inputs in R_{j-1} with outputs in U_{j+1};
14: Pair unpaired inputs in $\cup_{i \neq j, i \neq j-1} R_i$ with outputs in U_{j-1};
15: Pair unpaired inputs in $R_j$with unpaired outputs in V_j;
16: return
17: else //case (b)
18: $a[j] \leftarrow (m/p \leq |R_j|)$; //$p$-bit array for reindexing
19: end if
20: end for
21: $r \leftarrow$ prefixSum(a); $s \leftarrow$ prefixSum$(\neg a)$;
22: $d[j] \leftarrow a[j]r[j] + (\neg a[j])(s[j] + r[p]); 1 \leq j \leq p$; //new indices
23: $R'_{d[j]} \leftarrow R_j$; $U'_{d[j]} \leftarrow U_j, 1 \leq j \leq p$;
24: for $j \leftarrow 2 : r[p]$ do
25: Take first m/p elements from R'_j and place them in P'_j;
26: Pair inputs in P'_j with outputs in U'_{j-1};
27: $Q'_j \leftarrow R'_j \backslash P'_j$;
28: end for
29: Pair inputs in $R'_{r[p]+1}, R'_{r[p]+2}, ..., R'_p, R'_1, Q_2, Q_3, ..., Q_{r[p]}$ with outputs in $U'_{r[p]}, U'_{r[p]+1}, ..., U'_p$;
30: end function

Line 12 then pairs as many active inputs in $R_i, 1 \leq i \neq j \leq p$ as possible, if any, with unused outputs in U_j. At this point, any active remaining inputs in $R_i, 1 \leq i \neq j \leq p$, are paired with unused outputs in $U_l, l \neq j$ in lines 13 and 14. This pairing is implemented in such a way that one of the $U_i, 1 \leq i \neq j$ is arbitrarily fixed (U_{j-1} in the algorithm) to start the pairing. It is possible that U_{j-1} may not even have an active input, but line 13 serves to initiate pairing of the active inputs in the remaining subsets of inputs other than I_j. We note that the number of active inputs in I_{j-1} is less than $m/p = |U_{j+1}|$ and a full crossbar connection exists between I_{j-1} and U_{j+1}. Similarly, in line 14, it is always possible to pair the number of active inputs in $\bigcup_{i \neq j, i \neq j-1} I_k$ with the outputs in U_{j-1} as U_{j-1} has m/p available outputs, which are more numerous than the active inputs in $\bigcup_{i \neq j, i \neq j-1} I_k$, and a full crossbar connection exists between the two sets. Here, the indexing is cyclical, and it is assumed that j, $j-1$, and $j+1$ are distinct from each other and $p \geq 3$. If $p = 2$ then line 14 is not needed. Finally, any remaining inputs in R_j are paired with the remaining unused outputs in V_j in line 15 as there is a full crossbar connection between the inputs in I_j and outputs in $V_j, 1 \leq j \leq p$.

These steps are illustrated in Figure 2.5, where the circled numbers identify the steps in the algorithm. In this example, lines 13 and 14 do not result in any pairing. As another example, let $m = 12, p = 3$, $R_1 = \{1, 2, 3, 5, 6, 7, 8, 9, 10\}$, $R_2 = \{3, 8\}$, $R_3 = \{1\}$. We have $U_1 = \{z_1, z_2, z_3, z_4\}$, $U_2 = \{z_5, z_6, z_7, z_8\}, U_3 = \{z_9, z_{10}, z_{11}, z_{12}\}$, and $|R_1| = 9 > 8 = m - m/p$. Thus, $x_{1,1}, x_{1,2}, x_{1,3}$ are paired with z_1, z_2, z_3 in U_1 in line 11, $x_{2,3}$ is paired with z_4 in U_1 in line 12, $x_{3,1}$ in I_3 is paired with output z_5 in U_2 and the remaining active input $x_{2,7}$ in I_2 is paired with output z_9 in U_2 in lines 13 and 14. Finally, $x_{1,5}, x_{1,6}, x_{1,7}, x_{1,8}, x_{1,9}, x_{1,10}$ are paired with the remaining outputs all of which belong to V_1. We established that any

routing request in case (a) can always be realized in lines 11 through 15. Now suppose that the condition $|R[j]| > m - m/p$ fails for all $j, 1 \leq j \leq p$. The remaining part of the algorithm handles case (b) as described next.

Case (b): First, in line 17, we initiate a process to identify the subsets of inputs with at most m/p active inputs. This is done to reindex the sets of inputs $I_j, 1 \leq j \leq p$ so that those that have between m/p and $m - m/p$ active inputs are given lower index values. Effectively, the R_j's and the corresponding U_j's are implicitly sorted in descending order, based on whether they contain more than the threshold of m/p active inputs. To facilitate this, we follow the approach in [48] and form a bit-array of p elements, a in line 18 such that $a[j] = 1$ if and only if $m/p \leq |R_j| \leq m - m/p, 1 \leq j \leq p$. This array is used to split R_j into two groups, those for which $m/p \leq |R_j| \leq m - m/p$ and those for which $|R_j| < m/p$. This splitting is carried out by computing the prefix sums of the bits in a and $\neg a$ into two p-element arrays r and s in line 21, where r ranks those R_j for which $m/p \leq |R_j| \leq m - m/p$, and s ranks those R_j for which $|R_j| < m/p$. The ranks index the active inputs in each group separately. The ranks are threaded together in line 22 to obtain a p-element array, d, which represents a permutation of the indices of R_j. This step essentially involves selecting one of $r[j]$ or $r[p] + s[j]$ into $d[j]$ and is used to compute the indices of R_j's. It is not difficult to see that d is a permutation of the indices of R_j's and applying d to the indices of R_j's and U_j's amounts to renaming them so that R_j becomes $R_{d[j]}$ and U_j becomes $U_{d[j]}, 1 \leq j \leq p$. For clarity, we replace $R_{d[j]}$ and U_j by $R'_{d[j]}$ and U'_j in the algorithm.

As an example, let $m = 20, p = 5$, and suppose that I_1, I_2, I_3, I_4, I_5 have $3, 4, 6, 5, 2$ active inputs, respectively. Given that $m/p = 5, m - m/p = 20 - 20/5 = 16$, $|I_1|, |I_2|, |I_5| <$

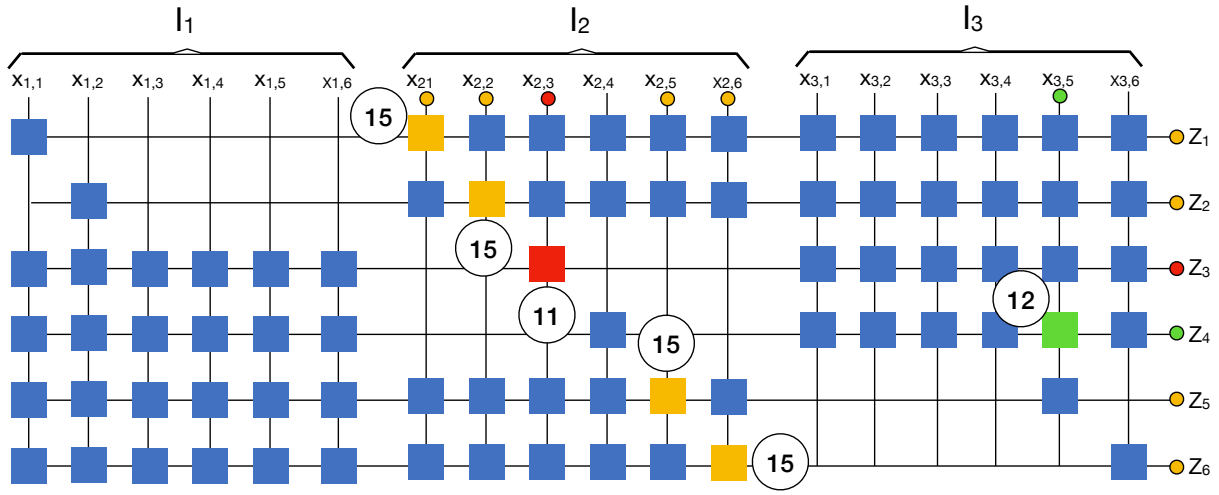


Figure 2.5: Routing in a regular (18,6)-fat-and-slim concentrator.

m/p and $m/p < |I_3|, |I_4| \leq m - m/p$ so that $a = [0,0,1,1,0], \neg a = [1,1,0,0,1]$, $r = [0,0,1,2,2], s = [1,2,2,2,3], d = ar + (\neg a)(r[p]+s) = [0,0,1,2,0] + [3,4,0,0,5] = [3,4,1,2,5]$. Thus, $R'_3 \leftarrow R_1, U'_3 \leftarrow U_1, R'_4 \leftarrow R_2, U'_4 \leftarrow U_2, R'_1 \leftarrow R_3, U'_1 \leftarrow U_3, R'_2 \leftarrow R_4, U'_2 \leftarrow U_4, R'_5 \leftarrow R_5, U'_5 \leftarrow U_5$. This gives a permuted set of registers in descending cardinalities, i.e., $R'_3, R'_4, R'_1, R'_2, R'_5$ and $U'_3, U'_4, U'_1, U'_2, U'_5$.

Once this sorting process is completed in line 23, we select the first m/p active inputs in each of the $r[p]$ sets in which the number of active inputs lies between m/p and $m - m/p - 1$, and store all, but the first m/p active inputs into $P'_j, 2 \leq j \leq r[p]$ in line 25. These active inputs are paired with sets of m/p outputs in $U'_{j-1}, 2 \leq j \leq r[p]$ in line 26 using the full crossbar connection between them. This will complete the pairing of $(r[p]-1)m/p$ active inputs and $m - (r[p]-1)m/p = (p - r[p] + 1)m/p$ active inputs remain.

The remaining active inputs in R'_j are saved into $Q'_j, 2 \leq j \leq r[p]$ in line 27. Finally, the active inputs in Q_j's are concatenated with the active inputs in $R_{r[p]+1}, R_{r[p]+2}, ..., R_{[p]}, R_1$ which are then paired together with the outputs in sets $U'_{r[p]+1}, U'_{r[p]+2}, ..., U'_p$ in line 29.

Note that the number of active inputs in $R'_{r[p]+1}, R'_{r[p]+2}, ..., R'_p, R'_1, Q_2, Q_3, ..., Q_{r[p]}$ is given by $(p-r[p]+1)m/p$ and it must match the number of active outputs in in $U'_{r[p]}, U'_{r[p]+1}, ..., U'_p$ after the pairing in line 26.

Algorithm 3 has a time complexity of $O(n)$ as shown below.

1. Lines 2-8: This is a linear search of active inputs out of n inputs, and therefore has a time complexity of $O(n)$.

2. Lines 9-29: By the time we get to these lines, we already know where the active inputs are located because R_j's consist only of active inputs. At this point the only task left to do is to route these active inputs. We route in two different ways as described in cases (a) and (b) in the algorithm. Both take $O(m)$ time as described below.

   (a) Lines 9 and 10: We see that the for statement here is iterated at most p times, which occurs either when none of $R_j, 1 \leq j \leq p$ has more than $m - m/p$ active inputs or R_p does. During the jth iteration, the if statement in line 10 checks if R_j has more than $m - m/p$ elements. Since we can query the size of sets in $O(1)$ time and comparison of values in the if statement can also be completed in $O(1)$ time, this line has a time complexity of $O(p)$.

   (b) Lines 11-16: In case (a), there are more active inputs than non-diagonals can handle alone, and therefore diagonal crosspoints must be used. These lines execute at most once and take $O(m)$ time as explained below.

      i. Line 11 can be done by iterating through R_j and checking if the current element is in W_j. Since W_j is an interval where endpoints are fixed and

known, the membership decision amounts to the comparison of the current element against the two endpoints, and this takes $O(1)$ time in our classical bit-parallel processor model. Moreover, the number of pairing operations is clearly bounded by $|W_j| = |U_j| = m/p$, and pairing an active input with the corresponding output in U_j takes $O(1)$ time. Therefore, this line can be done in $O(\max(m, m/p)) = O(m)$ time, the size limit of R_j.

ii. Line 12: Let $\hat{R} = \cup_{i \neq j} R_i$. Given that all inputs in $\hat{R}$ are connected to outputs in U_j by a full crossbar connection, the active inputs in $\hat{R}$ can be paired with unused outputs in U_j by scanning $\hat{R}$ from left to right and U_j from top to bottom until all unused outputs in U_j are paired. As we represent $R_j, 1 \leq j \leq p$ by linked lists and since there are m active inputs at the most, the pairing can be completed in $O(m)$ time in this case.

iii. In line 13 we pair two sides of a full crossbar connection, i.e., inputs in R_{j-1} and outputs in U_{j+1}. We loop through these sets, removing one element from each to build a complete matching. Given that $|R_{j-1}| \leq m, 2 \leq j \leq m+1$, and $|U_{j+1}| = m/p$, this line has a time complexity of $O(m)$.

iv. In line 14, we proceed as in line 12, and note that $|\cup_{i \neq j, i \neq j-1} R_i| \leq m$. Therefore this line has a time complexity of $O(m)$.

v. Line 15 involves connecting the remaining active inputs in I_j and the unused outputs in V_j. It is not difficult to verify that this step can be completed in $O(m - m/p)$ time as there is a full crossbar connection between the inputs in R_j and $V_j, 1 \leq j \leq p$. It follows that the time complexity of the for loop

is $O(m - m/p)$.

(c) Lines 17-29: These lines are carried out in case (b) only. In this case, non-diagonal crosspoints can handle the assignment without diagonal crosspoints. Therefore, only the fat sections are used in the routing and these lines can be done in $O(m)$ time as explained below.

i. Line 18 involves querying size of R_j and comparing it to m/p, which can be completed in $O(1)$ time per each iteration of the for loop and in $O(p)$ time for all iterations.

ii. Line 21 has prefix sum of p elements that can be done in $O(p)$ time by an iterative summation of p bits.

iii. Line 22 involves an addition of two $O(\lg p)$-bit numbers and a 2×1 multiplexer, both of which take $O(1)$ time and so the loop in this line takes $O(p)$ time.

iv. Line 23 involves reindexing of R_j's, which clearly takes $O(p)$ time.

v. Lines 24-27 can be carried out in $O(m/p)$ time per each set of m/p inputs and in $O(r[p] \times m/p) = O(m)$ time for all $r[p]$ sets of m/p inputs.

vi. Line 29 can be done by pairing the elements between the two sets described in these lines one by one in $O(m)$ time.

Remark. Algorithm 1, Algorithm 2 and Algorithm 3 are all optimal in terms of order of execution time as it takes $\Omega(n)$ time to read a concentration assignment. □

2.4 Quantum Routing with Grover's Search

Now that we have established that classical routing algorithms all take $O(n)$ time for full and bounded capacity fat-and-slim concentrators described in Section 2.1, we turn our attention to quantum routing for them. Since both these concentrators require searching of active inputs out of all inputs, Grover's search [15] provides a possible approach to identify such inputs faster in the quantum domain. Grover's search approach has been followed to speed up various matching and network flow problems in [43] and [45]. These two papers were among the inspirations for our approach. We will use the version of Grover's search analyzed in Theorem 3 of [18] and refer to it as GroverSearch in this chapter. We note that GroverSearch is the only quantum step in the algorithms to be presented in this section and the rest of the chapter. The time complexity of GroverSearch is known to be $O(\sqrt{n/k})$ to find one of k marked items out of n possible items. Thus, it takes $O(\sqrt{nk})$ time to discover all k items by unmarking discovered items one at a time. This is a widely used method even though the original source is unknown. The proof can be found in Lemma 4.1 of [67].

Remark. As in [43] and [45], we use Monte Carlo amplification [68] to bound the total error of the entire algorithm resulting in an increase in run-time that is a logarithmic factor in the number of calls to the quantum subroutine, i.e., $\ln k$. Therefore, the time to discover all marked inputs is $O(\sqrt{nk}\ln k)$. □

Algorithm 4 Quantum Full F-S Concentration

```
1: function Quantum Full FS Route(in, n, m)
   //in: n-bit array that marks up to m active inputs
   //n: number of inputs.
   //m: number of outputs.
2:     L ← list()
3:     for i ← 1 : m do //slim section
4:         if in[m − n + i] == 1 then
5:             pair(y_i, z_i);
6:         else
7:             L.insert(z_i);
8:         end if
9:     end for
10:    while i ← GroverSearch(in[1 : n − m]) do//fat section
11:        z = L.remove(); pair(x_i, z);
12:        in[i] = 0
13:    end while
14: end function
```

2.4.1 Quantum Full-Capacity F-S Concentration

The pseudo-code of our routing algorithm for the full-capacity fat-and-slim concentrator is given in Algorithm 4. As in the classical case, this algorithm consists of two parts. In the first part, we route all the active inputs to corresponding outputs in the slim section. We also make a list of idle inputs in this part to identify the corresponding outputs that are not used by the inputs in the slim section. Such outputs are then paired with the active inputs that are found by GroverSearch in the second part of the algorithm. This part of the algorithm is classical and deterministic. The second part of the algorithm provides us with quantum speedup. We use GroverSearch to locate the active inputs in the fat part of the concentrator. Once an active input is found, we pair it with one of the idle outputs that were found in the first part and unmark that input for another round of GroverSearch.

The time complexity of Algorithm 4 is computed as follows. First, note that the slim

part of the algorithm is classical and deterministic. Since we assume that each query of the array in requires a constant time, identifying the active inputs, assigning them to their respective outputs, and also identifying the unused outputs in the slim part takes $O(m)$ time. Now, let k be the number of active inputs in the fat part. It is obvious that k is upper bounded by m. Hence by Remark 2.4, we find that the total time required for the fat part is $O(\sqrt{m(n-m)}\ln m)$. Therefore the total time complexity of concentration in a full-capacity, fat-and-slim (n,m)-concentrator is $O(m+\sqrt{k(n-m)}\ln m) = O(m+\sqrt{nm}\ln m)$, and hence, given that $n \geq m$, the total time complexity is $O(\sqrt{nm}\ln m)$, using a quantum processor with $O(\ln n)$ qubits. For comparison, any classical-bit processor routing algorithm for a full capacity, fat-and-slim (n,m)-concentrator takes $\Theta(n)$ time as we established in the earlier section (See Remark 2.3.3). When $n = \Theta(m\ln^2 m)$, we have $O(\sqrt{nm}\ln m) = O(n)$. Therefore, our algorithm performs better in the quantum domain than any known classical algorithm if $m\ln^2 m = o(n)$. One such value of m is clearly $\Theta(\sqrt{n})$. In general, it can easily be shown that $m = n^{\mu}$, satisfies $m\ln^2 m = o(n)$, for any $\mu, 0 < \mu < 1$, and the time complexity of our algorithm will increase less than linearly with n for such values of m.

2.4.2 Quantum Bounded-Capacity F-S Concentration

The time complexity of the classical routing algorithm for a bounded capacity fat-and-slim concentrator can also be reduced using quantum search as well, as described in Algorithm 5. The slim section of the algorithm finds at most c marked elements out of at most m total elements since $n - q \leq m$. Similarly, the fat section of the algorithm also finds c marked elements out of at most m total elements since $q \leq m$. The total time complexity of both

Algorithm 5 Quantum Bounded FS Concentration

```
1: function Quantum Sparse FS Route(in, n, m, q)
   //in: n-bit array that marks up to c active inputs.
   //n: number of inputs.
   //m: number of outputs.
   //q: the width of the fat section.
   //c = ⌊m/q⌋: capacity.
   //out: m-bit array that marks available outputs (initialized to all 1's).
2:     while i ← GroverSearch(in[1 : n − q]) do //slim section
3:         pair(x_i, z_i);
4:         in[i] ← 0;
5:         out[i] ← 0;
6:     end while
7:     L ← list(); //pseudo-fat section
8:     while i ← GroverSearch(in[n − q + 1 : n]) do //slim section
9:         L.insert(i);
10:        in[i] ← 0;
11:    end while
12:    while i ← L.remove() do
13:        for j ← 0 : c − 1 do
14:            if out[i + qj] == 1 then
15:                pair(y_i, z_{i+qj});
16:                break
17:            end if
18:        end for
19:    end while
20: end function
```

parts is $O(\sqrt{mc}\ln c)$. Lines 12-19 is the same as 13-20 of Algorithm 2. Therefore, the total time complexity is also $O(\sqrt{mc}\ln c) = O(\sqrt{nc}\ln c)$. Once again, this algorithm perform better than it's classical counterpart when $c\ln^2 c = o(n)$.

Remark. In full capacity F-S concentrator, active inputs for the dense part are located with Grover's Search while for bounded capacity FS concentrator, all active inputs are located with Grover's Search. □

2.4.3 Quantum Regular FS Concentration

Our last quantum algorithm handles concentration assignments for regular fat-and-slim concentrators. It is obtained by replacing the search for active inputs by GroverSearch. From the analysis in subsection 2.3.3 we know that once the active inputs are found, the routing takes $O(m)$. Just like full capacity fat-and-slim concentrator, the search as well as the total time takes $O(\sqrt{nm}\ln m)$. Therefore the analysis done in subsection 2.4.1 also applies here.

Remark. It should be added that, in both classical and quantum domains, the search process can be speeded up by replicating controller resources. In the case of a classical search algorithm, using k controllers results in a factor of k reduction in time complexity from $O(n)$ to $O(n/k)$. On the other hand, in the quantum domain, this reduction in time will be limited to a factor $\sqrt{k}$. Thus, the separation in time complexities between the classical and quantum domains becomes more pronounced only when k is small.

Algorithm 6 Quantum Regular F-S Concentration

1: function Quantum Regular F-S Route(in, p, m)
   //in: n-bit array that marks up to m active inputs.
   //$n = m * p$: number of inputs.
   //$R_j, 1 \leq j \leq p$: linked lists of active inputs.
   //m: number of outputs.
   //The size fields of $R_j, 1 \leq j \leq p$ are cleared.
2: while $(i, j) \leftarrow GroverSearch(in)$ do
3: $R_j.insert(i); R_j.size{+}{+}$
4: end while
5: for $j \leftarrow 1 : p$ do
6: if $R_j.size > m - m/p$ then //case (a)
7: Pair inputs in $R_j \cap W_j$ with outputs in U_j;
8: Pair inputs in $\cup_{i \neq j} R_i$ with unpaired outputs in U_j;
9: Pair unpaired inputs in R_{j-1} with outputs in U_{j+1};
10: Pair unpaired inputs in $\cup_{i \neq j, i \neq j-1} R_i$ with outputs in U_{j-1};
11: Pair unpaired inputs in R_j with unpaired outputs in V_j;
12: return
13: else //case (b)
14: $a[j] \leftarrow (m/p \leq |R_j|)$; //$p$-bit array for reindexing
15: end if
16: end for
17: $r \leftarrow$ prefixSum(a);
18: $d[j] \leftarrow a[j]r[j] + (\neg a[j])(s[j] + r[p]); 1 \leq j \leq p$; //new indices
19: $R'_{d[j]} \leftarrow R_j$
20: for $j \leftarrow 2 : r[p]$ do
21: Take first m/p elements from R'_j and place them in P'_j;
22: Pair inputs in P'_j with outputs in U'_{j-1};
23: $Q'_j \leftarrow R'_j \backslash P'_j$;
24: end for
25: Pair inputs in $R'_{r[p]+1}, R'_{r[p]+2}, ..., R'_p, R'_1, Q_2, Q_3, ..., Q_{r[p]}$
26: with outputs in $U'_{r[p]}, U'_{r[p]+1}, ..., U'_p$;
27: end function

2.5 Conclusions and Future Work

We have presented three quantum algorithms, all with smaller time complexity as compared to their classical counterparts. For full-capacity fat-and-slim concentrator, our quantum algorithm has a time complexity of $O(\sqrt{nm}\log m)$ versus the classical routing algorithm complexity of $O(n)$. Thus, in this case, our quantum algorithm has a smaller time complexity if $m = n^{\mu}, 0 < \mu < 1$. For regular fat-and-slim and bounded-capacity fat-and-slim concentrators, similar speed-up formulas apply as established in the chapter. The main takeaway from these proofs is that network that has a quantum controller can be asymptotically superior to a network that has a purely classical controller. After all, a quantum controller computes the optimal resource utilization of these classical networks in a more time-efficient way than a classical controller can. Quantum computing is not only useful to find the optimal solutions to these types of combinatorial problems but it is also useful to find the approximately optimal solutions. The next three chapters discuss quantum algorithms for approximate optimization.

Chapter 3: Quantum Adversarial Learning in Emulation of Monte-Carlo Methods for Max-Cut Approximation: QAOA is not Optimal [1]

The idea of quantum computing was conceived in the early 1980s [69] as a way of simulating quantum systems using the laws of quantum mechanics. Quantum simulation algorithms were further developed in the 1990s [10, 70–73] and onward, generating a vast literature as one of the primary categories of quantum algorithms. Today, these algorithms are capable of simulating many scientifically relevant systems [74–87]. The goal of quantum simulation is to predict the quantum properties of the simulated system, such as ground state properties, binding energies, reaction rates, and quantum dynamics. Many algorithms for these purposes use quantum simulation as a subroutine to predict quantum states at a small number (relative to problem size) of different points in time. Simulation achieves this by approximately taking a path in Hilbert space that corresponds to the path taken by the simulated system. The correspondence must be efficiently computable for the simulation algorithm to be useful. However, this does not necessarily mean that the quantum computer takes the most efficient path to the final state in terms of computational resources such as time, energy, hardware size, circuit depth, circuit size, or cost of gate-set [88].

During the Noisy Intermediate-Scale Quantum (NISQ) [89] computing era, one of the

most important constraints on quantum circuits is the coherence time of the underlying devices. Therefore, when generating quantum states using NISQ computers, it is important to take a time-efficient path to get there. Quantum simulators follow a path analogous to the path of the original system, but it is often possible to take a faster path (e.g. geodesics in the control landscape [88]). The problem is that finding such faster algorithms requires greater knowledge not just of the control landscape but of the target state itself, stunting the development of such algorithms in the literature compared to their simulation counterparts. Using Monte-Carlo emulation, we compare parameterized quantum Max-Cut approximation algorithms and demonstrate that our novel parameterization scheme can emulate a traditional parameterization scheme, such as the Quantum Approximate Optimization Algorithm (QAOA) [16], in a shorter amount of time.

This chapter is organized as follows: in section 3.1 we explain the necessary background; in section 3.2 we extend the concepts of simulation and emulation to computer programs that are Monte-carlo methods; in section 3.3 we present the Quantum Annealing schedule that generalizes QAOA; in section 3.4 we present our numerical data comparing QAOA to its generalization; and lastly we conclude with remarks and future directions.

3.1 Background

Quantum Simulation: Quantum simulation was the original impetus for the proposal of quantum computers [69]. Simulating quantum systems on classical computers is suspected to be a hard problem, and Feynman's original proposal for quantum computers was a means of solving this problem by using quantum mechanics itself to simulate quantum

systems.

At its core, simulation is a procedure whereby the behavior of a difficult-to-control system is mirrored and captured by a simpler, controllable system. The advantage here is that the simulator could be smaller than the simulated system, cutting away unnecessary degrees of freedom, and that the simulator is a controlled environment. While universal quantum computers can efficiently simulate any quantum system [90], many special-purpose quantum simulators are designed for a small number, or even one task [91].

Simulation is one of the cornerstones of quantum algorithms, and it is expected that many near-term quantum applications will be simulation-based, especially in quantum chemistry [74]. The main limitation of quantum simulation is that it mirrors the evolution of the underlying system. The simulator is constrained to mimic what the original system does. This is great if the goal is to understand the entire process, but simulation is too strict of a requirement in our setting, where the only goal is the end state.

For instance, QAOA was originally proposed [16] based on inspiration from adiabatic quantum computing (AQC). Both algorithms have the same goal, to minimize (or maximize) some cost function, but they are allowed the freedom to go about this goal in different ways. Thus, QAOA and AQC while related, are not simulations of each other with QAOA exhibiting very different behavior in practice and even having greater computational power for some problem classes [92,93]. Our goal here in going back from QAOA to a variational polynomial quantum annealing algorithm will also not be one of simulation but rather more generally emulation.

Emulation: In computing, emulation is the replication of a computer system's behavior by another one [94]. Simulation is also a type of emulation namely, the emulation of the internal state of a computer system. The most important type of emulation is the emulation of the input-output behavior of a system. When the term "emulation" is used without any qualifiers, it usually refers to the emulation of input-output behavior. For the rest of this chapter, we will use this convention. Just like in simulation, resource use, such as time and memory, is a central issue with the design of emulator algorithms. In this chapter, our primary goal is to show that we can emulate QAOA with the same number of variational parameters but shorter worst-case requirements for time using a novel parameterization of the Quantum Annealing schedule.

Machine Learning Concepts: Our definition of Monte-Carlo emulation was inspired by two primary concepts in machine learning: Generative Models and Reinforcement Learning. In Generative Models, the goal is to approximately learn a probability distribution, often with the goal of efficiently sampling from the target distribution [95]. Specifically, we were inspired by Generative Adversarial Learning [96]. Generative Adversarial Learning is a two-part system. On one side, there is a generator which is the type of model described above. On the other side, there is a discriminator which gets trained to classify if a given sample is from the real target distribution or from the generative model. The generator is then trained not to be detected by the discriminator. By training the discriminator and the generator together, it is possible to achieve a very strong discriminator and a very strong generator. The second concept, reinforcement learning, is a type of machine learning where the objective is to maximize a reward function [97]. In particular, multi-agent

reinforcement learning helps define Monte-Carlo emulation. In multi-agent reinforcement learning, each agent tries to maximize a local reward such as gaining points in competitive sport in a physics engine environment [98]. We have used these ideas to define Monte-Carlo emulation. In our definition of Monte-Carlo emulation, we have a two-agent system: the emulator and the emulated. The emulator generates an indiscriminant distribution from emulated, an independent computer program. On the other hand, emulated tries to generate a distribution without using unnecessary resources. We have used this definition to compare QAOA to our novel parameterization[1].

Max-Cut problem and Ising Hamiltonian: On graphs, we can partition nodes into two sets with the set of edges between the two partitions known as a "Cut". The cardinality of Cut, or the number of edges cut, is referred to as the cutsize, and the task in Max-Cut is to find the partitioning of the system that maximizes the cutsize [99].

The Max-Cut can be encoded into a problem of binary variables, z_i, where each variable takes on a value of ± 1, with $+1$ corresponding to nodes in one partition and -1 corresponding to nodes in the other partition. This problem is then formulated as

$$C = \sum_{\langle i,j \rangle} z_i z_j, \tag{3.1}$$

where $\langle i, j \rangle$ denotes nodes i and j that share an edge in the graph. For a given partitioning of the nodes z_i into ± 1 bins, this cost function C will change, with each edge crossing

[1]If QAOA literature further develops where parameter optimization can not only be performed for expectation energy but also for fidelity with states that have energy bellow a given level, these ideas can be further applied to numerically analyze emulation. See Appendix A.2 for how these ideas can be further applied as the literature for variational quantum algorithm optimization improves.

between partitions contributing -1 to C and each edge within a partition contributing $+1$. Therefore, a partitioning that corresponds to the Max-Cut will give the lowest possible value of C. Max-Cut for general graphs is known to be an NP-hard problem [99].

This computational problem is equivalent to finding the ground state of the Ising model from theoretical physics [100]. In an Ising model, the variables z_i in Eq. (3.1) represent (magnetic or quantum) spins that can either be up or down [101]. C in this setting corresponds to the energy of the physical system with the lowest energy state corresponding to the partitioning of the graph that gives the maximum cut. To look at a quantum Ising model, we can take Eq. (3.1) and promote the variables z_i to Pauli-z operators acting on the ith qubit. In our following analysis of quantum annealing and QAOA, we will take our problem Hamiltonian to be just this:

$$\hat{C} = \sum_{\langle i,j \rangle} J_{ij} \sigma_z^{(i)} \sigma_z^{(j)}, \tag{3.2}$$

and our mixer Hamiltonian will be a transverse field on the qubits:

$$\hat{B} = -\sum_i \sigma_x^{(i)} \tag{3.3}$$

where, $\sigma_x^{(i)}$ and $\sigma_z^{(i)}$ refers to Pauli x operator applied to ith qubit and Pauli z operator applied to ith qubit respectively. In Max-Cut problems, the connectivity matrix, J_{ij} would just be one on edges in the graph and zero otherwise. The Ising model is more general and corresponds to what is called weighted Max-Cut where the J_{ij} matrix can take on any real values.

Monte-Carlo Methods for Optimization Problems: For some problems, it is very hard, if not impossible to find an exact solution. However, it might be much easier to find a good approximate solution for the same problem [102]. Monte-Carlo Methods are algorithmic techniques for generating a random variable around the exact solution. For many practical applications, good approximate solutions can be substitutes for the exact solution and can be obtained by sampling these random variables. There are many classical algorithms in the literature, that are used to study quantum mechanics and other areas of physics, which are Monte-Carlo methods. Monte-Carlo methods can also be used to approximate good solutions to optimization problems. Some optimization problems are very hard to solve exactly, such as Max-Cut which is NP-hard. According to the Exponential Time Hypothesis, the time requirements for Max-Cut make it infeasible to find the exact solution in large instances [103]. However, there are both classical and quantum algorithms to find good solutions that give a large cut of the graph [16, 104–106]. QAOA is one such method as it samples from good solutions of optimization problems. It has been substantially studied in its application to the Max-Cut problem [107]. Following the example of the literature on QAOA, we will mostly keep our discussion around application performance to Max-Cut. However, most of our arguments apply to other optimization problems as well.

For Max-Cut problems, Monte-Carlo methods generate a distribution on the support of partitions with corresponding cut sizes. This means that the Monte-Carlo method also has a distribution on the support of cut sizes. More generally, Monte-Carlo methods for optimization problems sample from states with some energy distribution. The Boltzmann distribution was the original inspiration for simulated annealing, which in turn inspired

Quantum Annealing [102]. Boltzmann distribution, which describes the relationship between temperature, energy, and the probability of thermodynamic states, has long been used as a benchmark for Monte-Carlo methods; however, it is not always the best distribution to characterize the energy distribution of a given Monte-Carlo method. In this chapter, we define a novel method for comparing the performance of Monte-Carlo methods which looks at distributions globally rather than using the temperature from the Boltzmann fit.

Remark. Monte-Carlo methods should not be confused with Monte-Carlo algorithms for decision problems which form a class of algorithms where the probability of getting the correct answer is lower bound by 2/3 for all instances of the problem. The set of problems solvable in polynomial time by classical and quantum computers by Monte-Carlo algorithms are called BPP and BQP respectively [10].

3.2 Emulation of Monte-carlo Methods

In this section, we extend the definition of computational emulation to computer programs that are Monte-Carlo methods. As described in the background section, the energy of the solution outputs sampled from these programs is a random variable. To replicate the behavior of the emulated, the emulator must find states with the same energy at the same rate. Two observations help to define this.

- Interchangibility: In optimization problems, the states that have the same energy are equivalent to each other by definition. Therefore emulator must match the probability distribution function (PDF) of the emulated unless the following property is satisfied:
- Substitution: We assume that states that are better solutions are substitutes for

states that are worse solutions however, the other way around is not true (see Figure 3.1). The probability of better solutions can be used as a substitute for the probability of worse solutions to match the PDF of the emulated distribution.

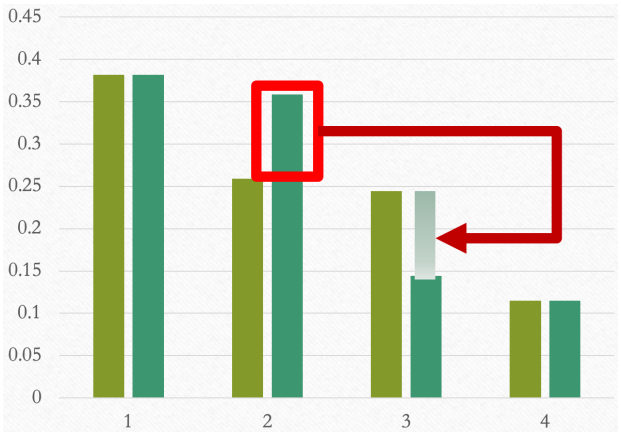


Figure 3.1: An example application of substitution property on a probability mass function.

For energy minimization problems, this means the emulator must majorize the Cumulative Distribution Function (CDF) of the probabilities of getting energy eigenstates from the emulated. Thus, for any statistic (such as median, expectation, or quantile) the performance of the emulator is bounded below by the performance of the emulated.

Monte-Carlo methods are invoked for optimization problems when the computational resources that are available is insufficient to compute the global optimum. Therefore, they are usually adaptable to different levels of resource availability [108]. When more resources are allocated, Monte-Carlo methods are usually able to produce better distributions. These computational resources are any of the assets that needs to be provided to a computer program to run to completion such as time, memory, energy, communication network,

queries etc. By using any combination of these, we can define a cost functional. Let

$$c_b^g[F] \tag{3.4}$$

be the cost of Monte-Carlo method b to generate a distribution that majorizes F for problem instance g. The minimum cost distribution generated by b does not have to match F only majorize it. Let

$$G_b^g[F] := c_b^g[G_b^g[F]] = c_b^g[F]$$

be such minimum-cost distributions. As a convention, let a distribution that is possible to generate but takes infinite resources, cost $\aleph_0$. Let distributions that cannot be generated by method b for problem instance g cost $\aleph_1$. Method b does not have $G_b^g[F]$ for these not generatable distributions. We shall denote these by a function that is not a CDF

$$c_b^g[F] = \aleph_1 \iff \forall x \in \mathbb{R}, G_b^g[F](x) = 2.$$

Furthermore, these definitions allow us to compare Monte-Carlo methods with each other. In many cases, two Monte-Carlo methods could be better at different aspects in the distributions that they generate. For example, one could have a lower expectation while the other one has a lower median in its distribution. However, given enough resources, one Monte-Carlo method might be able to emulate the other one. To quantify the relative resource requirements of method a emulating method b, we define "emulation factor":

$$J_{a,b}^g[F] := \frac{c_a^g[G_b^g[F]]}{c_b^g[F]} = \frac{c_a^g[G_b^g[F]]}{c_b^g[G_b^g[F]]}.$$

For parameterized Monte-carlo methods with p parameters, we define "parameterized emulation factor":

$$\hat{J}^g_{a,b,p}[F] := \frac{c^g_{(a,p)}[G^g_{(b,p)}[F]]}{c^g_{(b,p)}[G^g_{(b,p)}[F]]}.$$

In Section 3.4, we will use this definition to compare bang-bang parameterization and our novel polynomial parameterization of Quantum Annelaing. We will explore the extrema of this functional with respect to F, p and g. In the next section, we present our novel polynomial parameterization which is a generalization of QAOA that demonstrates that QAOA is not optimal by using parameterized emulation factor.

3.2.1 Post-Selection on Multiple Runs

One way we can improve the energy CDF of our samples is to replace each sample with the result of a binning process across $m > 1$ samples picking the lowest energy in the bin to be our new sample, a process we will refer to as post-selection. This way, the original CDF of the energy distribution of a single sample, $F(x)$, is transformed to

$$1 - (1 - F(x))^m.$$

We do not include this post-selection method to improve the CDF in the scope of this study. We choose to exclude this because the number of queries is a computational resource such as annealing time, number of qubits, or the number of parameters. So, one of our objectives is to avoid post-selecting on multiple samples, as this would increase the number of queries to the Monte-Carlo method. Furthermore, the number of samples needed for post-selection

may be unbounded. If an x value exists where the emulated CDF is 1 and the emulator CDF is not 1, post-selection requires an infinite amount of samples. This can easily be the case for large problem sizes where there is a lot of room for differing behavior between different algorithmic approaches. Therefore, we exclude[2] post-selection from this study.

3.3 Polynomial Parameterization

We parameterize our quantum annealing schedule by clipping polynomials. The Clip function refers to the function which projects its input to a window [109]:

$$K_{a,b}(x) = min(a, max(x, b)). \tag{3.5}$$

By scaling our Hamiltonians to the full parameter range of the experimental setup, we can ensure that the control function $u(t) \in [0, 1]$. To reflect this with the Clip function, we use $a = 0$ and $b = 1$. We use a clipped linear combination of the first $2p$ monomials:

$$u(t) = K_{0,1}\left(\sum_{j=0}^{2p-1} c_j x^j\right). \tag{3.6}$$

Our use of $2p$ is so that our number of control parameters matches up with QAOA.

We optimize for the linear combination coefficients c_js thus defining a variational schedule. The c_js are allowed to take any real values and, thus, do not have direct optimization bounds.

This parameterization is a generalization of QAOA because it can approximate any

[2] In practice it might be possible overcome the problem to the level of machine-epsilon or other sources of noise so it could be important to study it, see section 3.6.

QAOA schedule to arbitrary accuracy by using the same number of parameters. For a QAOA schedule with pulses changing at times $t_1, \ldots, t_{2p-1}$ the corresponding c_js are Lagrange interpolation of points

$$(0, \lim_{y \to \infty} -y), \forall j \in \{1, 2, \ldots, 2p-1\}, (t_j, 0). \tag{3.7}$$

QAOA with p layers is numerically observed to have a QAOA curve with polynomial of degree $p-1$ as defined in Refs. [30, 33]. Each layer of QAOA is defined by two variables so, QAOA of depth p has $2p$ variables. This means QAOA uses $2p$ variables to traverse an underlying curve which only requires p parameters to describe with polynomial parameterization. Therefore, compared to QAOA's underlying curve, the polynomial has twice the degree for the same number of variables.

3.4 Comparison of Time Resource Requirements to QAOA

In this section, we explore the ability of bang-bang and polynomial annealing schedules to emulate one another for the same number of parameters. We are interested in annealing time requirements for a given distribution which we will define as our cost function (section 3.4). This means that we will ignore the cost of parameter optimization in this section and assume that we have an oracle that gives optimal parameters to generate $G^g_{(b,p)}$. We have investigated the extrema of parameterized emulation factor of bang-bang and polynomial parameterizations on each other and quantified their relative capabilities. For large problem size $|g| = n$, the summary of our results shown is below:

$\hat{J}^g_{a,b,p}[F]$	a =Bang-Bang, b = Polynomial	a = Polynomial, b = Bang-Bang
$\max_{\lvert g\rvert=n,F,p}$	$\aleph_1$ (section 3.4.1)	1 (section 3.4.2)
$\min_{\lvert g\rvert=n,F,p}$	1 (section 3.4.3)	$\log(n)^{-\Omega(1)}$ (section 3.4.4)

Table 3.1: The upper and lower bounds of the emulation factor comparing Bang-Bang schedules and polynomial schedules.

In Section 3.4.1, we show how the linear adiabatic ramp [26] can be encoded as our polynomial schedule. In Section 3.4.2, we prove how Bang-Bang schedules are a sub-type of our polynomial schedule. In Section 3.4.3, we cite previously-known settings where Bang-Bang schedules are proven to be optimal. Lastly, in Section 3.4.4, we share the results of our classical simulations that we used to obtain the conjecture for scaling, $\log(n)^{-\Omega(1)}$.

3.4.1 Emulation Limits of Bang-Bang Schedules

As formulated, both QAOA and our poly schedules are limited only by the number of parameters involved, not the length of time the procedures take. Under this formulation, it is clear that the polynomial schedules are capable of tasks that QAOA is not.

For instance, consider a $p = 1$ QAOA schedule with $2p = 2$ parameters. QAOA is known to be strictly limited in its performance by the number of its parameters [16, 32], meaning that for a fixed number of parameters, there is a hard limit on how close QAOA can get to the target metric. On the other hand, even with $2p = 2$ parameters, the polynomial schedule can create a linear adiabatic ramp and use the quantum adiabatic theorem to asymptotically approach the target state in the long time limit [26]. Therefore, with time unbounded, the polynomial schedules can always perform the adiabatic path as opposed

to the fixed depth behavior of QAOA.

This means that QAOA can never emulate a polynomial schedule in an optimized, time-unbounded setting. This is a feature of unbounded time, and for much of our numeric study, we will alleviate this issue by focusing on the minimum time for emulation, usually determined by the natural time that QAOA's parameter optimization settles to.

3.4.2 Bang-Bang is a sub-type of the Polynomial Schedule

As seen in section 3.3, we can use Lagrange interpolation to approximate the Bang-Bang schedule to arbitrary accuracy. Therefore, every Bang-Bang schedule can be asymptotically approached by a polynomial schedule. This means that the time required for a polynomial schedule to generate a certain distribution is upper bounded by the time required by the shortest time Bang-Bang schedule with the same number of parameters regardless of the problem instance.

3.4.3 Bang-Bang Schedule can be the shortest schedule

There are a few settings where QAOA-like bang-bang schedules are optimal. The most obvious example would be a single qubit setting or settings where collections of qubits act like independent qubits or effectively single spins [110, 111]. For a single qubit, the optimal path can be shown to just be determined by standard Euler rotation angles, but this only works because of the homomorphism between $SU(2)$ and $SO(3)$.

A more interesting setting can be derived from optimal control theory, which says that the optimal annealing schedule must start and end with bangs [31]. The duration of these

bangs increases as the total annealing time decreases. Therefore, there exists a runtime limit below which these two bangs dominate, resulting in a schedule that is equivalent to a depth $p = 1$ QAOA schedule. Despite being a very short schedule, this limit does exist and can be observed numerically. Furthermore, while the optimal annealing protocol typically does not take on an exclusively bang-bang form [31], there is nothing theoretically preventing this. For example, bang-bang is the optimal schedule in the connected graph of 2 nodes. Similarly, a larger graph made up of $n/2$ disconnected components of 2 nodes also has bang-bang as the optimal schedule, and it is possible that other less trivial examples exist as well.

The last setting where Bang-Bang schedules take the optimal amount of time is $p \to \infty$. This is because, with unbounded p, Trotter-Suzuki decomposition has no time overhead for $u(t) \in [0, 1]$. This can be seen by the standard Trotter formula which for a single time step gives

$$e^{-i\Delta t(u(t)\hat{B}+(1-u(t))\hat{C})}$$

$$= e^{-i\Delta t u(t)\hat{B}} e^{-i\Delta t(1-u(t))\hat{C}} + \mathcal{O}(\Delta t^2). \tag{3.8}$$

The operators on both sides of this equation will take time Δt, up to the corrections which will vanish in the limit as $\Delta t \to 0$, which can be achieved when $p \to \infty$. Note that just because QAOA can become optimal via this small-time step, Trotterization does not at all mean that this is how QAOA behaves in practice.

3.4.4 Computational results for time requirements for Polynomial Parameterization to Majorize QAOA

Unlike the previous sections, the theory in this area is harder to develop. Hence, we rely on numerics to develop insights into how polynomial parameterization can be used to emulate a Bang-bang parameterization. In this section, we present how low parameterized emulation factor can get for Polynomial schedules when emulating Bang-Bang schedules. In other words, we find the time overhead in using Bang-Bang parameterization compared to Polynomial parameterization even when the distribution of Bang-Bang parameterization is well-suited for the application.

As the number of qubits grows, the number of unique optimization problems increases very fast. For example, there are polyexponentially many unlabeled graphs (see OEIS A000088) in the number of nodes which means that there are that many unique unweighted Max-Cut problems. This increases the variety of problems thus the probability of finding a problem that is well suited for Polynomial parameterization but not well suited for Bang-Bang parameterization. The separation between these two parameterizations might be similar to the separation between two gate sets that satisfy Solovay–Kitaev universality. Therefore, we would like to further conjecture that the emulation factor decreases as a poly-logarithmic function of the number of qubits

$$\frac{1}{\log(n)^{\Omega(1)}}.$$

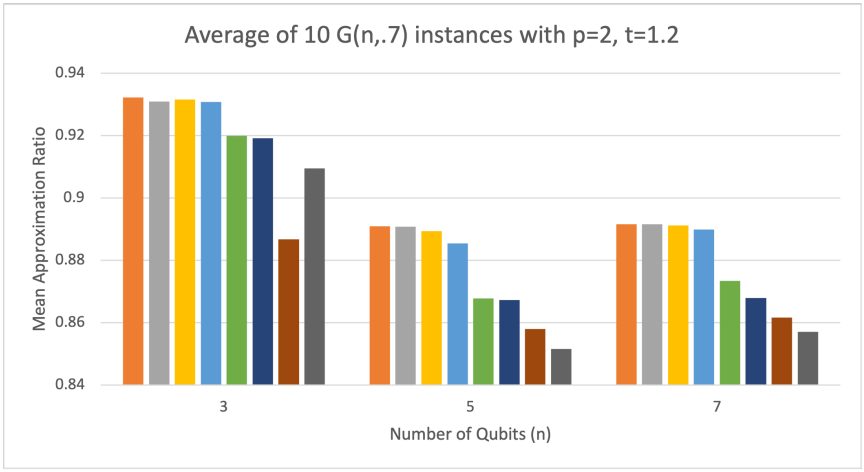
We have computational results that support this conjecture (Table 3.2). Bang-bang

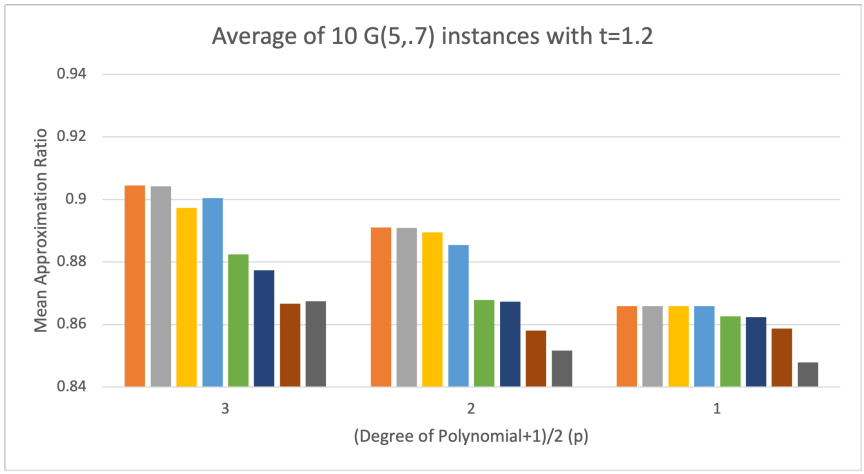
parameter optimization is known to be NP-hard for the ideal case [112]. However, QAOA parameter optimization literature has seen improvements in recent years to approximate the ideal schedule very well. Without a priori knowledge, bootstrapping with Nelder-mead is widely accepted to generate schedules that give very close expectations to what is achievable with that number of parameters. Since this method produces a locally minimum expectation, we can assume that it also generates a minimal time schedule for its distribution or a close approximation of the minimal time. So we majorized these QAOA schedules with poly and observed the following time separations:

p \ n	1	2	3	4
1	1	1	.98	.963
2	1	1	1	.947
3	1	1	1	1

Table 3.2: For problem size n, we were able to find problem instances and parameter values where polynomial schedule with $2p$ parameters takes the given fraction of time compared to QAOA to emulate QAOA with $2p$ parameters.

See Appendix A.1 for full data. We would like to remind the reader that these separations are likely lower bounds on parameterized emulation factor of respective instances as the literature for optimizing parameters for QAOA is well-developed but the literature for optimizing polynomial parameters is not explored. We discuss how these results can be improved in Section 3.6. Next, we present the performance of different gradient-free methods for polynomial parameterization.





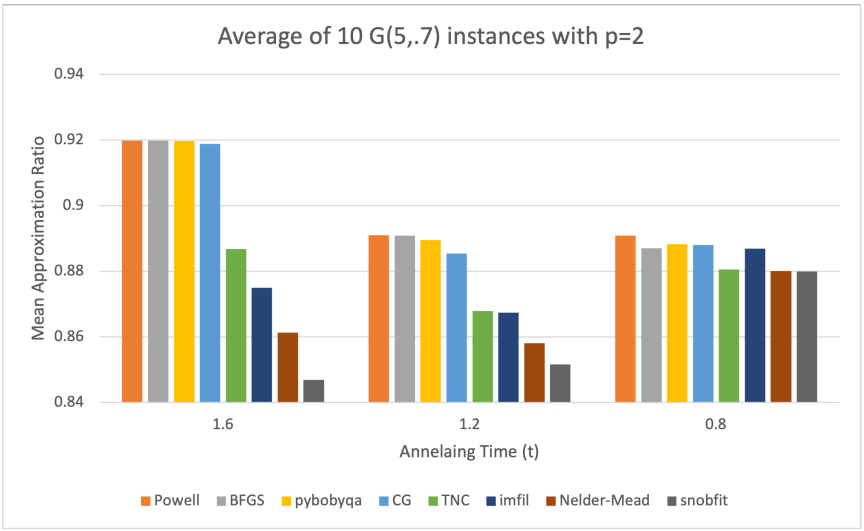


Figure 3.2: The summary of our data which shows the comparison between the performance of gradient-free optimizers for the polynomial schedule in terms of mean approximation ratio over 10 runs.

3.5 Parameter Optimization Method

We compared gradient-free optimizers from skquant [113] and scipy [109]. Averaging over 10 independent instances of Erdos-Reyni graphs[3] for each combination of parameters, we observed the approximation ratios in Figure 3.2 (see appendix A.1 for full parameter information). We can observe that Powell, BFGS, pybobyqa, and CG consistently performed better than TNC, imfill, Nelder-Mead, and snobfit ([Sic] following the capitalization convention of the documentation). Among those Powell was the best performer and our results suggest that it is a robust method for optimizing the Polynomial schedule regardless of the number of qubits, polynomial degree, or annealing time.

The folklore[4] suggests Nelder-mead is a very good optimizer for QAOA with bootstrapping. Unlike polynomial parameterization, annealing time in Bang-Bang schedules time is not usually treated as a computational resource. Rather, it is determined as a consequence of optimizing the parameters. Therefore, while optimizing for parameters, the total time might be different if two different methods converge to different local minimums. We have first optimized QAOA optimized using Nelder-Mead and Powell. In order to establish a fair comparison, we have taken the total QAOA time generated by both these methods and optimized Polynomial using Nelder-Mead and Powell for each of these times. Based on the averages we collected, in Figure 3.3, we can see that Powell applied to Polynomial performs comparably to or better than QAOA regardless of the number of qubits, number of parameters, or annealing time resulting from QAOA optimization method.

In order to give the reader some intuition for the properties of QAOA versus the

[3] Erdos-Reyni AKA, $G(n, p)$ is a random graph with n nodes with probability p for each edge [114].
[4] AKA mathematical folklore

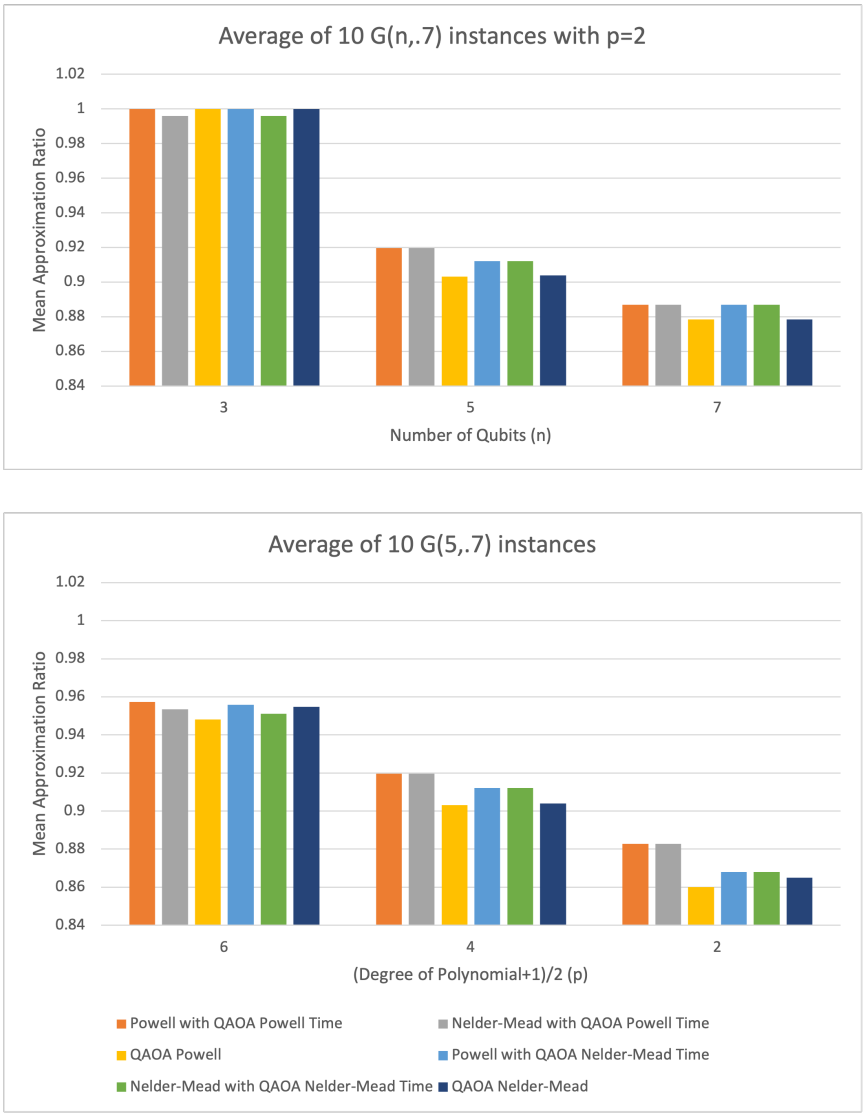


Figure 3.3: Comparison of gradient-free optimizers for polynomial and QAOA

Polynomial schedules, Figure 3.4 shows how a Polynomial schedule optimized with Powell compares to QAOA optimized with Nelder-Mead. We would like to note the visible difference in CDFs between QAOA and Polynomial schedules and the negligible difference between Polynomial and optimal schedules. This shows that the polynomial schedule has discretized the domain of all annealing schedules effectively, even with a low number of parameters. For full data, see Appendix A.1.

3.6 Conclusion

To sum up, the polynomial schedule can do everything Bang-bang can do, sometimes even in a shorter time, and never requires a longer time. Bang-bang cannot generate some distributions that the Polynomial parameterization generates. Polynomial parameterization can be optimized via Powell and often is comparable or better in terms of approximation ratio.

Overall, the polynomial is better than the Bang-bang schedules in terms of distributions it can generate and ease of optimization. The shortcoming of the polynomial is that the underlying computational model assumption of the Bang-bang schedule is a subset of the polynomial schedule. Namely, the type of control required on the annealing schedule to be able to experimentally implement the polynomial schedule is at least as hard as the type of control required for QAOA. Even for two different control models which share important properties, such as Solovay–Kitaev Universal gate sets, the experimental implementation might be vastly different and challenging in distinct ways. Thus, we cannot assume that experimental challenges for the polynomial schedule would not cause significant overhead

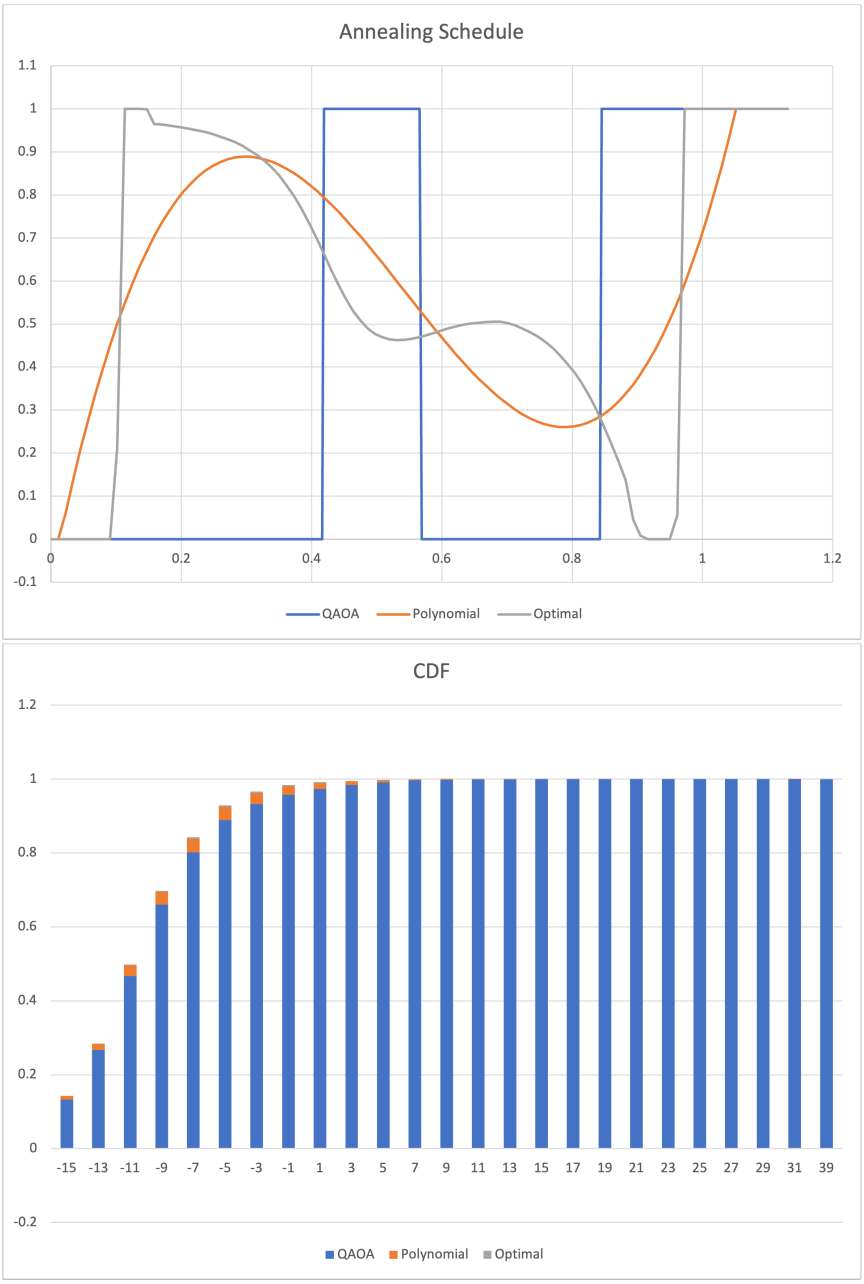


Figure 3.4: The top graph shows the annealing schedule, $u(t)$ (as in Equation 1.1), for optimized QAOA with 4 parameters, the optimized polynomial schedule with 4 parameters, and the optimal schedule.
The bottom graph shows the CDF that each schedule generates in terms of approximation ratio. Note the visible difference between the CDFs of QAOA and the polynomial schedule.

compared to a gate-based system.

This brings us to many important implications and directions for future work. Firstly, this study gives motivation for experimental groups to explore ways of implementing arbitrary schedules on annealers. Secondly, an important question about the initialization of parameters remains for polynomial. Thirdly, Monte-Carlo emulation can and should be used in other areas of machine-learning literature to measure the relative resource requirements of different models and different types of cost functions (see Section 3.2.1). Expanding on that, Monte-Carlo emulation defines a comparison method for the number of parameters in generative machine learning models. Lastly, the separations that we found in the section 3.4.4 were just examples. It would be valuable to see how large these separations can be. (Appendix A.2 further discusses this.).

Chapter 4: Job Scheduling Applications of Quantum Annealing with Higher-Order Magnetic Coupling Hamiltonian [3]

4.1 Introduction

Quantum computers offer a new and potentially more powerful method of tackling a variety of problems. As discussed in the previous chapter, one of the possible applications is in combinatorial optimization. While algorithms like Quantum Annealing (QA) [17], Quantum Adiabatic Optimization (QAO) [26], and the Quantum Approximate Optimization Algorithm (QAOA) [16] can natively handle unconstrained optimization problems, they falter when it comes to handling constraints. These algorithms natively handle optimization problems by encoding them into Hamiltonians on qubits. For classical problems, the problem Hamiltonian is diagonal, essentially just functions of binary variables. This means that it can be hard to represent the relations between the binary variables compared to representing their energy function. These additional relations are often in the form of constraints, which are boolean functions of the variables that must be true.

In classical computing, these constraints can be handled through the structure of the algorithm used to solve the problems such as branch-and-bound methods [115]. Methods exist in quantum computers for restricting the search space of an algorithm to valid,

constraint-satisfying, solutions. For instance, it is possible to design the mixer, $\hat{B}$, in QAOA such that the evolution is restricted by symmetries in the system to only move in the valid subspace of the problem [116]. The problem with this is that it can be difficult to engineer an appropriate mixer for all but the simplest problem instances. This can potentially be solved by using iterative techniques similar to existing classical methods [117, 118], but it is currently unclear how much quantum advantage exists in such methods.

The more standard approach for applying constraints in quantum algorithms and classical spin systems is to impose the constraints via a penalty function [119]. In this form, violations of the constraint are penalized with additional energy (or negative energy in maximization problems), and bit-strings that satisfy the constraints receive no alteration to their energy. These penalty functions are then added onto the basic cost or energy function to create a new Hamiltonian where the valid solutions are in the ground subspace of the combined penalty terms. Then the only art is to design sufficiently weight the penalty terms so that it is advantageous to stay in the valid subspace while the algorithm explores the landscape of the cost portion of the energy function.

This penalty method works well for constraints that can be described in terms of equalities, such as $f(\vec{x}) = b$ where $\vec{x}$ is a vector of binary variables, f is some function of those variables, and b is the constraint. In this case, a simple penalty function would just be quadratic in the form $(f(\vec{x}) - b)^2$ such that all invalid bit-strings have positive energy. This becomes much harder for inequality constraints, such as $f(\vec{x}) \leq b$. One standard practice here is to employ additional slack variables [119] that can take on values such that the inequality essentially becomes a host of equalities involving the slack variables. In [119], the authors suggested using so-called slack variables $\vec{\Delta}$. These are placed in a

penalty function to impose the linear constraint $C\vec{x} \leq \vec{b}$ with q lines on binary vector $\vec{x}$:

$$\acute{L}(\vec{x}, \vec{\Delta}) = \lambda \sum_{i=0}^{q-1} ((C\vec{x} - \vec{\Delta})_i)^2$$

where $\Delta_i \in \mathbb{Z}_{b_i+1}$ and λ is a large coefficient. This mapping works because $\vec{\Delta}$ can take the same value as the cost $C\vec{x}$ if and only if $\vec{x}$ is a valid state. When $C\vec{x} = \vec{\Delta}$, $\acute{L}(\vec{x}, \vec{\Delta}) = 0$ and is positive otherwise. With this method, integer problems are only mapped to Hamiltonians with quadratic spin interactions i.e. Ising Hamiltonians. With recent experimental developments, NISQ devices no longer need to be restricted to quadratic interactions. Now it is possible to implement higher-order interactions [120]. This allows for constraint-mapping methods other than the slack variable method. A recent work by Djidjev proposes using the roots of higher-order polynomials, which can be implemented with the higher-order interaction Hamiltonians [121]. In this mapping, constraint $C\vec{x} \leq \vec{b}$ with q lines on binary vector $\vec{x}$ is mapped to the penalty function

$$\grave{L}(\vec{x}, \vec{\Delta}) = \lambda \sum_{i=0}^{q-1} \prod_{j=0}^{b_i} ((C\vec{x})_i - j)$$

where λ is a large coefficient. This method works because, for each valid $(C\vec{x})_i$, there is a j such that $(C\vec{x})_i = j$ making $(C\vec{x})_i - j = 0$ thus $\prod_{j=0}^{b_i}((C\vec{x})_i - j) = 0$. Both these methods become very expensive for real-world applications. Slack variable method uses $\sum_{i=0}^{q-1} \lceil \log_2(b_i + 1) \rceil$ bits for $\vec{\Delta}$. This can result in a large number of additional slack variables, which can be a significant hurdle for NISQ implementations. The higher-order polynomial method requires a spin interaction of degree $\max_i b_i$, which can easily exceed

the capability of the NISQ systems. To illustrate the cost in an application setting, consider the model of a High-Performance Computer (HPC) job queue in the following section.

4.2 Modeling of an HPC Job Queue

We will look at T timesteps to the future, where T is a modeling parameter. We have N cores in the system and discretize the total memory into M units. We assume that cores are interchangeable and memory units are interchangeable. We also assume that these components are well connected. At the current timestep, there are k elements in the queue. We will describe an optimization task that will be repeated in each timestep since elements might be removed from the queue and added to the queue at each timestep. The timesteps are indexed with $j \in \mathbb{Z}_T$ with 0 denoting the current timestep. There might be executions currently running in the system that started in some previous timestep. Based on starting time and maximum requested execution time of the currently active tasks, let N_j be the number of cores available at timestep j and let M_j be the amount of memory available at timestep j. There are k tasks in the queue indexed by $i \in \mathbb{Z}_k$ with requested memory m_i, requested number of cores n_i, and requested maximum time T_i. Each element in the queue has a reward value, R_i, computed from a variety of factors such as wait time, size, and the submitter. The goal is to strike a balance between collected total rewards and how quickly these rewards are collected since time is a non-fungible resource, and the queue might receive more elements, so we define a penalty term for jobs that are scheduled

late, f, a decreasing function, also a modeling parameter. The optimization task is

$$\max_{X} \sum_{i=0}^{k-1} \sum_{j=0}^{T-T_i} f(j+T_i) R_i X_{i,j} \tag{4.1}$$

s.t.

$$X \in \mathbb{Z}_2^{k \times T}, \tag{4.2}$$

$$\forall i \sum_{j=0}^{T-1} X_{i,j} \leq 1, \tag{4.3}$$

$$\forall j \in \mathbb{Z}_T, \sum_{i=0}^{k-1} \sum_{t=j-T_i}^{j} m_i X_{i,t} \leq M_j, \tag{4.4}$$

$$\forall j \in \mathbb{Z}_T, \sum_{i=0}^{k-1} \sum_{t=j-T_i}^{j} n_i X_{i,t} \leq N_j. \tag{4.5}$$

In this optimization task, $X_{i,j} = 1$ represents scheduling task i to begin at timestep j and (4.4),(4.5) are the constraint to stay within the available hardware resources at a given timestep. As we transition from the current timestep to the next one, each task, i, for which $X_{i,0} = 1$, begins its execution and is removed from the queue. All others remain in the queue. We repeat the optimization procedure with new values when new elements are added to the queue. Note that, T and f are meta-parameters of the model.

Scheduling jobs with this simple model is too expensive with the above-mentioned mappings. Constraint (4.3) can be imposed without needing the above-mentioned mappings and without needing additional resources. To impose this constraint, we use the penalty function:

$$\sum_{i=0}^{k-1} \sum_{t \neq j} X_{i,t} X_{i,j}.$$

However, constraints (4.5) and (4.4) requires up to $(\lceil \log_2(N) \rceil + \lceil \log_2(M) \rceil)T$ slack variables with the mapping in [119]. This overhead can easily be prohibitive, especially for NISQ devices. On the other side, using the higher-order polynomial method requires spin interactions of degree $\max(N, M)$, which only allows the simplest instances of this problem to be mapped to NISQ devices. The next section discusses how we can overcome these mapping challenges.

4.3 Encoding Constrained Integer Programs with Binary Polynomials

One way to circumvent the problems caused by the slack variable approach and higher-order polynomials approach is to do a combination of them. If the system allows up to pth order spin interactions, we can map $C\vec{x} \leq \vec{b}$ to:

$$\check{L}(\vec{x}, \vec{\Delta}) = \lambda \sum_{i=0}^{q-1} \prod_{j=0}^{p-1} ((C\vec{x})_i - p\vec{\Delta}_i - j)$$

where λ is a large coefficient. In this setting, $\sum_{i=0}^{q-1} \lceil \log_2((b_i + 1)/p) \rceil$ bits are needed for $\vec{\Delta}$. The slack variable method, higher-order polynomial method, and hybrid method are all "perfect" mappings in the sense that they preserve the order of the energy values of the solution before and after mapping. In other words, when we map

$$\min_{\vec{x}} A\vec{x} \text{ s.t. } C\vec{x} \leq \vec{b}$$

to

$$\min_{\vec{x}, \vec{\Delta}} A\vec{x} + \check{L}(\vec{x}, \vec{\Delta}),$$

two valid states $\vec{y}, \vec{z}$ have the following relation

$$\min_{\vec{\Delta}} A\vec{y} + \check{L}(\vec{y}, \vec{\Delta}) > \min_{\vec{\Delta}} A\vec{z} + \check{L}(\vec{z}, \vec{\Delta})$$

if and only if

$$A\vec{y} > A\vec{z}$$

because, for any valid state $\vec{\chi}$,

$$\min_{\vec{\Delta}} A\vec{\chi} + \check{L}(\vec{\chi}, \vec{\Delta}) = A\vec{\chi} + \check{L}(\vec{\chi}, C\vec{\chi}) = A\vec{\chi}.$$

If we remove the requirement for "perfect" mappings, we can further generalize the hybrid approach and decrease the number of slack variables needed even further with a novel approach. The level of accuracy provided by "perfect" mappings is not required for all applications. In fact, mapping errors might be negligible when other sources of error are considered. Here, we demonstrate how relaxing the requirement for perfect mapping decreases the number of slack variables.

Since the overall goal is to perform approximate optimization, it might be acceptable not to differentiate between valid solutions with similar objective values. However, it is still important to be able to create a mapping such that all invalid states take higher energy values in the Hamiltonian than all valid states since the best solutions are usually those that are close to the edge between valid and invalid, which means that it is absolutely essential that we do differentiate between them. Let p be the maximum coupling, $R_{\max}$ be the maximum reward per element in $\vec{x}$, and ϵ be the maximum difference in the reward

that we are willing to not differentiate for valid states. For the rest of this chapter, we will only consider even values for p.

As p increases, so do the different ways we can couple the qubits. Instead of $\Delta_i \in \mathbb{Z}_{b_i+1}$ consider $\Delta_i \in \mathbb{Z}_{2^{s_i}}$ where s_i is the number of bits needed to represent Δ_i. Let $g_i = \lceil (b_i+1)/2^{s_i} \rceil$. We consider the penalty function in the form

$$\sum_{i=0}^{T-1} w_i L_i((C\vec{x})_i - g_i\vec{\Delta}_i)$$

where $w_i > 0$ and L_is are polynomials of degree p which that are normalized to $L_i(y) \in [0,1], \forall y \in \mathbb{Z}_{g_i}$. The above-mentioned requirements are satisfied when

$$\sum_{i=0}^{T-1} (L_i(g_i))^{-1} < \frac{\epsilon}{1+R_{\max}}. \tag{4.6}$$

See Appendix B.1 for the derivation of (4.6). In order to keep the number of bits needed to represent slack variables, $\sum_i s_i$, low, it is important to be able to handle larger g_is. So, we would need to maximize

$$\min_{x \in \mathbb{Z} \setminus \mathbb{Z}_{g_i}} L(x) \text{ s.t. } \forall y \in, \mathbb{Z}_{g_i} L_i(y) \in [0,1]. \tag{4.7}$$

We have numerically observed that maximizing (4.7) converges to scaling and translation of Chebyshev Polynomials of the first kind fairly quickly as g_i grows. Despite that, the other polynomials that are better maximizers of (4.7) when g_i is small are especially important since the cases where g_i is small will be seen more often than cases where g_i is large. We can calculate the optimal number of slack bits with a greedy algorithm by iteratively

incrementing s_j where $j = \arg\max_i g_i$ as this decreases $\sum_{i=0}^{T-1}(L_i(g_i+1))^{-1}$ the most, once we know what L_is should be. Although p and $\vec{g}$ are problem dependent, for a given p and g_i, the optimal polynomial is independent of the problem. So, we have pre-computed the coefficients of these polynomials up to $p = 8$, which is presented in the next section.

4.4 Non-Chebyshev Polynomials

When g_i is small, the polynomial, L, of degree p that maximizes (4.7) is not always the Chebyshev Polynomial. For example, in the case where $g_i = 7$ and $p = 6$, scaling and translation of Chebyshev Polynomial to fit our $[0, 1]$ normalization gives

$$L_c(x) = 1 - 6x + \frac{35}{3}x^2 - \frac{226}{27}x^3 + \frac{8}{3}x^4 - \frac{32}{81}x^5 + \frac{16}{729}x^6.$$

However, the optimal polynomial is

$$L^*(x) = 1 - \frac{208}{15}x + \frac{1148}{45}x^2 - \frac{52}{3}x^3 + \frac{49}{9}x^4 - \frac{4}{5}x^5 + \frac{2}{45}x^6.$$

This is illustrated in the following table:

x	-1	0	1	2	3	4	5	6	7
$L_c(x)$	30.05	1.	0.66	0.73	0.	0.73	0.66	1.	30.05
$L^*(x)$	64	1	0	1	0	1	0	1	64

Table 4.1: Values $L_c(x)$ and $L^*(x)$ outputs for the elements of $\mathbb{Z}_7$ compared to values at -1 and 1.

In this case, the optimal polynomial is twice as better as Chebyshev Polynomial in terms

p	g	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
6	7	1.00000000	-13.86666667	25.51111111	-17.33333333	5.44444444	-0.80000000	0.04444444		
6	8	0.35435023	-7.16582416	12.76746540	-7.85357245	2.17199845	-0.27701481	0.01325804		
6	9	0.13908643	-3.66240041	6.25740236	-3.50887547	0.87255459	-0.09932660	0.00422280		
6	10	0.97508236	-4.89184402	6.33476329	-2.99501748	0.64176345	-0.06348058	0.00235616		
6	11	0.99973670	-4.10583917	4.85762486	-2.07692112	0.40098101	-0.03568586	0.00119099		
6	12	0.99999890	-3.56190182	3.80015650	-1.47714215	0.25920624	-0.02095237	0.00063492		
6	13	0.00306566	-0.36981389	1.24828298	-0.59578318	0.11014889	-0.00886326	0.00026009		
6	14	0.99999376	-2.82515191	2.54612639	-0.83692651	0.12423728	-0.00849672	0.00021786		
6	Chebyshev	1.00000000	-2.57142857	2.14285714	-0.65306122	0.08996252	-0.00571191	0.00013600		
8	9	1.00000000	0.00000000	-0.40634921	0.00000000	0.04444444	0.00000000	-0.00138889	0.00000000	0.00001240
8	10	0.99960213	-14.77787429	34.45842065	-29.88252335	12.84188765	-3.02699637	0.39827183	-0.02742981	0.00077009
8	11	0.06933130	-7.04935331	17.50278703	-14.49428649	5.78281520	-1.25018842	0.14986310	-0.00936430	0.00023786
8	12	0.37837318	-5.56095770	12.03259726	-8.97007056	3.22434639	-0.62769178	0.06775145	-0.00381346	0.00008731
8	13	0.26010639	-5.91679713	10.80251465	-7.03191288	2.25168879	-0.39504622	0.03869840	-0.00198524	0.00004153
8	14	0.54680940	-5.36853757	8.70931147	-5.19346598	1.53235910	-0.24779491	0.02235324	-0.00105506	0.00002030
8	15	0.90441610	-4.99165578	6.86535846	-3.59250369	0.94413284	-0.13801834	0.01139444	-0.00049659	0.00000887
8	16	0.99998185	-5.46620890	7.52213702	-3.94447175	1.02215928	-0.14448825	0.01134797	-0.00046498	0.00000775
8	17	0.99976712	-4.78750996	6.17827460	-3.03740780	0.73768506	-0.09773184	0.00719487	-0.00027636	0.00000432
8	18	0.83034732	-3.92631069	5.00087944	-2.35349216	0.54179660	-0.06780763	0.00470946	-0.00017056	0.00000251
8	19	1.00000105	-3.85410462	4.47817438	-1.97657463	0.42917723	-0.05071079	0.00332483	-0.00011364	0.00000158
8	20	1.00002024	-3.51566811	3.87136370	-1.62373582	0.33493062	-0.03756167	0.00233559	-0.00007566	0.00000100
8	21	1.00000040	-3.23747284	3.39719923	-1.35821002	0.26676620	-0.02846288	0.00168284	-0.00005182	0.00000065
8	22	1.00003459	-3.12540708	3.13064123	-1.19388136	0.22346516	-0.02270927	0.00127839	-0.00003747	0.00000045
8	23	1.00001700	-2.97103248	2.83276767	-1.02750382	0.18311713	-0.01773457	0.00095205	-0.00002662	0.00000030
8	24	0.99279505	-2.82153402	2.59726283	-0.90546182	0.15478738	-0.01436298	0.00073823	-0.00001976	0.00000021
8	25	1.00000000	-2.69517662	2.36739555	-0.79074963	0.12957525	-0.01152423	0.00056764	-0.00001456	0.00000015
8	Chebyshev	1.00000000	-2.56000000	2.15040000	-0.68812800	0.10813440	-0.00922747	0.00043621	-0.00001074	0.00000011

Figure 4.1: The coefficients for cases where we found better solutions than Chebyshev Polynomials. c_is are the coefficient of monomial x^is.

of (4.7). This is not the only case where Chebyshev Polynomial is not the optimal choice.

We found many cases where Chebyshev Polynomial is not the optimal solution to (4.7). Despite that, Chebyshev Polynomials are the optimal solution even for small g_i values for some p. Since all parabolas are equivalent under scaling and translation, for $p = 2$, Chebyshev is the optimal case. In order to pre-compute the rest, we have optimized coefficients using scipy Nelder-Mead, Conjugate Gradient, BFGS, Powell, and TNC optimizers [109] and have taken the best one among the results generated by different methods. These numerics showed that for $p = 4$, Chebyshev with scaling and translation is the optimal case. For the other two cases, $p = 6$ and $p = 8$, we were able to find cases that were noticeably different from the Chebyshev case, see Figure 4.1.

4.5 Resource Estimate

Consider the model in Section 4.2. In this section, we will assume that no current active jobs exist. We will see how the number of slack bits changes as a function of relevant variables. These are T, N, $N - M$[1], p and $\frac{R_{\max}}{\epsilon}$.

As an example, let $T = 15$, $N = 10$, $N - M = 5$, and $p = 4$.

- When $\epsilon = 0$, as in the "hybrid approach" mentioned above, 120 slack bits are required.
- When $\epsilon = R_{\max}/10000$, the number of slack bits needed decreases to 91.
- When $\epsilon = R_{\max}/4000$, the number of slack bits needed decreases to 60, only half as much as the hybrid approach.
- When $\epsilon = R_{\max}/10$, the number of slack bits needed decreases to 59.
- When $\epsilon = R_{\max}/4$, the number of slack bits needed decreases to 57.
- When $\epsilon = R_{\max}$, the number of slack bits needed decreases to 53, only half as much as the hybrid approach.

This example demonstrates that it is possible to use 50% fewer slack bits and still maintain a degree of differentiability between valid states.

The decrease in the number of slack bits could be even more pronounced in other examples. Let $T = 2$, $N = 15$, $N - M = 15$, and $p = 8$.

- When $\epsilon = 0$, as in the "hybrid approach" mentioned above, 18 slack bits are required.

[1] N and M impose interchangeable constraints, so it is sufficient to consider N and $N - M$.

- When $\epsilon = R_{\max}/17500$, the number of slack variables is nearly halved to 10.
- When $\epsilon = R_{\max}/125$, we only need 6 slack bits, just a third of what is needed for the "hybrid approach".
- When $\epsilon = R_{\max}/4$, 2 slack bits are needed.
- When $\epsilon = R_{\max}/1.75$, just a single slack bit is sufficient.
- When $\epsilon = R_{\max}$, no slack bit is needed.

4.6 Conclusion

Requiring complete differentiability between valid states in the penalty function is not required for most applications in approximate optimization. In this chapter, we have demonstrated that it is possible to greatly reduce the number of slack bits and still maintain a high degree of differentiability between the valid states in the penalty function. As demonstrated in the previous section, this novel approach considerably reduces the resource requirements to use a NISQ device for the HPC job scheduling problem. Additionally, this approach is applicable to many other constrained optimization problems as well.

Chapter 4: Protein Structures with Oscillating QPacker

The quantum algorithms mentioned in the previous chapters are applicable to a wide range of problems in addition to packing, scheduling, and routing mentioned above. The protein design problem is a problem that is relevant to both literature and the industry. It can be approached in a similar way as packing, scheduling, and routing. One of the most important steps in protein modeling and design is determining the rotometers given a protein backbone [122]. The protein backbone is the overall shape of the protein in 3-dimensional space. Rotometers consist of two ingredients, the amino-acid type, and conformation. Conformation refers to the angle at which the amino acid attaches to the backbone. The task is to find these rotometers that stabilize the given backbone shape. This can be done by minimizing the energy of chemical interaction forces in the rotometer space. A previous work titled "Designing Peptides on a Quantum Computer" [122] has shown that this problem can easily be encoded as an Ising problem, the native problem Hamiltonian type that DWave quantum annealer processors use. This algorithm is named QPacker. QPacker, despite its name, does not actually solve the packing problem as in integer programming. It does (approximately) solve the rotometer problem as mentioned above. QPacker uses the default Dwave schedule, which is an approximation of linear ramp [123]. This chapter analyses the performance of the DWave quantum annealer for

the rotometer problem when a variational schedule is used. This chapter describes the research I did during my 2021 internship for Menten AI, Inc.

4.1 Optimal Annealing Schedule

As we have seen in Chapter 3, the default Dwave schedule is not always the best annealing schedule. Given a finite time, T_f, the annealing schedule that optimizes energy expectation was introduced in a recent paper [124]. The description of this schedule is given in terms of the time evolution of the Schrödinger equation; therefore does not provide an efficient way to obtain the annealing schedule. The literature is very recent on this; therefore, there are many unanswered questions. It is not known if there exists an approximation ratio such that the number of queries needed to achieve it with the optimal schedule has a lower complexity than the number of queries needed to achieve it with the linear schedule. It is not known if the optimal annealing schedule can be computed efficiently. However, an algorithm to approximate the optimal annealing schedule was given in a later paper [125]. The general idea of this approximation algorithm is to use a lower number of variables in the schedule and optimize them using a gradient-free optimization method and queries to the quantum computer. This approximate optimal annealing schedule is referred to as (Bang-Anneal-Bang) BAB for the rest of this chapter.

4.2 Dwave

Unfortunately, Dwave quantum annealer cannot natively anneal with this BAB schedule. There are several reasons why this schedule cannot work with Dwave:

- Dwave cannot handle more than 12 points in the annealing schedule; however, BAB has a constant value at the beginning and end and oscillating values between these two constants. Two points are needed to encode each constant part, leaving 8 points for oscillation at most. If we just put 1 point at each peak and valley, this allows up to 4 cycles total.

- Dwave cannot handle slopes outside of the range $[-1, 1]$ which means that $A\omega \leq \frac{1}{2\pi}$.

- Dwave cannot handle more than 12 points or slopes of $[-1, 1]$ also means that the underlying QAOA annealing curve cannot be too complicated.

- Dwave must end with Hamiltonian C at T_f and BAB ends with B at T_f.

- If Dwave starts with C at $t = 0$ the system must be in computational basis state, however, BAB assumes we start at the uniform superposition state and C at $t = 0$.

Therefore, with these restrictions in mind, I have developed an annealing schedule that fits well with Dwave's restrictions.

4.3 Sawtooth Schedule

This schedule was inspired by "Sine interpolation" of [125]. It is the weighted sum of a Sin wave

$$u_1(t) = .5 - .5\cos(\frac{2\pi t c_y}{T_f})$$

with line between $(0, 0)$ and $(T_f, 1)$

$$u_0(t) = t/T_f.$$

The linear combination of these two is

$$u(t) = (1-\alpha)u_0(t) + \alpha u_1(t) = (1-\alpha)t/T_f + \alpha(.5 - .5\cos(\frac{2\pi t c_y}{T_f})),$$

where $c_y \in \{1.5, 2.5, 3.5, 4.5, 5.5\}$ is the number of cycles in the annealing schedule. The $u(t)$ slope must be in $[-1, 1]$ between the interpolation points. The slope of $u_0(t)$ is always $1/T_f$. Placing the interpolation points at the peaks and valleys of $u_1(t)$, the change in $u(t)$ value is 1 in the time span of half a cycle $.5T_f/c_y$ giving a maximum slope of $2c_y/T_f$. Since Dwave cannot handle slopes outside of $[-1, 1]$, the restriction imposed on alpha is

$$\frac{(1-\alpha) + 2\alpha c_y}{T_f} \leq 1$$

$$\alpha \leq \frac{T_f - 1}{2c_y - 1}.$$

Since α is the interpolation coefficient, $\alpha \in [0, 1]$ therefore the restriction is

$$0 \leq \alpha \leq \min(1, \frac{T_f - 1}{2c_y - 1}).$$

We have 2 degrees of freedom; one of them is discrete, c_y, and the other one is continuous α. Please note that we do not have to consider the case of $c_y = .5$ because we place points at the peaks and valleys so $c_y = .5$ just gives us 2 points, therefore a linear interpolation, which is equivalent to setting $\alpha = 0$.

4.4 Testing Results

We have some evidence that this parameterization is useful even for metaheuristics. I did some testing by using random problem instances from internal files of Menten AI. These files have not been made public. The important point is that the overall behavior of the oscillating schedule compared to the default schedule is more favorable. Figure 4.1 shows the comparison of the annealing samples with these files. The figure was generated by averaging over different files the difference between the lowest energy generated by the oscillating schedule and the lowest energy generated by the linear schedule. This is a qualitatively useful metric because we only care about the lower energy samples, and higher energy samples are ignored. The actual difference is more useful than the relative difference because the lower the energy is in absolute terms, the more useful it is to this particular application. Also, expectation energy (which is what [125] optimizes) does not directly apply here. As a preliminary metric, Figure 4.1 demonstrates good results. However, more robust metrics such as statistical hypothesis testing could be useful to determine which schedule to move forward with.

4.5 Conclusion and Future Work

This kind of expansion in the parameter space can be useful to increase the probability of sampling from low-energy states. However, it is not immediately apparent how to find α and c_y. An expectation minimization algorithm such as Algorithm 1 of [125] was not possible to apply because the number of queries required to make to the quantum computer

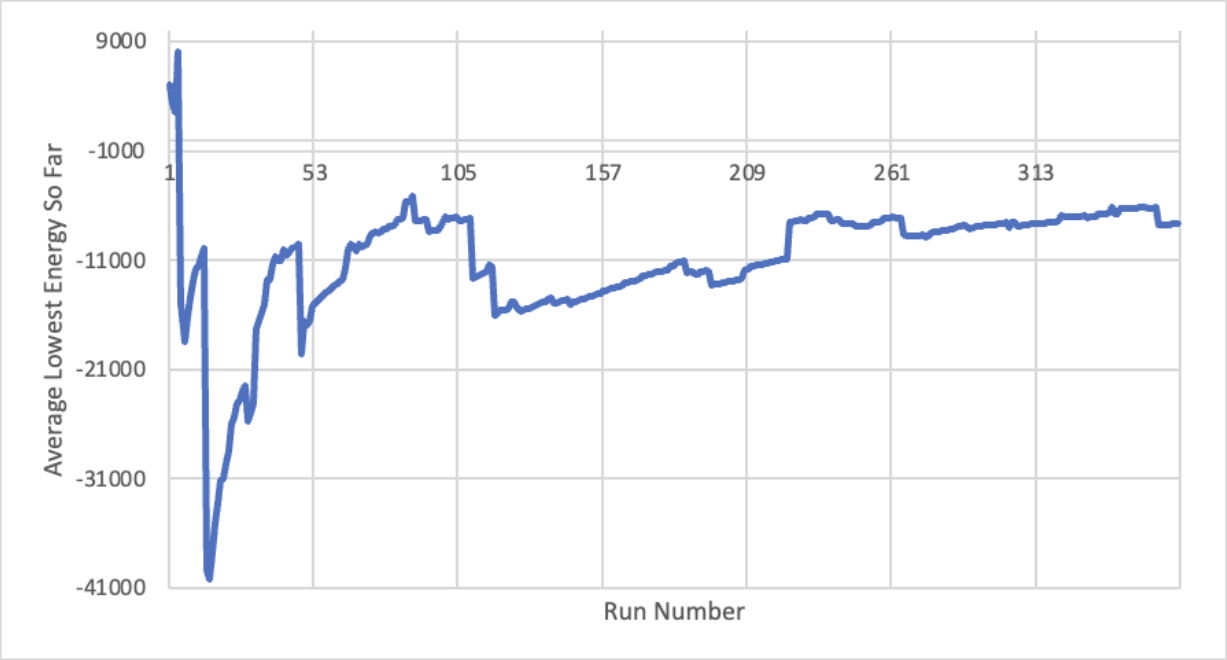


Figure 4.1: Lowest energy averaged over the number of runs for random instances of problems for $num_samples = 40, T_f = 6, c_y = 4, \alpha = .4$. The final value is -7661.478426.

to estimate the expectation is impractically high for the current DWave system. More research is needed on how these parameters can be efficiently optimized. These parameters can be efficiently optimized if any of the following were true:

- The expectation energy is a monotone function of α,
- the expectation energy is a monotone function of c_y,
- or the best α and c_y only dependent on T_f but not the problem parameters.

Given a problem and T_f, grid search can be used to sample from multiple different points for α and c_y and take the lowest energy state. However, for a fair comparison, the number of queries in the linear case must be the same as the sum of the number of queries in all grid points. All in all, the lowest sample energy and the average number of queries to achieve a certain approximation ratio are a function of problem definition, T_f, α, and c_y. This means

that the optimal α and c_y are functions of the problem definition and T_f.

Chapter 5: On the Complexity of Generalized Discrete Logarithm Problem [4]

5.1 Introduction

Shor's Algorithm for prime factorization and discrete logarithm (DLP) [66] has been one of the central topics in quantum information since its inception. Hardness assumptions regarding prime factorization and discrete logarithms provide the basis for the security, privacy, and authentication of some of the most common communication protocols [126, 127]. Shor's Algorithm has demonstrated that discrete logarithm is not hard for quantum computers. In [128], a team of researchers has defined a slight generalization of discrete logarithm and posed the open problem of finding a polynomial time quantum algorithm for this generalization with the hope that a similar generalization of Shor's Algorithm would solve this problem in polynomial time. They named this problem Generalized Discrete Logarithm Problem (GDLP). However, the time complexity of this problem was left as an open problem.

GDLP for non-abelian groups has some cryptographic applications [128, 129] in addition to standard well-known applications of DLP for abelian groups [127, 130]. In this chapter, we prove that GDLP is NP-hard and therefore is a good candidate for post-

quantum era cryptography and secret sharing which is not the case for DLP due to the ease of solution with a quantum computer. This chapter is organized as follows, we first present the necessary background and definitions in section 5.2. Next, we discuss a subproblem of GDLP which is distinct from DLP in section 4.3. In sections 4.4-4.7, we prove the NP-hardness of GDLP and some of its relevant subproblems. In section 4.8 we give an alternate approach to solve GDLP. We discuss applications in post-quantum cryptography and the implications in the field of quantum information in 4.9.

5.2 Background

5.2.1 Discrete Logarithm Problem and Its Efficient Solution

Discrete Logarithm Problem is assumed to be hard for classical computers. This assumption provides the theoretical basis for the security of Diffie-Hillman Key Exchange protocol [127]. Discrete logarithm problem is the following: given $g, y \in \mathbb{Z}_s$ and $s \in \mathbb{Z}^*$ find $x \in \mathbb{Z}_s$ such that

$$g^x \mod s = y.$$

Although this problem is assumed to not be solvable in polynomial time using classical computers [127], a polynomial time algorithm exists for quantum computers [66]. Shor's algorithm reduces DLP and factoring to Abelian Hidden Subgroup Problem (HSP) and uses a polynomial time quantum algorithm to solve Abelian HSP [66]. However, this polynomial time reduction does not generalize to GDLP.

5.2.2 Symmetric Groups and Cycle Notation

All finite groups are isomorphic to a subgroup of a symmetric group by the Cayley Theorem; therefore, we can analyze GDLP in the context of symmetric groups. Symmetric group is the set of all permutations of elements of a set of size d, and the standard notation for this set is S_d where $d \in \mathbb{Z}^*$. The elements of S_d are equivalent to $d \times d$ permutation matrices [131]. Just like with permutation matrices we can multiply to compose them and get another permutation. In this chapter, we use the "cycle notation" to denote permutations. Cycle notation consists of the indices of elements that go to the index written to its right inside the parenthesis, separated by spaces. For example

$$(1 \quad 15 \quad 2 \quad 7)$$

is the permutation where the element in index 1 goes to index 15, the element in index 15 goes to index 2, the element in index 2 goes to index 7, and the element in index 7 goes to index 1. As seen in this example, only the elements that change indices are written, and other indices do not have to be explicitly noted. When two or more cycles are written next to each other, this means multiplication and works the same as the multiplication for permutation matrices. We would like to note that in this chapter, we use zero-based indexation as our standard. Many pure mathematicians that work on permutation groups use one-based indexation, so readers should keep this in mind to avoid any confusion.

4.2.3 Generalized Discrete Logarithm Problem

Given a permutation y and a vector of permutations

$$\vec{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{q-1}),$$

the goal in GDLP is to find a vector of positive integers

$$\vec{x} = (x_{0,0}, x_{0,1}, \ldots, x_{0,q-1}, x_{1,0}, \ldots, x_{k-1,q-1})$$

for

$$\min_{\vec{x}} k$$

$$\text{s.t.} \quad \prod_{i=0}^{k-1} \alpha_0^{x_{i,0}} \alpha_1^{x_{i,1}} \ldots \alpha_{q-1}^{x_{i,q-1}} = y$$

where $q, k \in \mathbb{Z}^*$ [128].

When $\vec{\alpha}$ consists of a single permutation, this problem is equivalent to DLP. However, when $\vec{\alpha}$ has more than one element, the problem takes a vastly different structure. In this chapter, we show that this problem is NP-hard.

In the decision version, the goal is to decide if $k \leq \hat{k}$ for a given $\hat{k} \in \mathbb{Z}$. For $\hat{k} = q^{O(1)}$ GDLP is in NP as we can verify the condition satisfaction for a given $\vec{x}$ of size $q^{O(1)}$ in a polynomial number of multiplications via exponentiation by squaring. In section 4.4, we show that the decision problem is NP-hard even for $\hat{k} = 1$, thereby is in the intersection of NP and NP-hard, implying that the decision version is NP-complete. Furthermore, we

show that the decision problem remains NP-complete even when a_is are cycles with length restricted to be at most any integer ≥ 3 in section 4.7.

4.2.4 Exactly-1 Positive 3-SAT

Like the more commonly known 3-SAT problem, Exactly-1 Positive 3-SAT (Positive 1-in-3SAT) is an NP-complete problem [132]. We use reductions from this problem to GDLP to prove its NP-hardness. In Positive 1-in-3SAT, there are n literals and m clauses. Clauses can only contain positive literals, in other words, the negation of a literal is not allowed as an input in problem instances. The function of each clause is

$$f(u, v, w) = (u \wedge \neg v \wedge \neg w) \vee (\neg u \wedge v \wedge \neg w) \vee (\neg u \wedge \neg v \wedge w)$$

where u, v, w are binary variables. Each clause is conjugated with others by logical AND operators. Positive 1-in-3SAT is defined by its inputs

$$u_i, v_i, w_i \in \{z_0, z_1, \ldots, z_{n-1}\} \text{ for } i \in Z_n.$$

The problem is to determine if there exists $\vec{z} \in \mathbb{Z}_2^n$ such that

$$f(u_0, v_0, w_0) \wedge f(u_1, v_1, w_1) \wedge \ldots f(u_{m-1}, v_{m-1}, w_{m-1}) = 1.$$

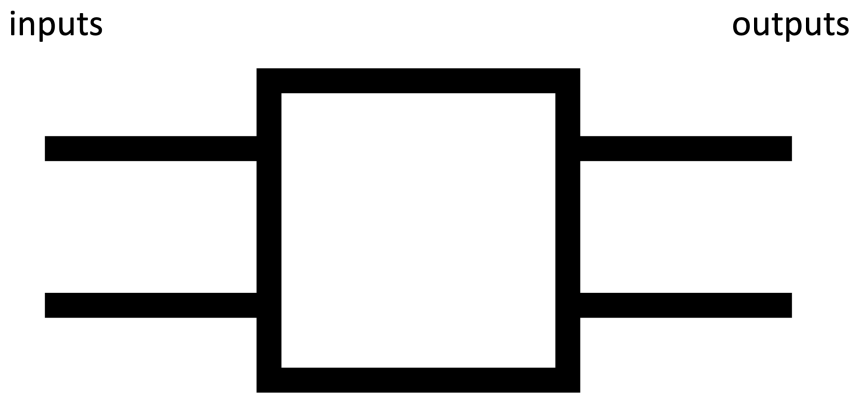


Figure 4.1: We use a square with 2 inputs and 2 outputs to denote a 2-by-2 switch.

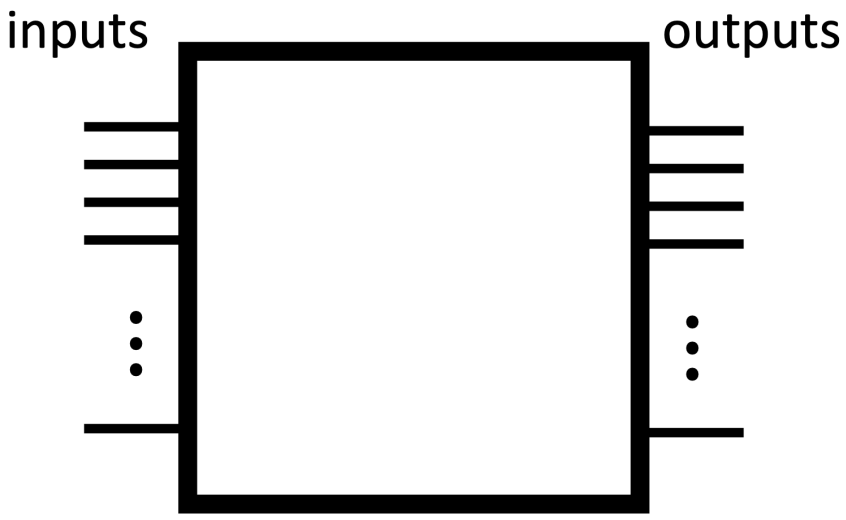


Figure 4.2: We use a square with h inputs and h outputs to denote an h-by-h.

4.2.5 Circuit Switching and Symmetric Groups

Circuit switching is a routing model where a dedicated channel is created between sender(s) and receiver(s) for the purposes of communication [133]. It is often contrasted to packet switching where instead of dedicated channels being formed packets of bits are sent to receivers via shared wires. A permutation network is a circuit switching model where there are h inputs and h outputs in the circuit. Generally, there can be anywhere between 1 and h inputs forming the sender(s). For the purposes of this chapter, we are only interested in the permutation network model that is saturated with senders, namely h senders. In the permutation network model, each sender has a distinct receiver. We can build a permutation network with 2-by-2 switches (see Figure 4.1) where the state of each switch (parallel or cross) is controlled by a single unique bit. If a permutation network can realize all $h!$ permutations, it is said to be an h-by-h switch (see Figure 4.2). One such network is the Complementary Benes Network that consists of $O(h \log h)$ 2-by-2 switches [134]. This is the same order of convergence as the number of bits needed to describe all $h!$ permutations. The routing time for Complementary Benes Network is known to be $O(h \log h)$ [134]. However, larger networks exist with lower routing times. Cellular permutation networks have $O(h^2)$ switches but only requires $O(h)$ time to route [133]. Considering $O(h)$ time is needed to even read the routing assignment, this is optimal. The routing algorithm for Cellular permutation networks works by decomposing the target permutation to the successive application of permutations of size 2 via an algorithm based on the group-theoretic structure of the target permutation. In the next section, we formulate the general circuit switching problem in terms of permutations of size 2 and the Generalized Discrete

Logarithm Problem.

4.3 Circuit Logarithm Problem

Given a permutation circuit constructed from 2-by-2 switches, deciding if a given permutation is routable on that circuit will be referred to as Circuit Logarithm Problem (CLP). This problem is equivalent to the decision of GDLP with $k \leq 1$ and bases consisting of 2 cycles. This equivalence is computed by setting the output permutation to y and the action that each switch performs to GDLP bases (a_is). In order to map the switches to the bases, we first topologically sort the switches from the inputs to the outputs. This gives us an ordering to map to bases from left to right. CLP is in NP but we do not know the complexity lower bound of this problem. However, we will use it as a building block for the complexity lower bound of GDLP.

4.3.1 Complementary Benes Network

A Complementary Benes Network is a recursively defined h-by-h switch shown in Figure 4.3. There is an efficient algorithm to route any assignment on it [134]. The property that we care about in these networks is the fact that the number of middle 2-by-2 switches that are in the cross setting is equal to the number of elements in the bottom half of the inputs that are permuted to the top half of the outputs. Additionally, the positions of these cross settings are arbitrary by the Slepian-Duguid theorem [135]. We will use this

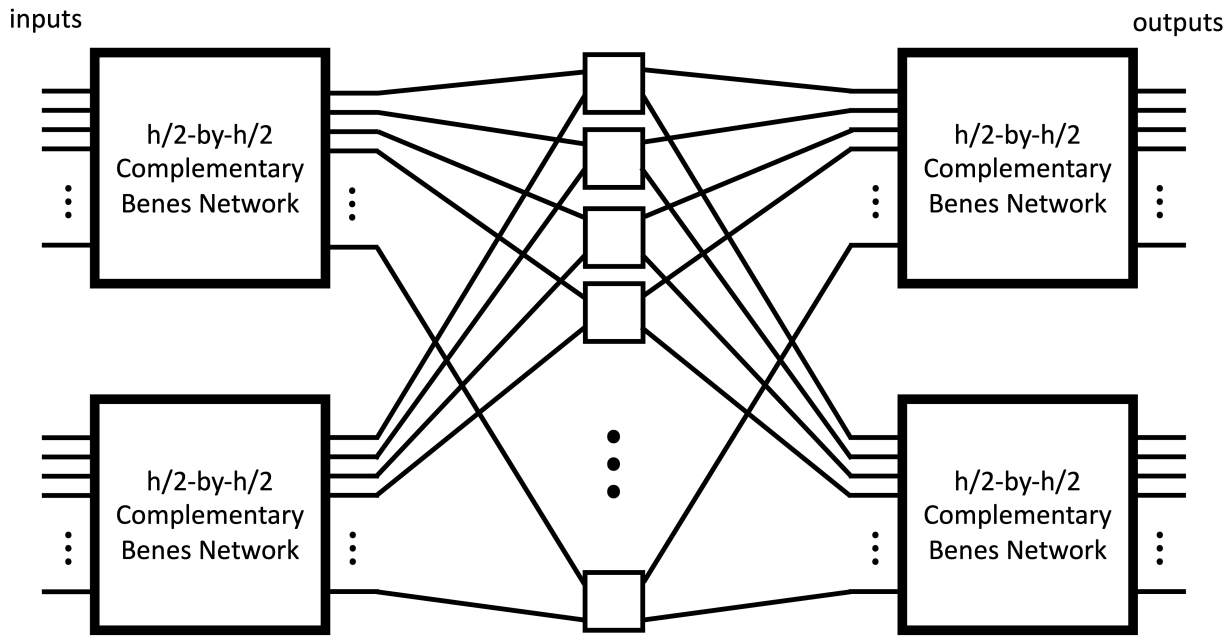


Figure 4.3: The recursive definition of Complementary Benes Network. If h is odd, the remainder 1 gets connected from the top left to the top right without getting switched in the middle.

fact to simulate 1 clause of Positive 1-in-3SAT. This is done by realizing the permutation

$$(0\,1\,2\,3\,4\,5)$$

on a 6-by-6 Complementary Benes Network (see Figure 4.4).

4.4 Converting Positive 1-in-3SAT to GDLP

In this section, we construct an instance of GDLP that is $k \leq 1$ if and only if the given instance of Exactly-1 Positive 3-SAT is satisfiable. This means that we can use an algorithm that determines if $k \leq 1$ in a given instance of GDLP to decide the satisfiability of Exactly-1 Positive 3-SAT instance. We also show that our construction is bounded by a polynomial in the number of clauses which means that if there exists a polynomial time (in

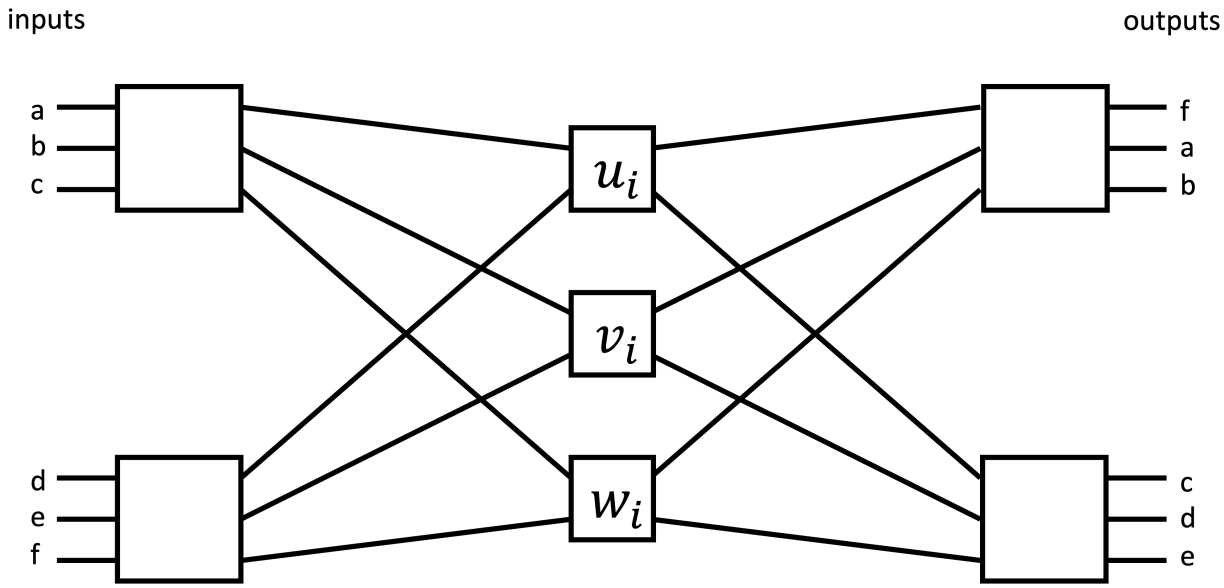


Figure 4.4: 6-by-6 Complementary Benes Network

input size) algorithm for GDLP, there also exists a polynomial time algorithm for Exactly-1 Positive 3-SAT thus, all the problems in NP. To aid with this analysis, we introduce a concept that we call "Tethered Switches".

4.4.1 Tethered Switches

In CLP we require that each switch in the base vector $\vec{\alpha}$ is of size 2 and is independently exponentiated by the element in $\vec{x}$. If we remove the requirement for independence of exponents, this allows us to impose global restrictions on the problem which can be used to simulate variables in Exactly-1 Positive 3-SAT that appear in more than 1 clause. A tethered switch is a set of switches that are controlled by a single parameter. For example, given a tether-set of switches $\{S_0, S_1, \ldots S_{p-1}\}$ which switch between top wires $T_0, T_1, \ldots T_{p-1}$ and

bottom wires $B_0, B_1, \ldots B_{p-1}$ respectively, the corresponding element in $\vec{\alpha}$ is

$$(T_0 \quad B_0)(T_1 \quad B_1)\ldots(T_{p-1} \quad B_{p-1}).$$

Now that we have shown how 3-by-3 switches and tethered switches can be written as entries in $\vec{\alpha}$, we will show how a circuit can be constructed by these elements which correspond to the given instance of Exactly-1 Positive 3-SAT.

4.4.2 Tethered Permutation Circuit Corresponding to Exactly-1 Positive 3-SAT

We first start with how each clause can be simulated. As mentioned in Section 4.3.1, if exactly 1 of the top elements ends up at the bottom, Complementary Benes Network is routable if and only if exactly 1 of the switches in the middle is in the cross position. The positions of the middle switches simulate each clause of Exactly-1 Positive 3-SAT where the top switch simulates the left literal, the middle switch simulates the center literal and the bottom switch simulates the right literal. In order to simulate all clauses of Exactly-1 Positive 3-SAT we can replicate this circuit m times. This simulates Exactly-1 Positive 3-SAT if each literal were to be unique. In order to simulate the case where literals might not be unique, the middle switches in each complementary Benes network corresponding to a non-unique literal will be placed in the same tether set as all the other switches corresponding to that literal. Now we are ready to write down GDLP corresponding to an

arbitrary case of Exactly-1 Positive 3-SAT with n literals and m clauses:

$$f(u_0, v_0, w_0) \wedge f(u_1, v_1, w_1) \wedge \ldots f(u_{m-1}, v_{m-1}, w_{m-1})$$

$$\text{where } u_i, v_i, w_i \in \{z_0, z_1 \ldots z_{n-1}\}.$$

Using a topological sort of the corresponding bases we get a GDLP instance. First (On the left), we start with left 3-by-3 components of Complementary Benes Networks which gives us the bases (from left to right):

$$(6i+0 \quad 6i+1)$$

$$(6i+1 \quad 6i+2)$$

$$(6i+0 \quad 6i+1)$$

$$(6i+3 \quad 6i+4)$$

$$(6i+4 \quad 6i+5)$$

$$(6i+3 \quad 6i+4)$$

as i increases from 0 to $m-1$. Next (in the middle), we include the bases corresponding to clause variables. Let $u^j = \{u_i | u_i = z_i\}$, $v^j = \{v_i | v_i = z_i\}$, and $w^j = \{w_i | w_i = z_i\}$. Then, the corresponding base elements can be written as:

$$\prod_{u_i \in u^j} (i+0 \quad i+3) \prod_{v_i \in v^j} (i+1 \quad i+4) \prod_{w_i \in w^j} (i+2 \quad i+5)$$

as j increases from 0 to $m-1$. Lastly, (in the right) the following bases are once again included to represent the right 3-by-3 components of Complementary Benes Networks:

$$(6i+0 \quad 6i+1)$$

$$(6i+1 \quad 6i+2)$$

$$(6i+0 \quad 6i+1)$$

$$(6i+3 \quad 6i+4)$$

$$(6i+4 \quad 6i+5)$$

$$(6i+3 \quad 6i+4)$$

as i increases from 0 to $m-1$. Finally, the target permutation y that we check the satisfiability for $k=1$ is

$$(0\ 1\ 2\ 3\ 4\ 5)(6\ 7\ 8\ 9\ 10\ 11)\ldots(6m-6$$

$$6m-5 \ \ 6m-4 \ \ 6m-3 \ \ 6m-2 \ \ 6m-1)$$

.

4.5 NP-hardness of 6-GDLP

In the previous section, we have proved that deciding $k \leq 1$ for GDLP that consists of bases with 2 cycles and cycles of the form:

$$\alpha_t = (I_{t,0} \;\; I_{t,0} + 3)(I_{t,1} \;\; I_{t,1} + 3) \ldots (I_{t,p-1} \;\; I_{t,p_t-1} + 3) \tag{4.1}$$

is NP-hard. In this section, we prove that deciding $k \leq 1$ remains NP-hard even if the bases are at most length 6 (this will be referred as 6-GDLP for the rest of the chapter). To do this, we show how to construct an instance of 6-GDLP that is $k \leq 1$ if and only if the GDLP problem constructed in the previous section is also $k \leq 1$. If we take the problem instance from the previous section and substitute each base of the form in 4.1 in-place with p_t bases:

$$(K_{t,p_t-1} \;\; S_{p-1})(I_{t,0} \;\; I_{t,0} + 3)(K_{t,0} \;\; S_{t,0}),$$

$$(K_{t,0} \;\; S_{t,0})(I_{t,1} \;\; I_{t,1} + 3)(K_{t,1} \;\; S_{t,1}),$$

$$\vdots$$

$$(K_{t,p_t-2} \;\; S_{t,p_t-2})(I_{t,p_t-1} \;\; I_{t,p_t-1} + 3)(K_{t,p_t-1} \;\; S_{t,p_t-1})$$

where all $K_{i,j}$s and $S_{i,j}$s are distinct permutation elements that are $\geq 6m$ and; keep the same y as the previous section, the truth value of $k \leq 1$ is kept the same. This is because such y requires the identity output pattern on $K_{i,j}$s and $S_{i,j}$s giving 2 possible configurations for each t, one where nothing is flipped and one where everything is flipped simulating a

tethered set entry in $\vec{\alpha}$. This gives us $15m$ bases and permutation of $12m$ elements compared to original m clauses.

4.6 NP-hardness of 4-GDLP

In the previous section, we have proved that deciding $k \leq 1$ for GDLP that consists of bases with 2 cycles and cycles of the form:

$$\alpha_r = (\mathcal{A}_r \quad \mathcal{B}_r)(\mathcal{C}_r \quad \mathcal{D}_r)(\mathcal{E}_r \quad \mathcal{F}_r). \tag{4.2}$$

In this section, we prove that deciding $k \leq 1$ remains NP-hard even if the bases are at most length 4 (this will be referred to as 4-GDLP for the rest of the chapter). To do this, we show how to construct an instance of 4-GDLP that is $k \leq 1$ if and only if GDLP problem constructed in the previous section is also $k \leq 1$. If we take the problem instance from the previous section and substitute each base of the form in (4.2) in-place with 4 bases from left to right:

$$(\mathcal{A}_r \quad \mathcal{B}_r)(L_{r,0} \quad L_{r,1})$$

$$(\mathcal{C}_r \quad \mathcal{D}_r)(L_{r,1} \quad L_{r,2})$$

$$(\mathcal{E}_r \quad \mathcal{F}_r)(L_{r,2} \quad L_{r,3})$$

$$(L_{r,0} \quad L_{r,1} \quad L_{r,2} \quad L_{r,3})$$

where $L_{r,i}$s are distinct permutation elements that are $\geq 15m$ and; keep the same y as the previous section, the truth value of $k \leq 1$ is kept the same. This is because the

output pattern for inputs $L_{r,i}$ are identity therefore the only possible outcomes of this substitution are the 2 elements generated by $(\mathcal{A}_r \quad \mathcal{B}_r)(\mathcal{C}_r \quad \mathcal{D}_r)(\mathcal{E}_r \quad \mathcal{F}_r)$. We have verified this by computing all configurations through an exhaustive search of all possible outcomes. This gives us $24m$ bases and permutation of $24m$ elements.

4.7 NP-hardness of 3-GDLP

In the previous section, we have proved that deciding $k \leq 1$ for GDLP that consists of bases with 2 cycles and cycles of the form:

$$\alpha_b = (\beta_b \quad \gamma_b)(\delta_b \quad \epsilon_b) \tag{4.3}$$

and of the form:

$$\alpha_c = (\zeta_c \quad \eta_c \quad \theta_c \quad \kappa_c). \tag{4.4}$$

In this section, we prove that deciding $k \leq 1$ remains NP-hard even if the bases are at most length 3 (this will be referred to as 3-GDLP for the rest of the chapter). To do this, we show how to construct an instance of 3-GDLP that is $k \leq 1$ if and only if the GDLP problem constructed in the previous section is also $k \leq 1$. If we take the problem instance from the previous section and substitute each base of the form in (4.3) in-place with 4 bases from left to right:

$$(\beta_b \quad \gamma_b \quad J_{b,0})$$

$$(\beta_b \quad \delta_b \quad \epsilon_b)$$

$$(J_{b,0} \quad \beta_b \quad J_{b,1})$$

$$(\beta_b \quad J_{b,0} \quad J_{b,1})$$

and substitute each base of the form in (4.4) in-place with 3 bases from left to right:

$$(\zeta_c \quad \eta_c \quad R_c)$$

$$(\zeta_c \quad \theta_c \quad \kappa_c)$$

$$(\theta_c \quad R_c)$$

where $J_{i,j}$s and R_is are distinct permutation elements that are $\geq 24m$ and; keep the same y as the previous section, the truth value of $k \leq 1$ is kept the same. This is because the output pattern for inputs $J_{i,j}$s and R_is are identity therefore the only possible outcomes of this substitution for (4.3) are the 2 elements generated by (4.3) and the only possible outcomes of this substitution for (4.4) are the 4 elements generated by (4.4). We have again verified this via an exhaustive search of all possible outcomes. This gives us $45m$ base elements and a permutation of $55m$ elements. These bases are cycles of length 2 and 3 therefore we have proved that GDLP is NP-hard even when the base elements have to be length $O(1)$ primes.

4.8 Routing on Graphs

3-GDLP can also be thought of as a type of finding non-intersecting routing paths on directed graphs. 3-GDLP consists of base elements of size 2 and size 3. A base element of

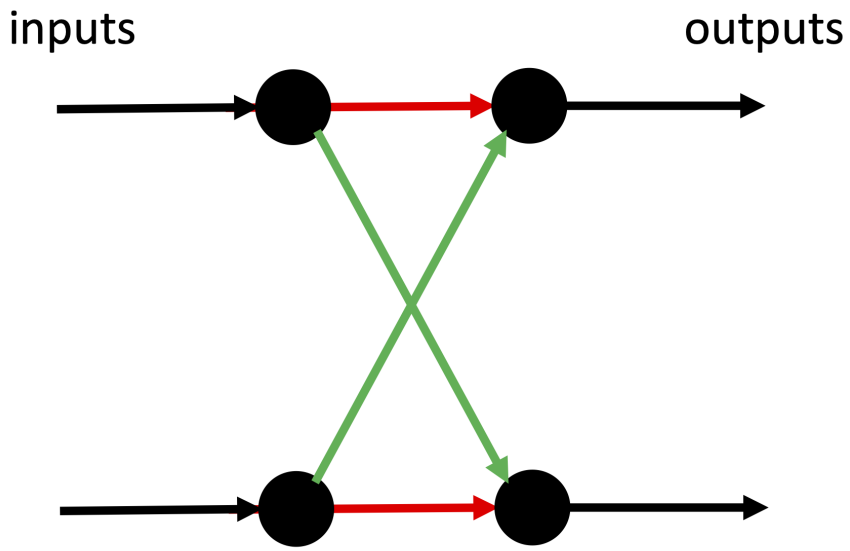


Figure 4.5: Routing paths corresponding to a 2 cycle

size 2 is equivalent to the 2 non-intersecting paths between inputs and outputs shown in Figure 4.5 with red and blue sets of arrows. A base element of size 3 has 3 distinct states, the identity, shift by 1, and shift by 2. These states correspond to the non-intersecting paths between inputs and outputs shown in Figure 4.6. Shift by 1 is the blue set of arrows followed by the yellow set. Shift by 2 is the red set of arrows followed by the green set. Identity is equivalent to two sets of disjoint routing paths. The first one is red followed by yellow. The second one is blue followed by green. By daisy-chaining copies of these two directed graphs, it is possible to construct non-intersecting routing paths on a directed graph problem instance that is equivalent to any instance of 3-GDLP.

4.9 Conclusion and Future Work

In this chapter, we prove that GDLP for symmetric groups is NP-hard even for constant prime size cycles or GDLP with cycle size limited to any integer ≥ 3. For potential

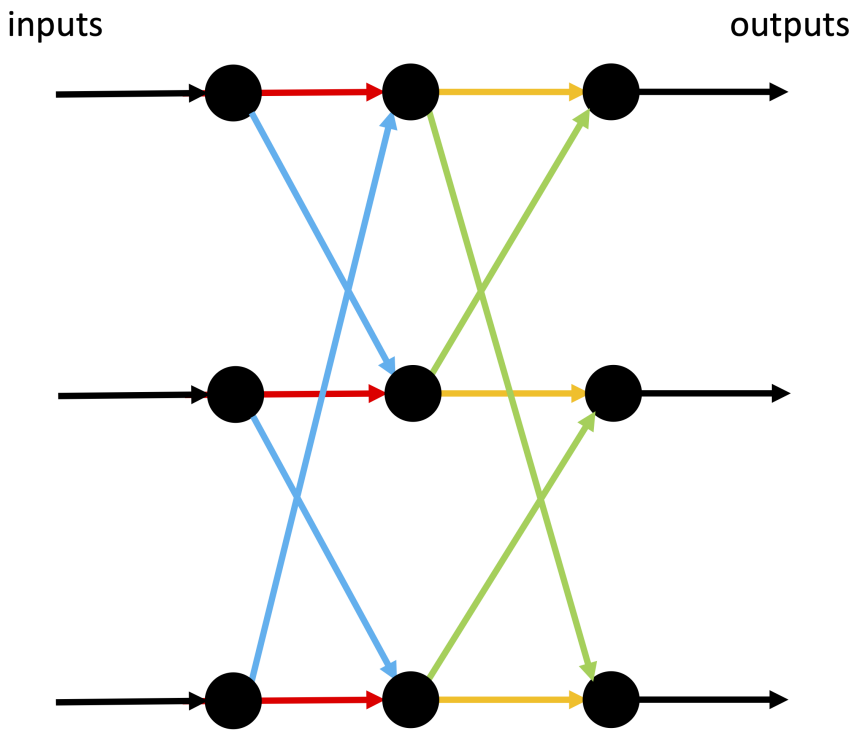


Figure 4.6: Routing paths corresponding to a 3 cycle

applications, it would be essential to analyze the classical and quantum parameterized complexities of GDLP. The relevant parameters are q, k, the number of permutation elements in all the bases, the number of bits needed to describe an instance of GDLP, and the cycle sizes of the base elements. There are several potential applications of this fact. First, GDLP for symmetric groups might be useful to develop homomorphic encryption schemes via Cayley Theorem. Next, if discrete logarithm based security protocols can be extended to generalized discrete logarithm then they can be resistant to quantum cryptoanalysis if NP $\not\subseteq$ BQP. Lastly, GDLP could be the gateway for subexponential (SUBEXP) time quantum algorithms for NP-complete problems if Shor's algorithm can be applied without pre- or post-processing exponential overhead. In [46] authors describe a query-efficient quantum algorithm to traverse a type of degree 2 graphs called "welded trees". Another way to think

about that algorithm is as a routing protocol through quantum channels with "welded tree" topology. As shown in section 4.8, we can formulate 3-GDLP as routing on degree 2 graphs. So a generalization of [46] to 3-GDLP routing graphs also has the potential to be a gateway for subexponential time quantum algorithms for NP-complete problems. All in all, the Generalized Discrete Logarithm is an interesting structure that has potential applications in quantum and classical computing, networking, and security.

Chapter 6: Concluding Remarks

Altogether, this thesis demonstrates the potential of quantum algorithms that are applicable to some of the most central computational problems in computer engineering. Using quantum computers to control large classical systems has the potential to not only amplify the power of quantum computers but also to bring us one step closer to a proof of "The power of small-depth quantum circuits" conjecture. The algorithmic techniques that are developed have current research and development applications in problems that appear in other fields such as drug development and computational biology. Furthermore, the types of analysis used such as Monte-Carlo Emulation and Switch Networks are mathematical tools that are applicable not just to quantum complexity theory but to complexity theory as a whole.

6.1 Future Work

An important future work that builds on this thesis could be an experimental implementation of polynomial schedule. It would be interesting to see how HPC scheduling application of polynomial schedule would perform in the real world. Monte-Carlo Emulation gives a consistent way to compare generative models including classical machine learning methods. It is worthwhile to study the comparison of machine learning architectures using

it. I have shown that quantum algorithms can route some networks more efficiently. It should be further explored what other networks can benefit from a similar improvement. Another direction to explore is keeping quantum algorithms in mind while designing interconnection networks since a network with desirable qualities might be hard to route with classical algorithms but might be easy to route with quantum algorithms. Lastly, the implications of NP-hardness of GDLP are yet to be explored for classical complexity theory, quantum complexity theory, and cryptography.

Appendix A: Quantum Adversarial Learning in Emulation of Monte-Carlo Methods for Max-cut Approximation: QAOA is not Optimal

A.1 Detailed Data

For Section 3.4, we only considered connected graphs of a certain size as disconnected graphs can be divided into smaller problems efficiently. We have enumerated all non-isomorphic connected graphs from 0 to 4 nodes and used $2, 4, 6$ parameters for each case. We picked QAOA as the Bang-Bang schedule since the literature for QAOA parameter optimization is well developed, giving confidence that the choice of parameters gives a total annealing time that is optimal for the distribution it generates in Bang-Bang parameterization. We have optimized QAOA with Nelder-Mead and bootstrapping. We have optimized a polynomial with gradient-descent at each step for the energy level that minimizes

$$F(x)/G(x)$$

where F is the CDF of QAOA, and G is the CDF of the polynomial at the current step. We have searched for minimum annealing time where $\forall x F(x)/G(x) \geq 1$ by hand. We did not find any complete graph where poly achieved an emulation factor on QAOA that is less than 1. However, for all of the other cases, we were able to find a value for p for which

QAOA was emulated faster with the polynomial schedule. The cases we found separations are the following:

p	g	bg	clist	t_f	$\hat{J}$
1	**(0, 2), (1, 2)**	**.393, .524**	**-.465, 1.914**	**0.898**	**.980**
1	(0, 3), (1, 3), (2, 3)	.393, .478	-.272, 1.723	0.860	.988
1	(0, 2), (0, 3), (1, 3)	.393, .468	-.358, 1.912	0.831	.965
2	**(0, 2), (0, 3), (1, 3)**	**.561, .318, .443, .771**	**-.937, 6.750, -8.057, 2.668**	**1.982**	**.947**
1	(0, 2), (0, 3), (1, 3), (2, 3)	.329, .326	-1.336, 5.572	0.654	.998
1	**(0, 2), (0, 3), (1, 2), (1, 3)**	**.393, .393**	**-.181, 1.800**	**0.757**	**.963**
2	(0, 2), (0, 3), (1, 2), (1, 3), (2, 3)	.391, .200, .258, .560	-.531, 7.954, -14.415, 7.039	1.376	.976

where $2p$ is the number of parameters, g is the graph instance, and bg is the QAOA schedule as given by $\beta_1, \beta_2, \ldots, \beta_p, \gamma_1, \gamma_2, \ldots \gamma_p$ following the notation of Ref [16], clist is the weights of the monomials in polynomial listed in increasing order by the degree, t_f is the total annealing time for polynomial schedule and $\hat{J}$ is the emulation factor for polynomial on QAOA with annealing time as the cost function. The maximum separations found for each (n, p) combination is marked with bold. We ran our simulations with 1001 points in our product formula.

For Section 3.4.4, we have summarized our data in Figures 3.2,3.3,3.4. To generate the data in Figure 3.2, we have started with $n = 5$, $p = 2$, and $t = 1.2$ as our baseline. For each of the rows, we changed one parameter at a time. We have re-sampled 10 new instances of $G(n, .7)$ with replacement and including disconnected instances for every parameter combination. We ran our simulations with 1001 points in our product formula.

To generate the data in Figure 3.3, we have started with $n = 5$ and $p = 2$ as our

baseline. For each of the rows, we changed one parameter at a time. We have re-sampled 10 new instances of $G(n, .7)$ with replacement and including disconnected instances for every parameter combination. We ran our simulations with 1001 points in our product formula.

For Figure 3.4, we have used the following graph: (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 8), (0, 9), (0, 10), (0, 11), (1, 3), (1, 5), (1, 8), (1, 9), (1, 10), (1, 11), (2, 3), (2, 4), (2, 8), (2, 11), (3, 4), (3, 6), (3, 7), (3, 8), (3, 11), (4, 7), (4, 8), (4, 10), (5, 6), (5, 9), (6, 7), (6, 8), (6, 9), (6, 10), (6, 11), (7, 9), (7, 11), (8, 11), (10, 11). We have optimized QAOA with Nelder-Mead and bootstrapping. We have optimized polynomial parameterization with Powell. We have optimized "Optimal" with gradient descent on 101 equally spaced points. We ran our simulations with 101 points in our product formula.

A.2 Successive Majorization

For Section 3.4, we have only optimized the Bang-Bang schedule once to get QAOA and then majorized it to generate our emulation factor data. However, these data points are likely upper bound on how low they can be. This is mainly due to two reasons. Firstly, the QAOA literature is well-developed, giving us good confidence that the parameters are close to optimal. Secondly, the polynomial is forced to conform to Bang-Bang, whereas Bang-Bang is free to optimize its expectation. If we had a way of optimizing the Bang-Bang schedule for a specific energy level and below as we do for polynomial, we could have had Bang-Bang and polynomial schedule iteratively majorize each other while polynomial shortens its annealing time whenever it can while keeping the majorization. This way, not only polynomial would have conformed to Bang-Bang, but also Bang-Bang would have

conformed to the polynomial while staying as the minimum-cost Bang-Bang schedule. We provide our gradient descent method for the polynomial schedule next. Please note that in Eq (A.7), if we replace $\hat{C}$ with an indicator function for an energy level and below, it allows us to optimize the schedule for that energy level.

A.3 Optimal Control Theory

In this section, we will use optimal control theory to derive what the gradients are for the clipped polynomial schedule in the main text. While we provide these results for completeness, we found in practice that optimization of the schedule using these gradients, in general, performed far worse than gradient-free optimization.

Consider a control problem of the form

$$|\dot{x}(t)\rangle = -i\hat{H}(t)|x(t)\rangle, \tag{A.1}$$

$$\hat{H}(t) = u(t)\hat{B} + (1 - u(t))\hat{C}. \tag{A.2}$$

The objective function we will initially take to minimize is

$$J = \langle x(t_f)|\hat{C}|x(t_f)\rangle, \tag{A.3}$$

but this objective function can be easily modified.

Our goal here will be to derive the conditions for optimality here given control over the function $u(t)$. The novelty of the current approach is that we will consider the control

function $u(t)$ to be a polynomial of degree p so that

$$u_0(t) = \sum_{i=0}^{p} c_i t^i. \tag{A.4}$$

Officially, we would need to consider this bounded above and below so that the actual function is

$$u(t) = \max\left(0, \min\left(1, \sum_{i=0}^{p} c_i t^i\right)\right). \tag{A.5}$$

For the first portion of this derivation, we will ignore this constraint and only put it in later.

We can follow a standard optimal control procedure here [31]. Several conditions are given, but ultimately what we want is for the change in the objective function due to changes in the control function to be zero. This condition is summarized by

$$\int_{t_0}^{t_f} dt[i\langle x(t)|\frac{\partial \hat{H}}{\partial u}|k(t)\rangle + c.c.]\delta u(t) = 0. \tag{A.6}$$

Here

$$|k(t)\rangle = \hat{U}^\dagger(t, t_f)\hat{C}\hat{U}(t_f, 0)|x(0)\rangle, \tag{A.7}$$

where $\hat{U}(t_b, t_a)$ is the unitary time evolution operator evolving us from t_a to t_b. This $|k(t)\rangle$ is a Lagrange multiplier in optimal control theory, with this form being the satisfying solution to the optimal control equations for $|k(t)\rangle$. If the target is to optimize the expectation of some goal other than $\hat{C}$, then this new target would replace $\hat{C}$ in the definition of $|k(t)\rangle$.

Naively if we did not have any constraints so that $u(t) = u_0(t)$, this could be tailored

to our problem via a chain rule

$$\delta u_0(t) = \sum_{i=0}^{p} t^i \delta c_i. \tag{A.8}$$

By the fundamental lemma of variational calculus, we then get conditions on each c_i that amount to

$$\begin{aligned} \frac{\delta J}{\delta c_i} &= \int_{t_0}^{t_f} dt \left[i\langle x(t)| \frac{\partial \hat{H}}{\partial u_0} |k(t)\rangle + c.c. \right] t^i \\ &= \int_{t_0}^{t_f} dt\, \Phi(t)\, t^i = 0, \end{aligned} \tag{A.9}$$

for all $i \in [0, p]$. This $\Phi(t)$ is the gradient of the objective function with respect to the control function $u(t)$. Therefore, the derivatives of the objective function with respect to the polynomial coefficients are just the moments of the integral of the ordinary gradient we get from optimal control.

If we want to be more precise with this, we need to consider the bounds on $u(t)$. We can try rewriting Eq. (A.5) in terms of Heaviside functions:

$$u(t) = u_0(t)\Theta\left(u_0(t)\right)\Theta\left(1 - u_0(t)\right) + \Theta(u_0(t) - 1), \tag{A.10}$$

where $u_0(t)$ is defined in Eq. (A.4).

We know simply that $\frac{\delta u_0(t)}{\delta c_i} = t^i$, so we can use the chain rule to get that

$$\begin{aligned}\frac{\delta u}{\delta c_i} = t^i \Big(& \Theta\left(u_0(t)\right)\Theta\left(1-u_0(t)\right) \\ & + u_0(t)\delta\left(u_0(t)\right)\Theta\left(1-u_0(t)\right) \\ & - u_0(t)\Theta\left(u_0(t)\right)\delta\left(1-u_0(t)\right) + \delta(u_0-1)\Big).\end{aligned} \tag{A.11}$$

We can simplify this a bit by recognizing that the delta functions in this expression take precedence over the Heaviside functions that will be active when the delta functions are nonzero. Furthermore, we get another simplification from the fact that if $\delta\left(u_0(t)\right)$ is active, then $u_0(t) = 0$, and similar when $u_0(t) = 1$. All of this leaves us with the simplified expression:

$$\frac{\delta u}{\delta c_i} = t^i \Theta\left(u_0(t)\right)\Theta\left(1-u_0(t)\right). \tag{A.12}$$

Then the gradients for the polynomial coefficients would be

$$\frac{\delta J}{\delta c_i} = \int_{t_0}^{t_f} dt \Phi(t) t^i \Theta\left(u_0(t)\right)\Theta\left(1-u_0(t)\right) = 0, \tag{A.13}$$

Appendix B: Quantum Annealing with Higher-Order Magnetic Coupling Hamiltonian and Its Job Scheduling Applications

B.1 Designing the Penalty Function

Our states have two components, the binary vector for problem variables $\vec{x}$ and the non-negative integer vector for slack variables $\vec{\Delta}$. For a given state ψ, we will denote these as ψ_x and ψ_Δ, respectively. Let $R(\psi)$ be the reward for that state, i.e. $\vec{R} \cdot \psi_x$. We define our Hamiltonian (objective) as

$$H(\theta) = Penalty(\theta) - R(\theta)$$

. We wish to design a penalty function such that:

- (a) for all valid pairs of states ψ and ϕ where $R(\psi) - R(\phi) > \epsilon$, $H(\psi) < H(\phi)$, and
- (b) for all valid states ψ and invalid states ϕ, $H(\psi) < H(\phi)$.

Let $C\vec{x} \leq \vec{b}$ be the binary constraint with m components we wish to satisfy, and p be the maximum degree of the polynomial we are willing to use. We will consider the penalty functions of the form

$$\sum_{i=0}^{T-1} w_i L_i((C\vec{x})_i - g_i\vec{\Delta}_i)$$

where w_is are some scalars that we refer to as "weights". We are assuming the normalization

$$L_i(y) \in [0,1] \text{ when } y \in \mathbb{Z}_{g_i}$$

where $g_i = \lceil (b_i+1)/2^{s_i} \rceil$ and s_i is the number of bits needed to represent the slack variable of the ith component. It is possible that 2 valid states with problem parameters ψ_x and ϕ_x have the same reward but different costs. In that case the maximum difference in is $\sum_{i=0}^{m-1} w_i$ due to $[0,1]$ normalization. Thus, condition (a) is satisfied when $\sum_{i=0}^{m-1} w_i < \epsilon$. Given a valid state ψ and an invalid state ϕ where there exists and index j such that $\phi_{x_j} = 1$ and $\psi_{x_j} = 0$. The maximum value $R(\psi) - R(\phi)$ can take is bounded by $R_{\max}$. When we compare the valid state ψ with the invalid state ϕ, we know that $\exists i$ such that $(C\vec{\psi_x} - C\vec{\phi_x})_i \geq 1$. We do not apriori know the value of $C\vec{\psi_x} - C\vec{\psi_x}$ for other indices, which means that the change in the penalty function from components other than i can be up to $w_i - \epsilon$. So we want to ensure that the i component of the penalty function, $w_i L_i$ has a high enough jump from any valid input $y \in \mathbb{Z}_{g_i}$ to the invalid input that has the lowest $w_i L_i$ value for ith penalty component. Our numerics have shown that this is g_i. This means that we can ensure condition (b) with

$$\forall i, w_i(L_i(g_i) - 1) > R_{\max} + \epsilon - w_i. \tag{B.1}$$

When we pick w_is that minimize the L_1 distance between the vector with components $w_i L_i(g_i)$ and the point with components $R_{\max} + \epsilon - w_i$ we get $w_i = \epsilon$ and

$$\forall j, w_j = \frac{\epsilon}{L_i(g_i) \sum_{i=0}^{m-1} L_i(g_i)^{-1}}.$$

Substituting this back into (B.1), we get the condition

$$\sum_{i=0}^{m-1} L_i(g_i)^{-1} > \frac{\epsilon}{1 + R_{\max}}.$$

Bibliography

[1] Cem M Unsal and Lucas T Brady. Quantum adversarial learning in emulation of monte-carlo methods for max-cut approximation: Qaoa is not optimal. arXiv preprint arXiv:2211.13767, 2022.

[2] Cem M Unsal and A Yavuz Oruc. Faster quantum concentration via grover's search. arXiv preprint arXiv:2103.09818, 2021.

[3] Cem M Unsal and Lucas T Brady. Quantum annealing with higher-order magnetic coupling hamiltonian and its job scheduling applications. work in progress, 2023.

[4] Cem M Unsal and Rasit Onur Topaloglu. On the complexity of generalized discrete logarithm problem. arXiv preprint arXiv:2212.12577, 2022.

[5] Andrew M Childs and Wim Van Dam. Quantum algorithms for algebraic problems. Reviews of Modern Physics, 82(1):1, 2010.

[6] Michele Mosca. Quantum algorithms. arXiv preprint arXiv:0808.0369, 2008.

[7] Minsung Kim, Srikar Kasi, P Aaron Lott, Davide Venturelli, John Kaewell, and Kyle Jamieson. Heuristic quantum optimization for 6g wireless communications. IEEE Network, 35(4):8–15, 2021.

[8] Kamil Khadiev, Aliya Khadieva, and Ilnaz Mannapov. Quantum online algorithms with respect to space and advice complexity. Lobachevskii Journal of Mathematics, 39:1377–1387, 2018.

[9] Scott Aaronson. Ten semi-grand challenges for quantum computing theory, 2005. URL: https://www. scottaaronson. com/writings/qchallenge. html, 2017.

[10] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information. American Association of Physics Teachers, 2002.

[11] Paul Bachmann. Zahlentheorie: th. Die analytische Zahlentheorie (1894), volume 2. BG Teubner, 1894.

[12] Edmund Landau. Handbuch der Lehre von der Verteilung der Primzahlen, volume 1. BG Teubner, 1909.

[13] John Thomas Gill. Probabilistic Turing machines and complexity of computation. PhD thesis, University of California, Berkeley, 1972.

[14] Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. Physical review, 28(6):1049, 1926.

[15] Lov K Grover. A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pages 212–219, 1996.

[16] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028, 2014.

[17] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. Physical Review E, 58(5):5355, 1998.

[18] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics, 46(4-5):493–505, 1998.

[19] Andris Ambainis and Robert Špalek. Quantum algorithms for matching and network flows. In Annual Symposium on Theoretical Aspects of Computer Science, pages 172–183. Springer, 2006.

[20] Sebastian Dörn. Quantum algorithms for matching problems. Theory Comput. Syst., 45(3):613–628, 2009.

[21] A Peruzzo et al. A variational eigenvalue solver on a photonic quantum processor. nat. commun. 5, 4213 (2014).[18] hempel, c. et al. quantum chemistry calculations on a trapped-ion quantum simulator. Phys. Rev. X, 8:031022, 2018.

[22] Sam McArdle, Tyson Jones, Suguru Endo, Ying Li, Simon C. Benjamin, and Xiao Yuan. Variational ansatz-based quantum simulation of imaginary time evolution. npj Quantum Information, 5(1), sep 2019.

[23] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. Nature, 549(7671):195–202, sep 2017.

[24] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. Nature communications, 9(1):1–6, 2018.

[25] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. Phys. Rev. E, 58:5355–5363, Nov 1998.

[26] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106, 2000.

[27] Sabine Jansen, Mary-Beth Ruskai, and Ruedi Seiler. Bounds for the adiabatic approximation with applications to quantum computation. Journal of Mathematical Physics, 48(10):102111, oct 2007.

[28] E. J. Crosson and D. A. Lidar. Prospects for quantum enhancement with diabatic quantum annealing. Nature Reviews Physics, 3(7):466–489, may 2021.

[29] Glen Bigan Mbeng, Rosario Fazio, and Giuseppe Santoro. Quantum annealing: a journey through digitalization, control, and hybrid quantum variational schemes. 2019.

[30] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. Physical Review X, 10(2), jun 2020.

[31] Lucas T. Brady, Christopher L. Baldwin, Aniruddha Bapat, Yaroslav Kharkov, and Alexey V. Gorshkov. Optimal protocols in quantum annealing and quantum approximate optimization algorithm problems. Physical Review Letters, 126(7), feb 2021.

[32] Lucas T. Brady, Lucas Kocia, Przemyslaw Bienias, Aniruddha Bapat, Yaroslav Kharkov, and Alexey V. Gorshkov. Behavior of analog quantum algorithms. 2021.

[33] Guido Pagano, Aniruddha Bapat, Patrick Becker, Katherine S. Collins, Arinjoy De, Paul W. Hess, Harvey B. Kaplan, Antonis Kyprianidis, Wen Lin Tan, Christopher Baldwin, Lucas T. Brady, Abhinav Deshpande, Fangli Liu, Stephen Jordan, Alexey V. Gorshkov, and Christopher Monroe. Quantum approximate optimization of the long-range ising model with a trapped-ion quantum simulator. Proceedings of the National Academy of Sciences, 117(41):25396–25401, oct 2020.

[34] Jérémie Roland and Nicolas J. Cerf. Quantum search by local adiabatic evolution. Physical Review A, 65(4), mar 2002.

[35] A. T. Rezakhani, W.-J. Kuo, A. Hamma, D. A. Lidar, and P. Zanardi. Quantum adiabatic brachistochrone. Physical Review Letters, 103(8), aug 2009.

[36] Shunji Matsuura, Takeshi Yamazaki, Valentin Senicourt, Lee Huntington, and Arman Zaribafiyan. VanQver: the variational and adiabatically navigated quantum eigensolver. New Journal of Physics, 22(5):053023, may 2020.

[37] Shunji Matsuura, Samantha Buck, Valentin Senicourt, and Arman Zaribafiyan. Variationally scheduled quantum simulation. Physical Review A, 103(5), may 2021.

[38] Yuki Susa and Hidetoshi Nishimori. Variational optimization of the quantum annealing schedule for the lechner-hauke-zoller scheme. Phys. Rev. A, 103:022619, Feb 2021.

[39] Dries Sels and Anatoli Polkovnikov. Minimizing irreversible losses in quantum systems by local counterdiabatic driving. Proceedings of the National Academy of Sciences, 114(20):E3909–E3916, 2017.

[40] Francesco Petiziol, Benjamin Dive, Florian Mintert, and Sandro Wimberger. Fast adiabatic evolution by oscillating initial hamiltonians. Physical Review A, 98(4), oct 2018.

[41] G. Passarelli, V. Cataudella, R. Fazio, and P. Lucignano. Counterdiabatic driving in the quantum annealing of the p-spin model: A variational approach. Phys. Rev. Research, 2:013283, Mar 2020.

[42] Jonathan Wurtz and Peter J. Love. Counterdiabaticity and the quantum approximate optimization algorithm. Quantum, 6:635, jan 2022.

[43] Andris Ambainis and Robert Špalek. Quantum algorithms for matching and network flows. In Annual Symposium on Theoretical Aspects of Computer Science, pages 172–183. Springer, 2006.

[44] Andris Ambainis. Quantum walk algorithm for element distinctness. SIAM Journal on Computing, 37(1):210–239, 2007.

[45] Sebastian Dörn. Quantum algorithms for matching problems. Theory Comput. Syst., 45(3):613–628, 2009.

[46] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pages 59–68, 2003.

[47] A Yavuz Oruc and HM Huang. Crosspoint complexity of sparse crossbar concentrators. IEEE Transactions on Information Theory, 42(5):1466–1471, 1996.

[48] Weiming Guo and A Yavuz Oruc. Explicit construction of bounded capacity sparse crossbar concentrators. In Proceedings of the Conference on Information Sciences and Systems, volume 1, page 233. Department of Electrical Engineering, Johns Hopkins University., 1996.

[49] Mark S Pinsker. On the complexity of a concentrator. In 7th International Telegraffic Conference, volume 4, pages 1–318. Citeseer, 1973.

[50] Nicholas Pippenger. Superconcentrators. SIAM Journal on Computing, 6(2):298–304, 1977.

[51] FRK Chung. On concentrators, superconcentrators, generalizers, and nonblocking networks. The Bell System Technical Journal, 58(8):1765–1777, 1979.

[52] Leonid Bassalygo. Asymptotically optimal switching circuits. Problems of Information Transmission, 17(3):206–211, 1981.

[53] Shinji Nakamura and Gerald M. Masson. Lower bounds on crosspoints in concentrators. IEEE Transactions on Computers, (12):1173–1179, 1982.

[54] Minze V Chien and A Yavuz Oruç. High performance concentrators and superconcentrators using multiplexing schemes. IEEE Transactions on Communications, 42(11):3045–3050, 1994.

[55] Emre Gündüzhan and A Yavuz Oruç. Structure and density of sparse crossbar concentrators. In Advances in Switching Networks, pages 169–180. Citeseer, 1997.

[56] Weiming Guo and A Yavuz Oruç. Regular sparse crossbar concentrators. IEEE Transactions on Computers, 47(3):363–368, 1998.

[57] Rahul Ratan and AY Oruç. Performance evaluation of inputqueued buffered sparse-crossbar packet concentrators. In Proc. Conference on Information Sciences and Systems CISS, volume 3. Citeseer, 2003.

[58] Rahul Ratan and A Yavuz Oruc. Self-routing quantum sparse crossbar packet concentrators. IEEE Transactions on Computers, 60(10):1390–1405, 2010.

[59] Grigorii Aleksandrovich Margulis. Explicit constructions of concentrators. Problemy Peredachi Informatsii, 9(4):71–80, 1973.

[60] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. Journal of Computer and System Sciences, 22(3):407–420, 1981.

[61] Noga Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. Israel Journal of Mathematics, 38(1-2):116–130, 1981.

[62] R Michael Tanner. Explicit concentrators from generalized n-gons. SIAM Journal on Algebraic Discrete Methods, 5(3):287–293, 1984.

[63] Shuji Jimbo and Akira Maruoka. Expanders obtained from affine transformations. In Proceedings of the seventeenth annual ACM symposium on Theory of computing, pages 88–97, 1985.

[64] Richard P Brent and Hsiang T Kung. A regular layout for parallel adders. IEEE transactions on Computers, (3):260–264, 1982.

[65] Vladimir Bužek and Mark Hillery. Quantum copying: Beyond the no-cloning theorem. Physical Review A, 54(3):1844, 1996.

[66] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th annual symposium on foundations of computer science, pages 124–134. Ieee, 1994.

[67] Andrew M Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. SIAM Journal on Computing, 41(6):1426–1450, 2012.

[68] John T Gill III. Computational complexity of probabilistic turing machines. In Proceedings of the sixth annual ACM symposium on Theory of computing, pages 91–95, 1974.

[69] Richard Feynman. Simulating physics with computers. Int J Theor Phys, page 467–488, 1982.

[70] Seth Lloyd. Universal quantum simulators. Science, 273(5278):1073–1078, 1996.

[71] Daniel S Abrams and Seth Lloyd. Simulation of many-body fermi systems on a universal quantum computer. Physical Review Letters, 79(13):2586, 1997.

[72] Katherine L Brown, William J Munro, and Vivien M Kendon. Using quantum computers for quantum simulation. Entropy, 12(11):2268–2307, 2010.

[73] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. Reviews of Modern Physics, 86(1):153, 2014.

[74] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. Science, 309(5741):1704–1707, 2005.

[75] Guang Hao Low and Isaac L Chuang. Optimal hamiltonian simulation by quantum signal processing. Physical review letters, 118(1):010501, 2017.

[76] Isaac H Kim. Holographic quantum simulation. arXiv preprint arXiv:1702.02093, 2017.

[77] Bela Bauer, Dave Wecker, Andrew J Millis, Matthew B Hastings, and Matthias Troyer. Hybrid quantum-classical approach to correlated materials. Physical Review X, 6(3):031045, 2016.

[78] Michael Foss-Feig, David Hayes, Joan M Dreiling, Caroline Figgatt, John P Gaebler, Steven A Moses, Juan M Pino, and Andrew C Potter. Holographic quantum algorithms for simulating correlated spin systems. Physical Review Research, 3(3):033002, 2021.

[79] U-J Wiese. Ultracold quantum gases and lattice systems: quantum simulation of lattice gauge theories. Annalen der Physik, 525(10-11):777–796, 2013.

[80] Zhi-Xin Chen, Zheng-Wei Zhou, Xingxiang Zhou, Xiang-Fa Zhou, and Guang-Can Guo. Quantum simulation of heisenberg spin chains with next-nearest-neighbor interactions in coupled cavities. Physical Review A, 81(2):022303, 2010.

[81] Erik Gustafson, Yannick Meurice, and Judah Unmuth-Yockey. Quantum simulation of scattering in the quantum ising model. Physical Review D, 99(9):094503, 2019.

[82] EE Edwards, S Korenblit, K Kim, R Islam, M-S Chang, JK Freericks, G-D Lin, L-M Duan, and Chrisopher Monroe. Quantum simulation and phase diagram of the transverse-field ising model with three atomic spins. Physical Review B, 82(6):060412, 2010.

[83] Sam McArdle, Alexander Mayorov, Xiao Shan, Simon Benjamin, and Xiao Yuan. Digital quantum simulation of molecular vibrations. Chemical science, 10(22):5725–5735, 2019.

[84] Hendrik Weimer, Markus Müller, Hans Peter Büchler, and Igor Lesanovsky. Digital quantum simulation with rydberg atoms. Quantum Information Processing, 10(6):885–906, 2011.

[85] Jiazhong Hu, Lei Feng, Zhendong Zhang, and Cheng Chin. Quantum simulation of unruh radiation. Nature Physics, 15(8):785–789, 2019.

[86] Stefan Kühn, J Ignacio Cirac, and Mari-Carmen Bañuls. Quantum simulation of the schwinger model: A study of feasibility. Physical Review A, 90(4):042305, 2014.

[87] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pages 20–29, 2003.

[88] Marin Bukov, Dries Sels, and Anatoli Polkovnikov. Geometric speed limit of accessible many-body state preparation. Physical Review X, 9(1):011034, 2019.

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

[90] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. Proc. R. Soc. Lond., 400, 1985.

[91] Xiaoling Wu, Xinhui Liang, Yaoqi Tian, Fan Yang, Cheng Chen, Yong-Chun Liu, Meng Khoon Tey, and Li You. A concise review of rydberg atom based quantum computation and quantum simulation. Chinese Physics B, 30(2):020305, 2021.

[92] Edward Farhi and Aram W Harrow. Quantum supremacy through the quantum approximate optimization algorithm, 2016.

[93] Seth Lloyd. Quantum approximate optimization is computationally universal, 2018.

[94] Shashi Guruprasad, Robert Ricci, and Jay Lepreau. Integrated network experimentation using simulation and emulation. In First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities, pages 204–212. IEEE, 2005.

[95] Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems, 14, 2001.

[96] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. Communications of the ACM, 63(11):139–144, 2020.

[97] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. Journal of artificial intelligence research, 4:237–285, 1996.

[98] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. Control strategies for physically simulated characters performing two-player competitive sports. ACM Transactions on Graphics (TOG), 40(4):1–11, 2021.

[99] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In Proceedings of the sixth annual ACM symposium on Theory of computing, pages 47–63, 1974.

[100] Ernst Ising. Beitrag zur theorie des ferro-und paramagnetismus. PhD thesis, Grefe & Tiedemann, 1924.

[101] Wilhelm Lenz. Beitrag zum verständnis der magnetischen erscheinungen in festen körpern. Z. Phys., 21:613–615, 1920.

[102] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092, 1953.

[103] Daniel Lokshtanov, Dániel Marx, Saket Saurabh, et al. Lower bounds based on the exponential time hypothesis. Bulletin of EATCS, 3(105), 2013.

[104] Michel X Goemans and David P Williamson. . 879-approximation algorithms for max cut and max 2sat. In Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, pages 422–431, 1994.

[105] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research, 134:105400, 2021.

[106] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. Physical Review E, 58(5):5355–5363, nov 1998.

[107] Gian Giacomo Guerreschi and Anne Y Matsuura. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. Scientific reports, 9(1):1–7, 2019.

[108] William L Dunn and J Kenneth Shultis. Exploring monte carlo methods. Elsevier, 2022.

[109] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods, 17(3):261–272, 2020.

[110] Siddharth Muthukrishnan, Tameem Albash, and Daniel A Lidar. When diabatic trumps adiabatic in quantum optimization. arXiv preprint arXiv:1505.01249, 2015.

[111] Aniruddha Bapat and Stephen Jordan. Bang-bang control as a design principle for classical and quantum optimization algorithms. arXiv preprint arXiv:1812.02746, 2018.

[112] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is NP-hard. Physical Review Letters, 127(12), sep 2021.

[113] Wim Lavrijsen, Ana Tudor, Jeffrey Larson, Kevin Sung, Lucy Linder, Juliane Mueller, Jarrod McClean, Ryan Babbush, Miroslav Urbanek, Costin Iancu, et al. Skquant-opt: Optimizers for noisy intermediate-scale quantum devices. In APS March Meeting Abstracts, volume 2019, pages F27–010, 2019.

[114] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci, 5(1):17–60, 1960.

[115] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. Springer, 2010.

[116] Stuart Hadfield, Zhihui Wang, Bryan O'gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algorithms, 12(2):34, 2019.

[117] Ruslan Shaydulin, Stuart Hadfield, Tad Hogg, and Ilya Safro. Classical symmetries and qaoa. arXiv preprint arXiv:2012.04713, 2020.

[118] Guido Pagano, Aniruddha Bapat, Patrick Becker, Katherine S Collins, Arinjoy De, Paul W Hess, Harvey B Kaplan, Antonis Kyprianidis, Wen Lin Tan, Christopher Baldwin, et al. Quantum approximate optimization of the long-range ising model with a trapped-ion quantum simulator. Proceedings of the National Academy of Sciences, 117(41):25396–25401, 2020.

[119] Rebekah Herrman, James Ostrowski, Travis S Humble, and George Siopsis. Lower bounds on circuit depth of the quantum approximate optimization algorithm. Quantum Information Processing, 20:1–17, 2021.

[120] Or Katz, Lei Feng, Andrew Risinger, Christopher Monroe, and Marko Cetina. Demonstration of three-and four-body interactions between trapped-ion spins. arXiv preprint arXiv:2209.05691, 2022.

[121] Hristo N Djidjev. Quantum annealing with inequality constraints: the set cover problem. arXiv preprint arXiv:2302.11185, 2023.

[122] Vikram Khipple Mulligan, Hans Melo, Haley Irene Merritt, Stewart Slocum, Brian D Weitzner, Andrew M Watkins, P Douglas Renfrew, Craig Pelissier, Paramjit S Arora, and Richard Bonneau. Designing peptides on a quantum computer. BioRxiv, page 752485, 2020.

[123] D-Wave Systems Inc. Annealing implementation and controls. docs.dwavesys.com, 12 2021.

[124] Lucas T Brady, Christopher L Baldwin, Aniruddha Bapat, Yaroslav Kharkov, and Alexey V Gorshkov. Optimal protocols in quantum annealing and quantum approximate optimization algorithm problems. Physical Review Letters, 126(7):070505, 2021.

[125] Lucas T Brady, Lucas Kocia, Przemyslaw Bienias, Aniruddha Bapat, Yaroslav Kharkov, and Alexey V Gorshkov. Behavior of analog quantum algorithms. arXiv preprint arXiv:2107.01218, 2021.

[126] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. Cryptographic communications system and method, September 20 1983. US Patent 4,405,829.

[127] Whitfield Diffie and Martin E Hellman. New directions in cryptography. In Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, pages 365–390. 2022.

[128] Lee C Klingler, Spyros S Magliveras, Fred Richman, and Michal Sramka. Discrete logarithms for finite groups. Computing, 85(1):3–19, 2009.

[129] GHJ Lanel, TMKK Jinasena, and BAK Welihinda. Cryptographic protocols using semidirect products of finite groups. International Journal of Computer Science & Network Security, 21(8):17–27, 2021.

[130] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. Guide to elliptic curve cryptography. Springer Science & Business Media, 2006.

[131] Thomas W Hungerford. Algebra, volume 73. Springer Science & Business Media, 2012.

[132] Valentin Bura. A kernel method for positive 1-in-3-sat. CoRR, abs/1808.02821, 9, 2018.

[133] A. Yavuz Oruc and M. Yaman Oruc. Programming cellular permutation networks through decomposition of symmetric groups. IEEE Transactions on Computers, 100(7):802–809, 1987.

[134] Václav E Beneš et al. Mathematical theory of connecting networks and telephone traffic. Academic press, 1965.

[135] Joseph Y Hui. Switching and traffic theory for integrated broadband networks, volume 91. Springer Science & Business Media, 2012.