A study of the performance of D-Wave quantum computers using spanning trees

By

John Spencer Hall

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Physics
in the Department of Physics and Astronomy

Mississippi State, Mississippi

May 2018

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A study of the performance of D-Wave quantum computers using spanning trees

By

John Spencer Hall

Approved:

_____

Mark A. Novotny
(Director of Thesis)

_____

Hyeona Lim
(Minor Professor)

_____

R. Torsten Clay
(Committee Member)

_____

Hendrik F. Arnoldus
(Graduate Coordinator)

_____

Rick Travis
Dean
College of Arts & Sciences

Name: John Spencer Hall

Date of Degree: May 4, 2018

Institution: Mississippi State University

Major Field: Physics

Select Appropriate Title: Add the name(s) of the person(s) heading your committee.

Title of Study: A study of the performance of D-Wave quantum computers using spanning trees

Pages in Study 77

Candidate for Degree of Master of Science

The performances of two D-Wave 2 machines (476 and 496 qubits) and of a 1097-qubit D-Wave 2X were investigated. Each chip has a Chimera interaction graph G. Problem input consists of values for the fields h_j and for the two-qubit interactions $J_{i,j}$ of an Ising spin-glass problem formulated on G. Output is returned in terms of a spin configuration $\{s_j\}$, with $s_j = +1$ or -1. We generated random spanning trees (RSTs) uniformly distributed over all spanning trees of G. On the 476-qubit D-Wave 2, RSTs were generated on the full chip with $J_{i,j} = -1$ and $h_j = 0$ and solved one thousand times. The distribution of solution energies and the average magnetization of each qubit were determined. On both the 476- and 1097-qubit machines, four identical spanning trees were generated on each quadrant of the chip. The statistical independence of these regions was investigated.

Key words: adiabatic, annealer, annealing, Chimera, D-Wave, quantum computer, quantum computing, spanning tree

DEDICATION

I would like to dedicate this research to my parents, John and Cindy Hall.

ACKNOWLEDGEMENTS

The author expresses his sincere gratitude to the many people without whose selfless assistance this thesis could not have materialized. First of all, sincere thanks are due to Dr. Mark A. Novotny, my committee chairman, for his magnanimity in expending time and effort to guide and assist me throughout the intricacies of the master's program and thesis process. Expressed appreciation is also due to the other members of my dissertation committee, namely, Dr. R. Torsten Clay and Dr. Hyeona Lim, for the invaluable aid and direction provided by them. Finally, the author would like to thank D-Wave Systems, Inc., and NASA for access to quantum computers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

DTM	deterministic Turing machine
NDTM	non-deterministic Turing machine
UTM	universal Turing machine
QTM	quantum Turing machine
RST	random spanning tree
UST	uniform spanning tree
h_i	Magnetic field strength
h_i	Magnetic field strength
$J_{i,j}$	Coupler strength
h_i	Magnetic field strength

CHAPTER I

INTRODUCTION

In this chapter, background material necessary for understanding the research will be reviewed and discussed. This includes relevant historical information up to and including recent research in the field, the importance and potential applications of quantum information theory, and necessary physical, computational, and mathematical theories, concepts, and equations.

Overview

This section will cover the history and development of the field of quantum information theory, leading up to a review of recent research in the field. The importance and potential applications of quantum computers will then be discussed. The section will end with an exploration of the motivations and objectives of the research.

Quantum Computing

Since the inception of quantum information theory, several models of quantum computing have been proposed and debated. Two of the most important and influential models are the quantum circuit model and the quantum annealing model. The first utilizes quantum logic gates to perform computations and is analogous to the classical digital circuit model, the most widely implemented model of classical computation. However,

all the results in this study rely on the second model, quantum annealing. Therefore, only background material pertaining to this model will be covered.

Background and History

One of the first important results in quantum information theory was published in 1973 by Alexander Holevo, who showed that *n* qubits cannot carry more than *n* classical bits of information, a result now known as Holevo's theorem or Holevo's bound [Holevo]. This important result establishes a fundamental limitation on the nature of quantum computation, showing that the amount of information accessible from an *n*-qubit quantum computer is no larger than an *n*-bit classical computer. Another important result came two years later, in 1975, when R. P. Poplavskii published a paper showing that it was computationally infeasible to simulate general quantum systems on a classical computer [Poplavskii], a problem repeated by Richard Feynman in 1981 when he proposed a model for a quantum computer [Feynman].

In 1982, a year after Feynman proposed his model, William Wootters and Wojciech Zurek, and independently Dennis Dieks, prove that it is impossible to create an identical copy of an arbitrary, unknown quantum state, a result known as the no-cloning theorem [Wootters, Dieks]. This leads to several important consequences, including the realization that classical error correction techniques cannot be utilized on quantum states. However, over a decade later, in 1995 and 1996, Peter Shor and Andrew Steane independently proposed methods for quantum error correction which do not violate the no-clone theorem [Shor, Steane], overcoming a formerly intractable limitation of quantum computing.

Over the next decade, advances continued to be made both in quantum information theory and in several specific models of quantum computation, including quantum circuit computing, NMR quantum computing, optical quantum computing, and quantum dot computing. However, it is not until 2007 that Canadian company D-Wave Systems, Inc., claims to have developed a 28-qubit quantum computer based on the quantum annealing model [D-Wave]. A year later in 2008, they claimed to have increased this number to 128-qubits [D-Wave]. In 2011, after further developments, they release their 128-qubit chip as the D-Wave One, which they claim to be the first commercial quantum computer [D-Wave]. Several reviews of the literature of quantum annealing have been published [Albash, Das].

Importance and Potential Applications

As the size of classical transistors approach the quantum limit, the exponential growth in computing power that has been the norm for the past several decades will begin to slow and even plateau. Even with the advent of high-performance computing and new computational techniques, such as parallel and distributed computing, this physical limit on the size of individual transistors means it will be difficult to study ever more complex and computationally-intensive problems. Additionally, as noted in the last section, certain problems can never be solved efficiently on a classical computer. These two facts highlight the importance of the growing field of quantum computing, which is expected to solve some problems more efficiently, requiring less time and other resources. This speed up would have a significant impact on several important, computationally-intensive fields, including cryptography, weather prediction and climate modeling, machine

learning and artificial intelligence, mathematical optimization, and the simulation of quantum systems.

Motivation and Objectives

Though significant theoretical foundations have been laid, quantum computing is still in its infancy experimentally. Significant technological and engineering hurdles still have to be overcome in order to realize a practical universal quantum computer.

The goal of this research was to explore ways in which to test the performance of adiabatic quantum annealers, in particular the D-Wave Two and the D-Wave 2X. The goal was to find ways to test and to validate the performance as the number of qubits are scaled in future generations of adiabatic quantum computers. It is envisioned that, in the future, the number of qubits will exceed the limits of simulation and comparison on a classical supercomputer. Random spanning tree problems are a good candidate for such validation and testing. The ground state energy and spin configuration are known, and thus the solutions returned by the D-Wave 2 and the D-Wave 2X can be checked without the use of classical supercomputers. Furthermore, every spanning tree incorporates every vertex of the graph, thereby testing every qubit, and the ensemble of all random spanning trees utilizes every edge of the graph, thereby testing all the couplers between the qubits. Because of this, these types of problems can be used as a common test for different sizes of lattices; additionally, the spanning tree approach works for any graph G. For smaller chips, the results of experiments on the two quantum computers can be compared with simulations on classical computers, namely simulated annealing and simulated quantum annealing.

Relevant Physical Theory

In this section, foundational physical concepts and definitions relevant to the research will be outlined and discussed, although knowledge of basic quantum theory is assumed of the reader. The section begins with an introduction to quantum information theory, leading into a discussion of the basic elements of quantum computing. The section ends with an overview of quantum annealing and the adiabatic theorem, which together form the basis of adiabatic quantum computing.

Quantum Information Theory

Using a generalized theory of the quantum mechanics of open systems and a generalized concept of observables, called semi-observables, it can be shown that quantum information theory is a generalization of classical information theory [Ingarden]. However, several important results show that quantum information differs in exceptional ways from classical information.

Qubits, Superposition, and Entanglement

A quantum bit, or qubit, is the fundamental unit of quantum information, analogous to the classical bit of classical information theory. A qubit is an example of a two-state quantum system and can be realized physically in a number of ways. Two of the most common examples are photon polarization and electron spin. For photon polarization, the two states would be horizontal and vertical polarization, and for electron spin, the two states would be up and down. Just as for classical bits, we can abstract away

from the particular physical implementation and label the two states |0> and |1>, where we have used the bra-ket, or Dirac, notation to emphasize that these are quantum states. For example, we might associate horizontal polarization of a photon with |0> and vertical polarization with |1>.

Despite the superficial similarities between bits and qubits, there are significant, fundamental differences between the two. Both are two-state systems, but unlike bits, which must always be in one state or the other, qubits can be in a linear superposition of both states. This can be represented mathematically as

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.1}$$

where α and β are, in general, two complex numbers, constrained only by the condition that $|\alpha|^2 + |\beta|^2 = 1$. However, like classical bits, only one of two states can ultimately be measured: |0> or |1>. The probability of measuring |0> is $|\alpha|^2$, and the probability of measuring |1> is $|\beta|^2$. Thus, even though a qubit can "hold" more classical information than a bit, the amount of classical information that can be retrieved is the same; this is the essence of Holevo's theorem [Holevo].

Another important distinction between qubits and classical bits is entanglement, which is a uniquely quantum phenomenon. Entanglement allows particles to be correlated in ways that are not possible classically. Particles become entangled when they are created in such a way or they interact in such a way that the state of each particle cannot be described independently. Instead, only the state of the entire system can be specified. One of the simplest, canonical examples of an entangled state is the first Bell state:

$$|\phi^+\rangle = (|00\rangle + |11\rangle) / \sqrt{2}. \tag{1.2}$$

In this example, there are only two qubits, and the two states |00> and |11> represent the only two possible states of the entire system. The state |00> is the state in which both qubits are found to be in the |0> state, and the state |11> is the state in which both are found to be in the |1> state. Thus, since these are only two possible states of the system, if the first qubit is found to be in the |0> state, then the other qubit must also be in the |0> state. In other words, when the two qubits are entangled in this way, the probability that both end up in the same state is exactly 1, even if they are physically separated by an arbitrary distance. If they were not entangled, then this correlation would not hold.

Entanglement is necessary for realizing the advantages of quantum computation over classical computation, since non-entangled states can be efficiently simulated by classical methods. Specifically, since every single-qubit state (1.1) can be rewritten as

$$|\phi\rangle = \cos(\theta)|0\rangle + e^{i\psi}\sin(\theta)|1\rangle, \tag{1.3}$$

which has only two real parameters, θ and ψ, every non-entangled qubit can be represented to n-bit accuracy by $2n$ classical bits. This means that non-entangled quantum computation can be simulated efficiently by classical computation. Additionally, quantum teleportation and superdense coding, a technique which uses one qubit to send two classical bits of information [Bennett], both make use of entanglement, as do some models of quantum cryptography [Ekert].

Size of the State Space

The source of quantum computers' advantage over classical computers lies in the number of states accessible during computation. As discussed above, non-entangled

qubits, like the ones represented by equation (1.3), can hold $2n$ classical bits of information to n-bit accuracy. Assuming we have m qubits, this represents $2mn$ classical bits of information; or, put another way, since each of the m qubits is, in general, in a superposition of the two states |0> and |1>, the number of basis states accessible while performing a computation is $2m$. However, since this represents an increase by only a constant factor, it provides almost no computational advantages over classical computers.

On the other hand, entangled qubits, like the pair represented by equation (1.2), can hold $2^{m+1}n$ classical bits of information to n-bit accuracy, meaning 2^m basis states are accessible during computation. This represents a significant, *exponential* increase over classical computers. In comparison, only one of these states is available to a classical computer at any given time during a computation.

Decoherence

One of the biggest obstacles to achieving practical quantum computers is decoherence, which occurs when a quantum system interacts with its surrounding environment in a way that is thermodynamically irreversible. Essentially, decoherence results in the entanglement between some or all of the qubits being lost. Decoherence can be caused by a variety of factors, from small mechanical vibrations to minute thermal fluctuations. Decoherence can even occur when interacting with a purely quantum system [Novotny]. This extreme sensitivity of quantum systems to their surrounding environment means that quantum computers need to be in a highly controlled and extreme environment in order to operate effectively.

Quantum Annealing

Quantum annealing is both a physical process and a heuristic technique that was first proposed in 1994 [Finilla] and was reformulated four years later in 1998 for the transverse Ising model [Kadowaki]. In this form, an initial superposition of states with equal probabilities is prepared and is allowed to evolve according to the time-dependent Schrödinger equation, with the probabilities of all the states changing according to the time-dependent strength of a transverse magnetic field. This allows for quantum tunneling between states. It has been demonstrated experimentally as well as theoretically that quantum annealing can indeed outperform thermal annealing in certain cases, especially where the potential energy landscape consists of very high but thin barriers surrounding shallow local minima [Nielsen].

Adiabatic Theorem

The adiabatic theorem was first proposed in 1928 by Max Born and Vladimir Fock [Born]. Suppose that at some initial time t_0 a quantum-mechanical system has an energy given by the Hamiltonian $H(t_0)$, and the system is in an eigenstate of $H(t_0)$. Changing conditions modify the Hamiltonian in a continuous manner, resulting in a final Hamiltonian $H(t_1)$ at some later time t_1. The adiabatic theorem states that as the time $\tau = t_1 - t_0 \to \infty$, known as an adiabatic process, the final state will be an eigenstate of the final Hamiltonian $H(t_1)$.

Adiabatic Quantum Computation

The D-Wave computer utilizes a model of computation called adiabatic quantum computing, which has been shown to be computationally equivalent to the quantum circuit model [Mizel]. First, a Hamiltonian is found whose ground state encodes the solution to the problem of interest. This is the problem Hamiltonian $H_P = H(t_1)$. Then a system with a simple Hamiltonian is prepared and initialized to the ground state of the driver Hamiltonian $H_0 = H(t_0)$. Finally, the simple Hamiltonian is evolved adiabatically to the desired Hamiltonian H_P according to the equation

$$H(t) = A(t)H_0 + B(t)H_P, \tag{1.4}$$

and by the adiabatic theorem, the system remains in the ground state, so at the end the state of the system describes the solution to the problem. This idea is illustrated in Figure 1.1 below [Vinci], which shows for the D-Wave Two the annealing schedule for $A(t)$ and $B(t)$.

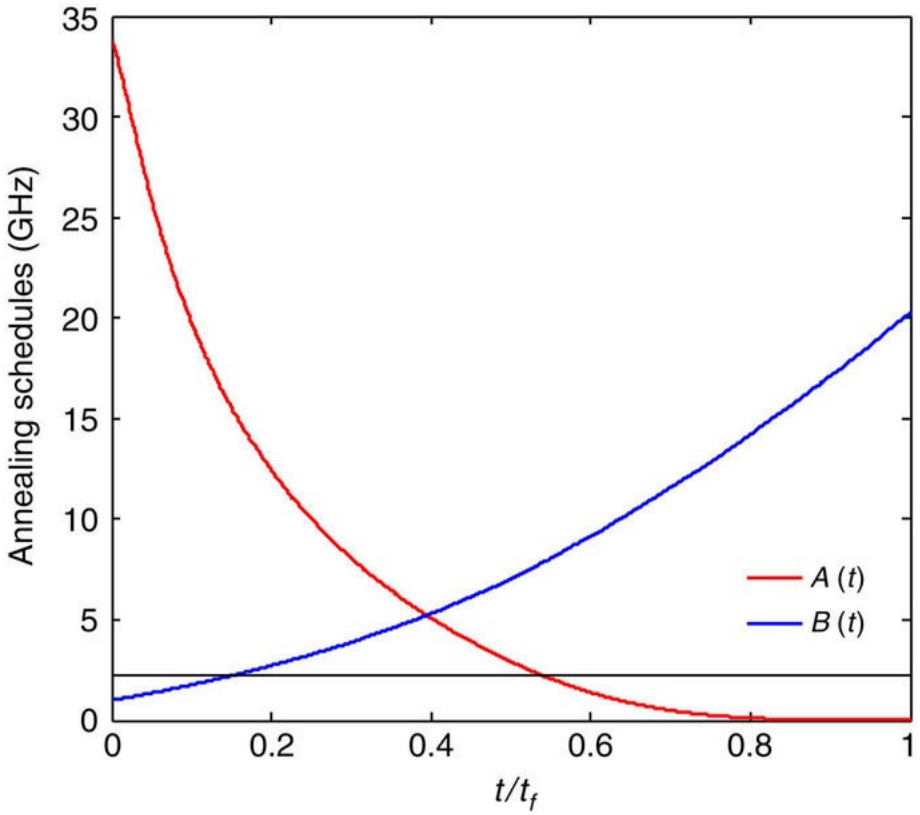

Figure 1.1 Adiabatic Quantum Annealing of the D-Wave Two

Relevant Computational Theory

Computational complexity theory is a branch of mathematics and computer science that studies the inherent difficulty of computational problems and tries to determine the relationships between them. A computational problem can be thought of as the collection of all the particular *instances* of that problem together with the solutions for every instance. For example, factoring an integer is a computational problem, while factoring the integer 30 is an instance of that problem. Computational complexity theory is closely related to the field of computability theory, which tries to determine which problems are computable in principle, regardless of the resources needed. It is also

related to the field of algorithm analysis, which examines the efficiency of particular algorithms by determining the amount of resources that they need.

Complexity Classes

At the highest level, there are several categories into which all problems fall. Some of the most important of these categories are decision problems, function problems, optimization problems, search problems, and counting problems. These categories are closely related to one another; for example, every counting problem has a corresponding search problem, and every search problem has a corresponding decision problem. In fact, every type of problem can be recast as a decision problem without significantly changing its complexity [Nielsen]. It is for this reason that decision problems are the usual objects of study in computational complexity theory. Decision problems can be further classified as either decidable or undecidable, meaning they can either be computed, in principle, or not. This is the main concern of computability theory.

The next piece of information used to help describe and classify the computational complexity of a problem is the type and amount of resources needed to reach a solution. For problems that are decidable, there are two resources that are usually considered when determining the difficulty of a particular problem or the efficiency of a particular algorithm: time and space. Of these two, time is usually the more limiting factor, since space is more easily acquired, especially when considering advances in computer memory and storage technology. As one might expect, the amount of resources needed to solve a problem often depends on the size of the input, that is, the particular instance. For this reason, the amount of a resource that is needed is specified as a function

of the size of the input. Resource usage most commonly grows at either a logarithmic, polynomial, or exponential rate with the size of the problem instance.

Finally, problems and algorithms are also divided according to the particular model of computation used when determining the amount of resources needed. The two primary models used are the deterministic Turing machine (DTM) and the non-deterministic Turing machine (NDTM). The DTM is the classic, standard model of computation. In this model, only one action is prescribed for every state of the machine; in other words, there is only one path forward at every step of the computation. The NDTM is an alternative model of computation which is equivalent to the DTM [Nielsen]. Unlike its deterministic counterpart, several actions may be prescribed for one or more states of the machine, meaning there may be several paths one could take at a particular step of the computation. Some problems and algorithms are more naturally suited to this model, making them easier to understand, study, and conceptualize. The most important complexity classes are listed along with a short description in Table 1.1 [Aaronson].

Table 1.2 Create a short, concise table title and place all detailed caption, notes, reference, legend information, etc in the notes section below

Name	Description
SPACE	deterministic space
EXPSPACE	deterministic exponential space
PSPACE	deterministic polynomial space
L	deterministic logarithmic space
NSPACE	non-deterministic space
NEXPSPACE	non-deterministic exponential space
NPSPACE	non-deterministic polynomial space
NL	non-deterministic logarithmic space
TIME	deterministic time
EXP	deterministic exponential time
P	deterministic polynomial time
NTIME	non-deterministic time
NEXP	non-deterministic exponential time
NP	non-deterministic polynomial time

NOTE: Alternative names exist in the literature for some of the complexity classes listed above, but they are easy to identify based on the patterns that can be discerned from the table, *i.e.*, EXPTIME or DEXPTIME are different names for EXP. Many classes not listed in this table also follow a similar naming scheme.

Several important relationships are known to exist between these classes. The space hierarchy theorems establish that both the deterministic and the non-deterministic space classes form proper subsets [Nielsen]:

$$L \subset PSPACE \subset EXPSPACE \text{and} NL \subset NPSPACE \subset NEXPSPACE. \quad (1.5)$$

The time hierarchy theorems establish similar relationships for both the deterministic and the non-deterministic time classes [Nielsen]:

$$P \subset \text{EXPand} NP \subset NEXP. \quad (1.6)$$

Savitch's theorem [Savitch] implies that

$$PSPACE = NPSPACE \text{and that} EXPSPACE = NEXPSPACE, \quad (1.7)$$

and together with the space hierarchy theorems, it also implies that

$$NL \subset PSPACE. \quad (1.8)$$

Furthermore, the following classes are known to form subsets, but it is not known whether or not they form proper subsets [Nielsen]:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq \text{EXP} \subseteq NEXP \subseteq EXPSPACE. \quad (1.9)$$

P and NP

Arguably, the most important and well-known complexity classes are P and NP. The importance of the complexity class P can be traced to the Cobham-Edmonds thesis, which argues that only those problems which can be solved in polynomial time are computationally feasible [Cobham]. Although there are exceptions, this assertion provides a good and useful rule of thumb. The importance of the complexity class NP is due to the number of important problems that are known to be in this class but are not known to be in P, including many search and optimization problems. Some examples

include the subset sum problem, the integer factorization problem, the graph isomorphism problem, the traveling salesman problem, and the Boolean satisfiability problem [Nielsen].

Formally, the complexity class NP is defined as the set of all decision problems that can be solved in polynomial time by a non-deterministic Turing machine. An alternative but equivalent definition of the class NP is the set of all decision problems whose affirmative answers can be *verified* in polynomial time by a deterministic Turing machine. This second definition is often viewed as being more intuitive than the first, and it has the additional advantage of relating the class NP to the more familiar deterministic model of computation.

There are several important open questions in computational complexity theory involving P and NP. Perhaps the most famous is whether or not P = NP; while it is known that P is a subset of NP, it is not known whether it is a proper subset or not. As explained above, only those problems which are in P are considered to be computationally feasible, but there are many important problems that are known to be in NP that have not yet been proven to be in P. If it is shown that P = NP, it would imply that algorithms exist which would solve these important problems in polynomial time. However, many experts believe that P ≠ NP, meaning that some of those problems will remain intractable for classical computers [Nielsen]. The suspected relationship between P and NP is illustrated in Figure 1.2.

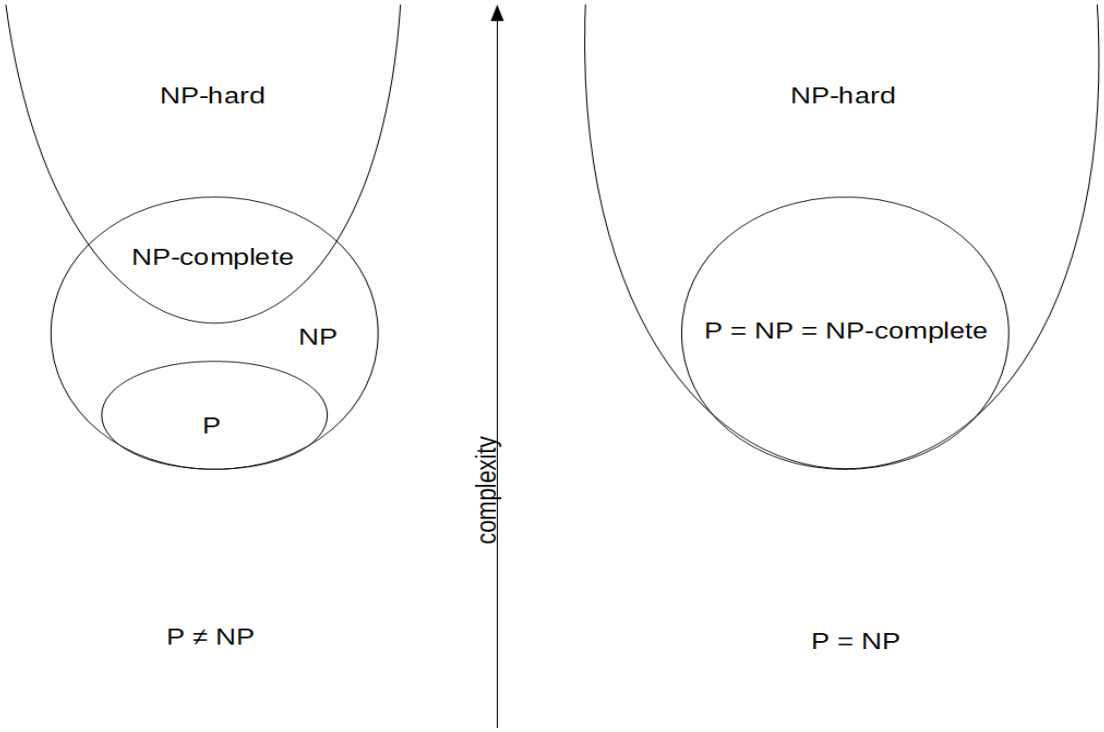

Figure 1.2 The Two Possible Relationships Between P and NP

NOTE: Informally, NP-complete problems can be thought of as the hardest problems in NP, while NP-hard problems are those problems which are at least as hard as NP-complete problems. Analogous divisions can be similarly defined for other complexity classes.

Probabilistic and Quantum Complexity Classes

The quantum analog of a universal Turing machine (UTM) is a quantum Turing machine (QTM), also called a universal quantum computer. This model of computation was first described in 1985 [Deutsch] and was later shown to be equivalent to the quantum circuit model [Yao]. Though perhaps counter-intuitive, the QTM model is computationally equivalent to all other Turing machine models [Nielsen], meaning that there is no problem that a quantum computer can solve that a classical computer cannot. Another way of stating this is that quantum computers, like classical computers, can only

solve decidable problems, not undecidable ones. It also means that a quantum computer can be simulated using a classical computer, albeit inefficiently.

Just as there are complexity classes defined using the classical deterministic and non-deterministic Turing machines, there are complexity classes defined using the quantum Turing machine. The most important of these is, perhaps, the complexity class QMA (quantum Merlin Arthur). This class contains the set of decision problems for which YES-proofs are verifiable in polynomial time on a QTM and are accepted 66% of the time, and for which NO-proofs are rejected 66% of the time. More informally, it can be thought of as the quantum analog to the classical complexity class NP. Several important probabilistic and quantum complexity classes are shown in Table 1.2 [Aaronson].

The class QMA is important because, in addition to subsuming many other important classes such as P and NP, there are interesting problems that are known to be in QMA but are not known to be in other classes such as NP. One such problem is the local Hamiltonian problem, which can be thought of as the quantum variant of the maximum satisfiability (MAX-SAT) problem. A list of known QMA-complete problems has been compiled [Bookatz]

Table 1.3 Important Probabilistic and Quantum Complexity Classes

Name	**Description**
PP	probabilistic polynomial time
BPP	bounded-error probabilistic polynomial time
RP	randomized polynomial time
ZPP	zero-error probabilistic polynomial time
PL	probabilistic logarithmic space
BPL	bounded-error probabilistic logarithmic space
RL	randomized logarithmic space
ZPL	zero-error probabilistic logarithmic space
QMA	quantum Merlin Arthur
BQP	bounded-error quantum polynomial time
PostBQP	BQP with postselection
MA	Merlin Arthur

Although it is believed that a quantum computer cannot efficiently solve all NP problems, it is suspected to be able to solve at least some of them efficiently. The class of problems which can be solved in polynomial time by a quantum computer with an error rate of at most 33% is known as BQP (bounded-error quantum polynomial time). While it is known that P is a subset of BQP, the relationship between NP and BQP is not fully understood. The suspected relationship of BQP to some other complexity classes is shown in Figure 1.3.

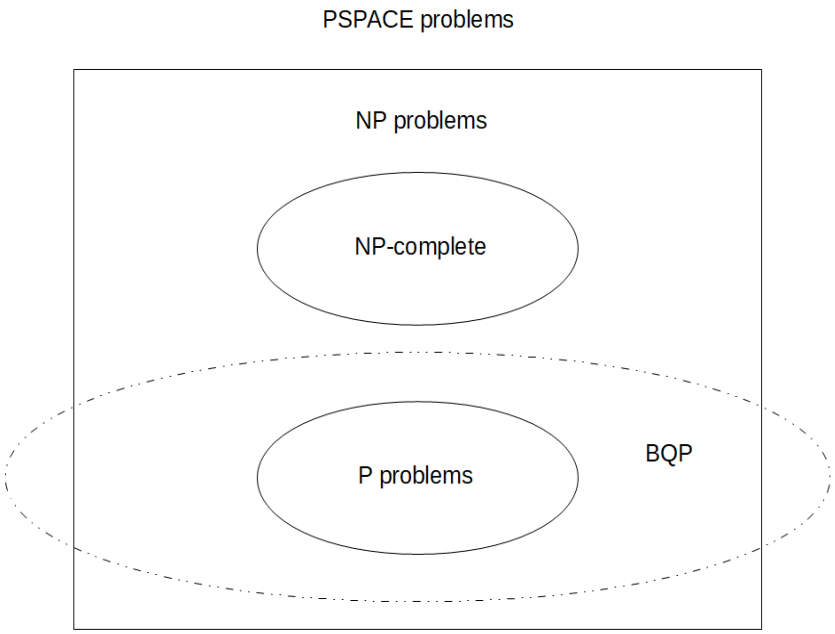

Figure 1.3 Suspected Relationship Between BQP and Other Classes

Classical vs. Quantum Computing

Although classical computers can, in principal, solve all decidable problems, it cannot solve all of them efficiently. Per the Codham-Edmonds thesis, only those problems which belong to the complexity class P can be solved in a reasonable time on a classical computer. Therefore, as shown in Figure 1.3, quantum computing holds the potential to solve a larger class of problems in a reasonable time than classical computers.

Spanning Trees

In this section, the mathematical concept of spanning trees will be discussed. First, spanning trees will be defined and their important properties will be reviewed. A special type of spanning tree, the random spanning tree, which is especially relevant to the research, will then be presented, followed by a discussion of the advantages and disadvantages of using spanning tree problems as a test or benchmark for quantum annealers.

Definition and Relevant Properties

A tree is a connected, undirected graph with no loops. It is a spanning tree of a graph G if it includes every vertex of G and is a subgraph of G; that is, every edge in the tree belongs to G. A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no loops, or as a minimal set of edges that connect all vertices. An example of a spanning tree is shown in Figure 1.4.

A given graph G may have more than one spanning tree. The number $t(G)$ of spanning trees that exist for a graph G can be found using Kirchhoff's matrix tree theorem [Kirchhoff]. The procedure involves constructing a matrix from the graph G and then computing its determinant, which will be equal to the maximum number of spanning trees $t(G)$. Furthermore, there are algorithms [Nielsen] that belong to the complexity class P which compute the determinant of a matrix, meaning that $t(G)$ can be found by a classical computer in a reasonable amount of time.

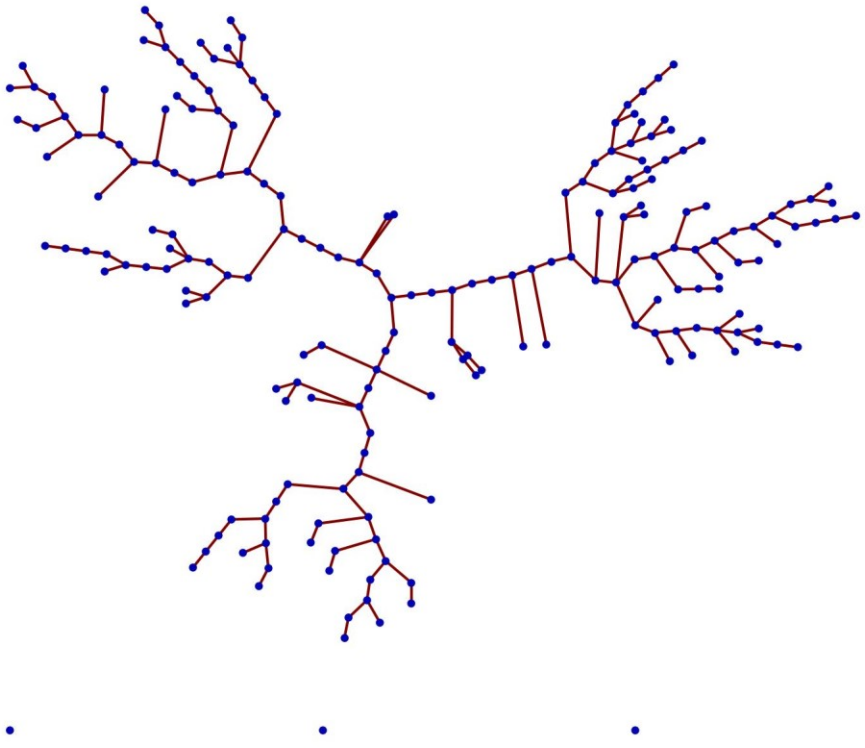

Figure 1.4 A Spanning Tree

Another property of spanning trees that was found to be useful is the relationship between the number of vertices V in the graph G and the number of edges E in any spanning tree of G. The relationship between V and E is

$$E = V - 1. \tag{1.8}$$

This is a necessary, but not sufficient, condition for a subgraph of G to be a spanning tree, and thus serves as a useful check on the graph problems used in this study.

Random Spanning Trees (RSTs)

A spanning tree chosen randomly from among all the spanning trees of a connected graph G is called a random spanning tree (RST). A spanning tree chosen randomly with equal probability from among all the spanning trees is called a uniform spanning tree (UST). It is a special type of random spanning tree. There exist several algorithms for generating uniform spanning trees; two of the simplest and most popular are the Aldous-Broder algorithm, described independently by both men [Aldous, Broder], and Wilson's algorithm [Wilson]. Either can be used to generate uniform spanning trees in polynomial time by a process of taking a random walk on the given graph and erasing the cycles created by this walk. This process is called a loop-erased random walk. Wilson's algorithm is slightly less intuitive but also more efficient: it is always as fast as or faster than the Aldous-Broder algorithm [Wilson].

The algorithm chosen for use in this study is the Aldous-Broder algorithm. The algorithm begins by randomly choosing a vertex V_1 of the graph G. This vertex is added to a list of vertices that have been visited. Then, another vertex V_2 is randomly chosen from all the vertices which are connected by an edge to V_1. If V_2 is in the list of vertices

that have already been visited, then we randomly choose another vertex V_3 from all the vertices which are connected by an edge to V_2. However, if it is not in the list, then it is added to the list, and the edge E_{12} connecting vertices V_1 and V_2 is added to the spanning tree. This process is repeated until all the vertices of G have been visited. The collection of edges E_{ij} obtained using this method make up a uniform spanning tree of the graph G [Aldous, Broder].

Motivation for using RSTs

Uniform spanning trees have many properties that make them appealing candidates to use as a test or benchmark for adiabatic quantum annealers. However, as with any benchmark or test, there are advantages and disadvantages to using spanning trees to investigate some of the properties of a D-Wave quantum annealer. The primary advantages and disadvantages of using USTs and their associated algorithms are discussed below.

Advantages

There are several advantages to using uniform spanning trees to test the performance of quantum annealers. The first is that every qubit, by definition, is included as a vertex of every spanning tree of the graph G of the annealer, and thus every qubit is sampled in the study. A second advantage is that every edge is included in the ensemble of all spanning trees; thus, by generating enough USTs, one can ensure that every coupler is sampled in the study. Third, USTs and the algorithms used to generate them are

scalable, meaning that they can be used to test an annealer of arbitrary size; this has the added benefit of allowing one to easily compare the performance of different size annealers. Finally, although the ground state energies and configurations of spanning tree are difficult problems for quantum annealers to solve, they are relatively easy to calculate by hand or by classical computers. Thus, one can easily check the solutions given by the quantum annealer to a problem which, for it, is nontrivial.

Disadvantages

There are two primary disadvantages to using uniform spanning trees to test quantum annealers. The first is that since there are, by definition, no loops in a spanning tree, there can be no frustration in the problem. This places the question of finding the ground state energies and configurations of USTs in the complexity class P instead of NP. Put another way, this means that the spanning tree problems are simply not hard enough. This leads to the second problem, which is that the primary advantage of quantum computers over classical computers is their ability to solve NP-hard problems, which generally are computationally infeasible for classical computers. Thus, spanning tree problems do not really give insight into this capability of a quantum annealer.

CHAPTER II

D-WAVE TWO EXPERIMENTS

In this chapter, the experiments performed on a D-Wave Two, an adiabatic quantum computer, will be described. The chapter begins with a description of the machine itself, including its operating environment and the architecture of the chip. This is followed by a description of the software used. Next, the particular problems investigated are discussed, and their implementation is described. Finally, the execution of the experiments themselves is explored.

Environment

The D-Wave Two consists of a 10-foot by 7-foot by 10-foot unit which houses a refrigeration system, shielding system, input-output system, and the processor itself. Additionally, there are adjoining cabinets which house control subsystems and servers which provide connectivity for users. The chip has a stated operating temperature of 20 mK and operates at a pressure of 10^{-9} atm [D-Wave 2]. The chip is magnetically shielded to less than 1 nanotesla across the processor in each axis, which is about fifty times less than the Earth's magnetic field [D-Wave 2]. The shielding system also provides protection from vibrations and radiation. The entire system consumes 15.5 kW [D-Wave 2].

Architecture

In this section, the processor itself will be described in more detail. First, the physical implementation of both the qubits and the couplers is briefly described. Then, the architecture and the connectivity between the qubits themselves are also described. Finally, the advantages and disadvantages of the architecture's design are discussed, as well as one proposed solution to a critical disadvantage.

Physical Implementation

Each qubit is made from a loop of metal niobium. Below 9.2 K, which is over two thousand times warmer than the operating temperature of the chip, niobium becomes a superconductor and begins exhibiting quantum mechanical effects. Since niobium has the highest critical temperature of all the elemental superconductors [Peiniger], its use allows the chip to more easily reach and maintain its superconducting and quantum mechanical properties. Additionally, each loop contains a number of Josephson junctions, which allow a current to flow indefinitely across the device without any external voltage or magnetic flux being applied. This phenomenon is called *supercurrent* or the Josepshon effect. Another essential piece of the processor is the coupler. In the D-Wave hardware, there is no physical coupler device: two qubits are "connected" when their induced magnetic fields interact. Specifically, the D-Wave Two processor used in this study consisted of 512 qubits, of which sixteen were not functional, for a total 496 usable qubits. Additionally, one coupler was non-functional.

The control circuitry mentioned earlier is used to apply an external magnetic field across the loop. Since the loop exhibits quantum mechanical properties, only an integer number of flux quanta are allowed to pass through the loop. When a non-integer flux is applied, this causes a current to flow around the loop, which in turn causes the loop to create its own magnetic field. This new field can point either "up" or "down," depending on whether the current is flowing clockwise or counterclockwise. The "up" and "down" states are used to encode the classical "1" and "0" bits. Due to the quantum mechanical effects, the current will be in a superposition of both the clockwise and counterclockwise directions, inducing a magnetic field that is in a superposition of both "up" and "down" states. This is the qubit: a superposition of both "1" and "0." Qubits created in this way, by applying an external magnetic flux to a metal superconducting loop, are called *flux qubits*. The Josephson junctions allow the qubit to maintain its current so that annealing can be performed on the prepared state.

Connectivity

The D-Wave Two utilizes what has been named by D-Wave as a Chimera architecture. Clusters of eight qubits are arranged into rows and columns, and the interactions among the qubits within each cluster are described by a complete bipartite graph $K_{4,4}$. That is, each cluster can be further divided into two groups of four; we can name these two groups the red group and the blue group. Every qubit in the red group interacts with every qubit in the blue group, but there are no interactions among members of the same group. A complete bipartite graph $K_{m,m}$ has m^{2m-2} spanning trees, which for $K_{4,4}$ is 4096.

Furthermore, each red qubit in a cluster is connected with corresponding red qubits in the neighboring rows, and each blue qubit in a cluster is connected with corresponding blue qubits in the neighboring columns. Therefore, each qubit interacts via a coupler with at most six other qubits. Each coupler can be assigned a weight or coupling strength $J_{i,j}$, between qubits i and j. Furthermore, the qubit labeled i can be assigned a magnetic field bias h_i. A representation of the Chimera architecture on thirty-two qubits is shown in Figure 2.2 below.

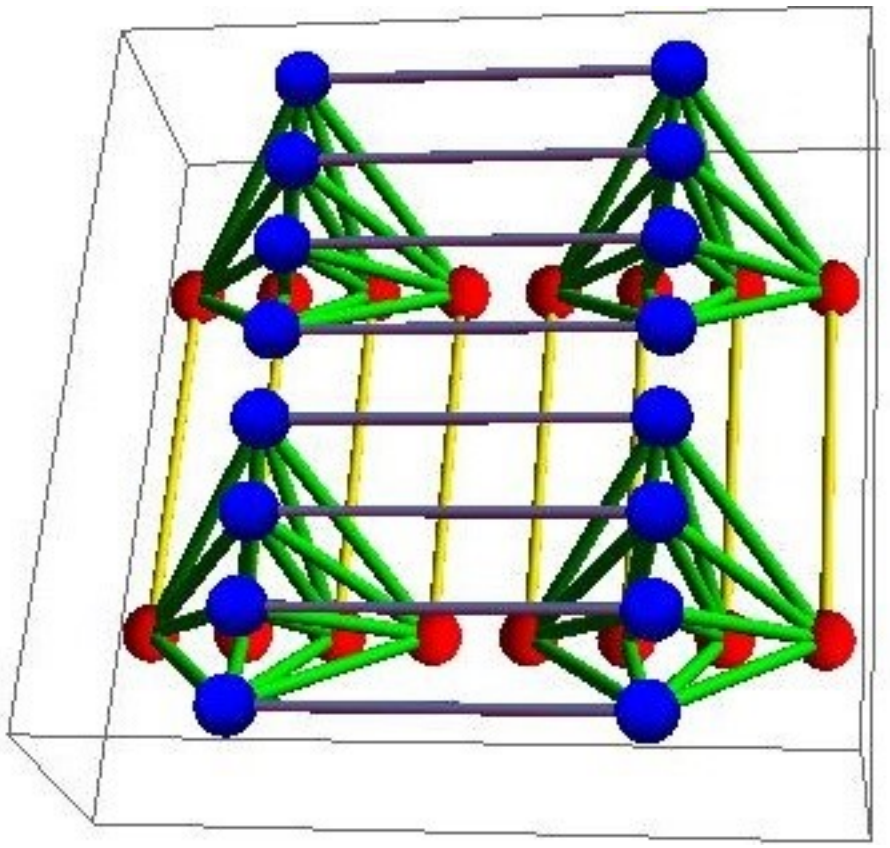

Figure 2.1 Representation of a 2x2 Chimera architecture

Limitations

One advantage of the current architecture is that it has proven to be scalable. D-Wave's first prototype quantum annealer, the 16-qubit Orion Quantum Computer, was introduced on February 13, 2007, at the Computer History Museum in Mountain View, California. Later that year, on November 12, they demonstrated a 28-qubit processor. Next came the company's first commercial quantum computer, the 128-qubit D-Wave One, announced in May of 2011. The company has continued to scale up the number of qubits with each new generation of processors, to 512, to over 1000, to now over 2000.

However, one major limitation of the Chimera architecture is the number of connections. With the current level of connectivity, the D-Wave architecture has no finite-temperature, spin-glass phase transition [Katzgraber]. This limits the class of problems that the D-Wave 2 can feasibly do. Ideally, every qubit would be connected to and interact directly with every other qubit, providing maximum connectivity. Physical limitations, of course, prevent this.

Several suggestions for increasing the number of connections between qubits have been suggested. One such suggestion involves re-designing the chip's architecture so that the resulting graph is a small-world network [Melchert]. Although there are several accepted ways to define the concept exactly, generally speaking, such a network is a mathematical graph for which the average characteristic path length L between two nodes is small; in addition, the nodes tend to cluster together, characterized by a high clustering coefficient C. This means that groups of nodes in the graph will exhibit dense connectivity within the group, but relatively sparse connectivity between groups. Usually, these networks have a higher-than-average number of *hubs*, or nodes with a high number of connections. A small-world Chimera graph has been shown to have a finite-temperature phase transition [Katzgraber and Novotny].

Software and Interfacing

This section will briefly discuss the software used to interact with the D-Wave Two processor. As mentioned earlier in this chapter, part of the D-Wave system consists of servers which regulate access to the machine by external users. Commands and parameters can be submitted in either Python, C, or MATLAB code. The control

subsystem then uses these commands and parameters to prepare the chip accordingly and to execute an annealing cycle or multiple annealing cycles. For this study, all of the software code was written in MATLAB due to more familiarity with that programming language. The API version and libraries used at the time was v1.5.2-beta2.

Description and Implementation of Problems on the D-Wave 2

In this section, the various problems used to test the D-Wave 2 will be presented and described. The first set of problems consisted of implementing a ferromagnet and anti-ferromagnet on the entire chip. These were used primarily as preliminary tests of the machine. Despite their simplicity, a very important result emerged immediately: a bias in the magnetic field strength. The second set of problems consisted of uniform random spanning trees with varying interaction weights. For all problems, the annealing time was 20 μs.

Ferromagnet and Anti-ferromagnet

In the initial experiment, the ferromagnet and the antiferromagnet were studied on the full 8*x*8 Chimera graph (the entire chip with 496 qubits) by applying equal interaction weights to each qubit pair; no magnetic fields were applied. The results are shown in Figure 2.2 below. For the ferromagnet, the interaction weights were varied from -1.5 to 0 in increments of 0.1; for the antiferromagnet, the values were varied from 1 to 0 in increments of 0.1. The results were similar in both cases: for interaction weights greater than 0.5, the D-Wave Two displayed an average success probability of 90% or greater for

finding the ground state energy; the success probability decreased quickly for smaller weights. For the ferromagnet, every ground state configuration found consisted of all spin-ups; no spin-down ground state was observed, suggesting that some bias exists in the finding of the ground state spins. The D-Wave machine used was not a delivered product, and was intended for use internal to or approved by the company; so, the tuning procedure had not been performed to remove this up-down bias.

Next, different random spanning trees were applied to the Chimera graph, again on the 496-qubit chip. Equal, ferromagnetic interaction weights were applied and were again varied from -1.5 to 0 in increments of 0.1. It was observed that the average success probability dropped significantly as the interaction weight decreased, shown in Figure 2.3. No clear pattern was observed among the data points, with the average success probability fluctuating; these fluctuations were most likely due to the up-down bias.

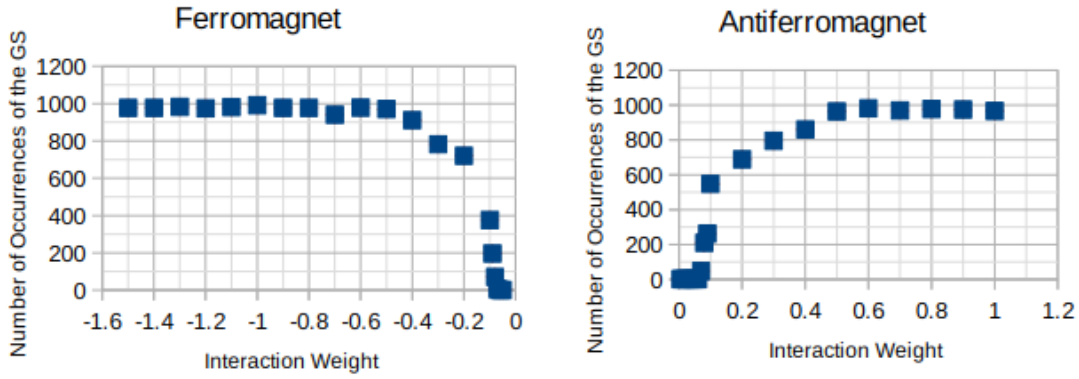

Figure 2.2 Results for the (a) ferromagnet and (b) antiferromagnet on the D-Wave Two

Random Spanning Trees

The second experiment consisted of finding the ground states of random spanning trees. In particular, uniform spanning trees were used and were generated using the Aldous-Broder algorithm discussed in Chapter 1. In the first set of problems, uniform weights were assigned to all the couplers of the spanning trees. In the second set of problems, a weight of -1 or +1 was randomly assigned to the couplers of the spanning trees.

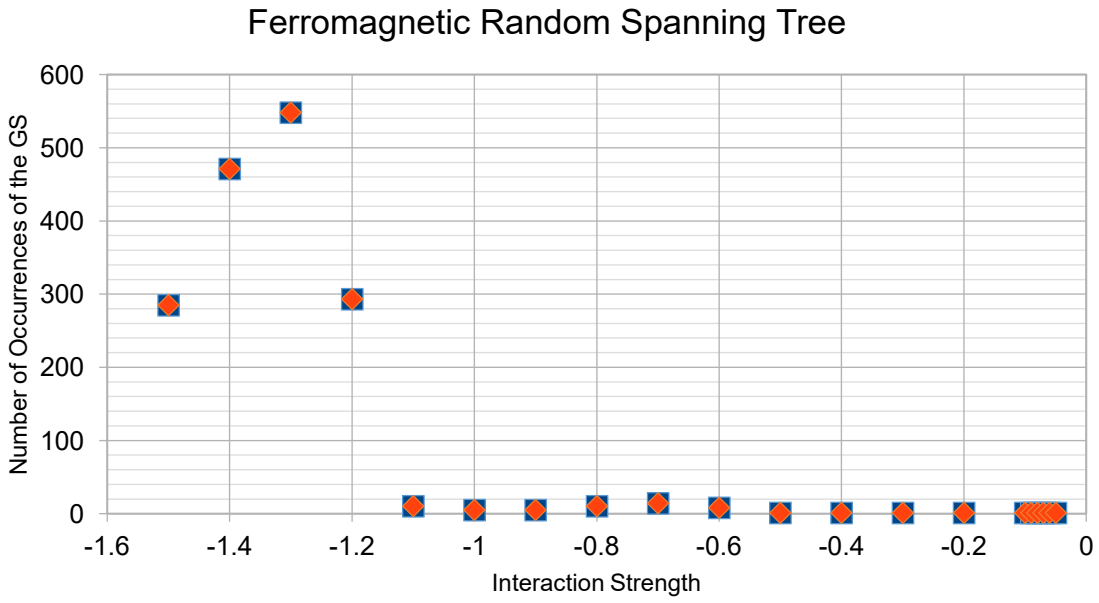

Figure 2.3 Results for the ferromagnetic random spanning trees

Uniform Weights

Random spanning trees were solved with equal, constant interaction weights applied to each qubit pair in the tree. This was done for the 3*x*3, 4*x*4, 5*x*5, 6*x*6, and 7*x*7 subgraphs on the 496-qubit chip. Up to one hundred different spanning trees were solved

on each subgraph, and each tree was submitted up to one hundred times. Each trial resulted in the execution of one thousand anneals, for a maximum of one hundred thousand anneals per tree. The success probabilities were observed to fluctuate widely from trial to trial; these fluctuations were attributed mainly to the up-down bias. The only pattern observed was that the 3*x*3, 5*x*5, and 7*x*7 graphs tended to have a higher average success probability than anticipated by data from the 4*x*4 and 6*x*6 graphs. This is shown in Figure 2.4. Again, only spin-up ground states were observed.

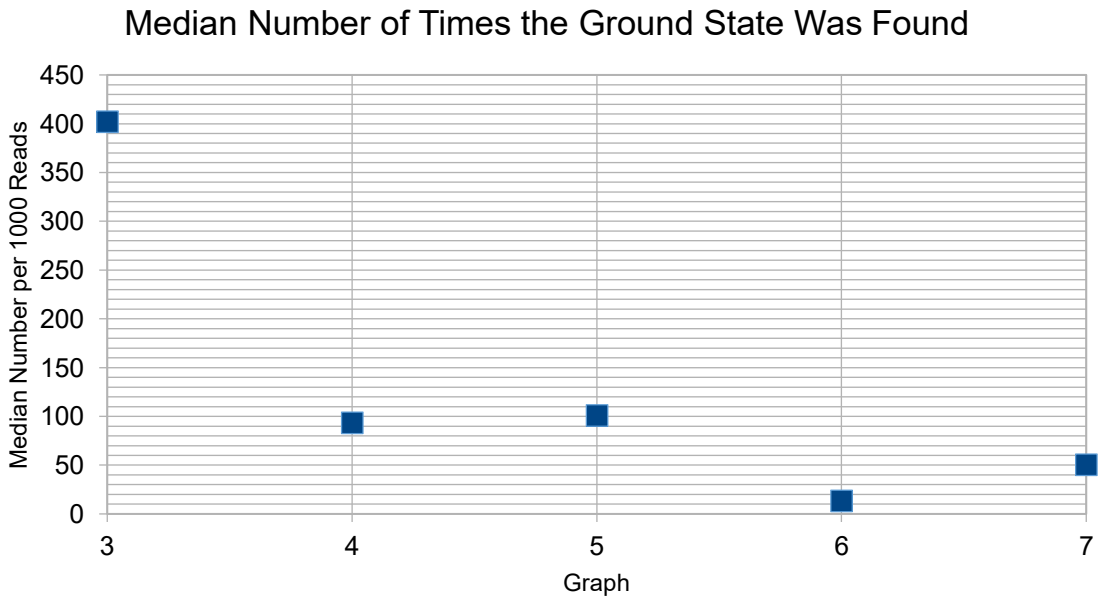

Figure 2.4 Results for spanning trees with uniform weights

Random Weights

In the final experiment, random spanning trees were again solved, but with interaction weights of -1 or +1 randomly assigned to each qubit coupler involved; again,

all h_j were set to zero on the 496-qubit chip. This corresponds to a random gauge transformation. The calculations were performed on the full 8*x*8 graph and on each subgraph, excluding the trivial 1*x*1 subgraph. All subgraphs were restricted to the upper-left corner of the Chimera lattice.

Up to one hundred spanning trees were solved on each subgraph; again, each tree was submitted up to one hundred times, with one thousand anneals performed per submission. The number of occurrences of the ground state energy was recorded. The mean, standard deviation, and median were calculated for each tree and across the trees of each subgraph. Additionally, for each subgraph, the trials were organized into bins according to their success probability. An example of such an analysis is shown in Fig. 2.5. For example, for the 5*x*5 subgraph, which had 197 available qubits, one hundred different spanning trees were generated, and each one was submitted one hundred times, giving a total of ten thousand trials. On each trial, the D-Wave Two solved the tree one thousand times, and the number of times that it found the ground state energy was recorded. These trials were then organized by their success probability into the bins shown in Fig. 2.5.

The success probabilities of the 5*x*5 subgraph demonstrated a type of bimodal distribution, with a larger peak around the lower success probabilities. The success rates were highest for the 2*x*2 subgraph and decreased for each successive subgraph up to the full lattice. The 2*x*2, 3*x*3, and 4*x*4 subgraphs tended to be much more successful on the average, while the 6*x*6, 7*x*7, and 8*x*8 graphs showed much lower success probabilities. Additionally, the 4*x*4 subgraph also demonstrated a bimodal distribution, with the larger peak shifted to the right toward higher success probabilities.

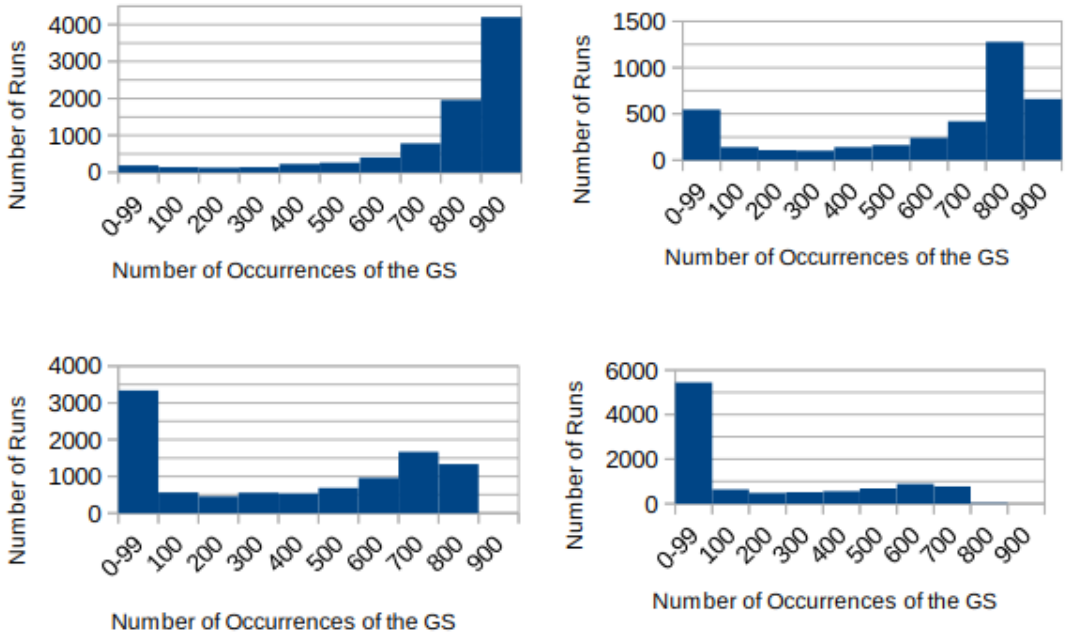

Figure 2.5 Results for the (a) 3x3, (b) 4x4, (c) 5x5, and (d) 6x6 subgraphs.

NOTE: Each bin represents an interval of one hundred. Except for the first bin, only the first number of the interval is shown. The last bin ranges from 900-1000.

In a final analysis, we looked at whether or not the ground state energy was found on each trial. If the ground state was found at least once out of the one thousand annealing trajectories, this was counted as a success. The results are presented in Fig. 2.6. Each point represents the success probability calculated over all trials on that subgraph. For the 2*x*2 and 3*x*3 subgraphs, the D-Wave Two found the ground state energy 100% of the time; this means that the ground state energy was found at least once on each trial. The success probability decreases for each larger subgraph; the success probability for the full 8*x*8 graph was 33%. The success of an adiabatic quantum annealer, such as the D-Wave Two, is a function of the annealing time and the temperature. For infinite annealing

time and operation at absolute zero temperature, the ground state is guaranteed, theoretically, to be found.

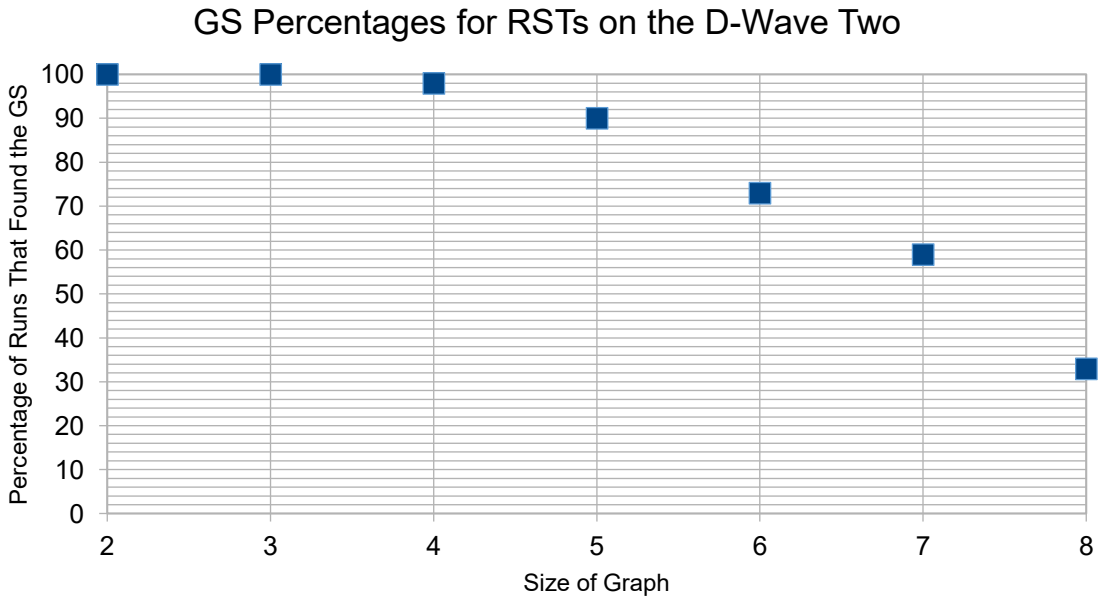

Figure 2.6 Success probabilities for the D-Wave Two

CHAPTER III

D-WAVE 2X EXPERIMENTS

In this chapter, the experiments performed on a D-Wave 2X, also an adiabatic quantum computer, will be described. The chapter begins with a description of the machine itself, including its operating environment and the architecture of the chip. This is followed by a description of the software used. Next, the particular problems investigated are discussed, and their implementation is described. Finally, the execution of the experiments themselves is explored.

Changes from the D-Wave 2

In this section, the operating environment of the chip, the architecture of the chip, and the software used will be each be described and discussed in turn. In particular, attention will be paid to the differences between and changes made from the D-Wave Two, which was described and discussed in the previous chapter.

Environment

The most significant change to the operating environment from the D-Wave Two to the D-Wave 2X is the operating temperature. The D-Wave 2X has a stated operating temperature of 15 mK [D-Wave 2X], which is 5 mK cooler than the D-Wave Two. As

mentioned in Chapter 2, it has been shown that cooler operating temperatures are needed as the size of the Chimera lattice grows in order to maintain performance [Albash]. These extreme temperatures are necessary to prevent the phenomenon of decoherence discussed in Chapter 1.

The stated operating pressure has remained the same, at 10^{-9} atm [D-Wave 2X]. The magnetic shielding system has also not changed: the chip is still magnetically shielded from its surroundings to less than one nanotesla across each axis, fifty times less than the Earth's magnetic field [D-Wave 2X]. Finally, the power consumption of the system has increased from 15.5 kW/h to 25 kW/h; this extra power is mostly used by the cooling system to achieve the lower operating temperature [D-Wave 2X].

Architecture

The D-Wave 2X still utilizes the Chimera architecture of the D-Wave Two previously described in Chapter 2. There are two important differences between the old D-Wave Two chip and the new D-Wave 2X chip. The first is the size of the Chimera lattice: the newer processor is comprised of a 12x12 grid of cells of eight qubits each, compared with an 8x8 grid for the D-Wave Two. The connections between the qubits in each cell again form a complete, bipartite graph $K_{4,4}$. Therefore, there are still 4096 possible spanning trees for each cell, as shown in Chapter 2.

The second important difference is the amount of disorder present in the lattice. The D-Wave 2X used in this study consisted of 1152 qubits, of which 56 were non-functional, for a total of 1096 usable qubits. Thus, approximately 5% of the qubits were

not working, compared with about 3% for the older chip. All 3060 couplers were functional on the D-Wave 2X processor.

Software

As with the D-Wave Two, external access to the D-Wave 2X is controlled by a collection of servers. Problems can again be programmed in either Python, C, or MATLAB and submitted along with other parameters using the latest client libraries. These parameters are then converted into quantum machine language (QMI), and the control subsystem prepares the processor accordingly. As with the previous part of this study conducted on the D-Wave Two, all of the problems and parameters were programmed in MATLAB. The API version and libraries used at the time was v2.1.1.

Description and Implementation of Problems on the D-Wave 2X

In this section, the various problems used to test the D-Wave 2X will be presented and described. The first problem consisted of implementing a ferromagnet on the entire graph of the chip and varying the bias field uniformly across the chip. As before, this was used primarily as a preliminary test of the machine. For the second problem, the study of random spanning trees was continued.

Ferromagnet

In the first test performed on the D-Wave 2X, an interaction strength of $J_{i,j} = +1$ was assigned to every working coupler, resulting in a ferromagnet. A uniform magnetic

field bias h_j was then assigned to every working qubit. The four values used for h_j were 0, 0.06, 0.07, and 0.08. In each case, one thousand anneals were performed and the ground state energies and configurations were found and recorded. In every case, the ground state, which has an energy of E = -3060, was found.

For $h_j = 0$, the probabilities of finding the "up" state, where s_j = -1, and the "down" state, where s_j = +1, should be equal. This means that out of the one thousand anneals performed when the external field was zero, about five hundred results should be in the "up" state and about five hundred should be in the down state. However, this was not the case: for all one thousand anneals, the "up" ground state was found. Based on experience with the previous ferromagnet experiments on the D-Wave Two, this result was not unexpected.

When an external field strength of $h_j = 0.7$ was applied, both ground states were found, although the results were still biased toward the "up" state. When $h_j = 0.8$, only the "down" state was found. This result is in agreement with the stated 3% analog control error for the D-Wave 2X processor. The complete results are shown in Table 3.1 below.

Table 3.1 Results for the ferromagnet experiment on the D-Wave 2X

h_j	**0.00**	**0.06**	**0.07**	**0.08**
$s_j = -1$	1000	1000	823	0
$s_j = +1$	0	0	177	1000

Continuation of RSTs

In the second experiment, uniform spanning trees were again used to test the performance of the D-Wave 2X processor. However, due to experience with the D-Wave Two, no experiments were performed with uniform weights assigned to the couplers of the spanning trees. Instead, only experiments with randomly-assigned interaction weights of -1 or +1 and external field $h_j = 0$ were performed. This again corresponds to a random gauge transformation on the section of the Chimera lattice involved.

Calculations were only performed on the 7x7, 8x8, 9x9, 10x10, 11x11, and 12x12 lattices. Once again, all of the subgraphs were anchored at the upper-left corner of the lattice. Spanning tree problems were not solved on smaller sub-lattices due to the 7x7 lattice having a consistent success probability of around one hundred percent. Based on the trends found on both the D-Wave Two and the D-Wave 2X, smaller sub-lattices were assumed to also have consistent success probabilities of nearly one hundred percent.

One hundred spanning trees were solved on each graph. Each tree was submitted one hundred times, with one thousand anneals performed on each submission, for a total of one hundred thousand anneals per tree and one million anneals per graph. The number of occurrences of the ground state energy was recorded. The mean, standard deviation, and median were calculated for each tree and across the trees of each subgraph. Additionally, for each subgraph, the trials were organized into bins according to their success probability. An example of such an analysis is shown in Figure 3.1.

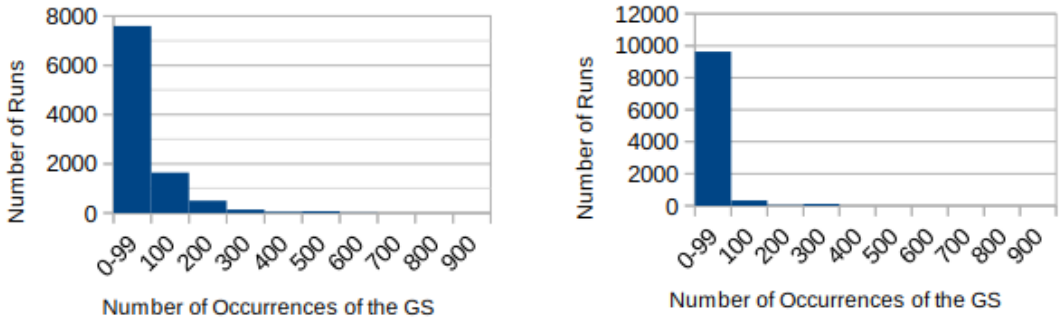

Figure 3.1 Results for the (a) 7x7 and (b) 8x8 subgraphs on the D-Wave 2X

In a final analysis, it was investigated whether or not the ground state energy was found on each trial. If the ground state was found at least once out of the one thousand annealing trajectories, this was counted as a success. The results are presented in Figure 3.2. Each point represents the success probability calculated over all trials on that subgraph. For the 7x7 subgraph, the D-Wave 2X found the ground state energy about 99% of the time; this means that the ground state energy was found at least once on all but one of the trials. The success probability decreases for each larger subgraph; the success probability for the full 12x12 graph was about 4%. The success of an adiabatic quantum annealer, such as the D-Wave 2X, is a function of the annealing time and the temperature. For infinite annealing time and operation at absolute zero temperature, the ground state is guaranteed, theoretically, to be found.

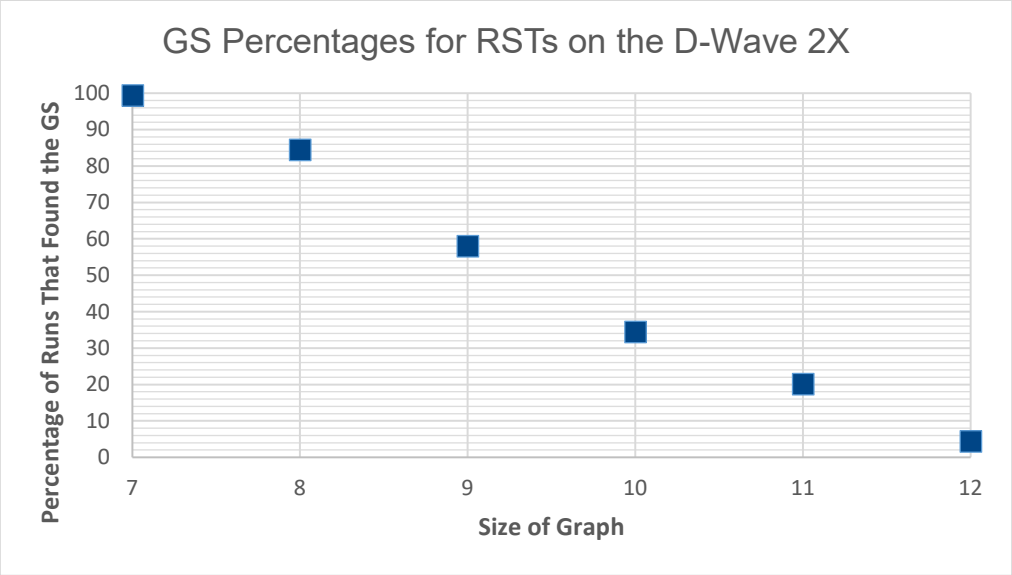

Figure 3.2 Success probabilities for the D-Wave 2X

CHAPTER IV

RESULTS AND DISCUSSION

The results from the experiments described in the previous two chapters will now be analyzed, discussed, and compared. The chapter begins by discussing the results of the experiments performed on the D-Wave 2 and their implications. An analysis of the results from the D-Wave 2X experiments then follows. The chapter ends by comparing the results and discussing the implications of the data as a whole.

D-Wave 2

It can be concluded from the first study of the ferromagnet and the antiferromagnet that the probability of finding the ground state correlates with the strength of the interaction, as expected; higher success rates are seen for stronger interaction weights and decrease at lower interaction weights. In the second study of ferromagnetic spanning trees, it was determined that the up-down bias was influencing our results to an unacceptable degree, prompting a third study using spanning trees with randomly assigned interaction weights of -1 or +1. This was found to sufficiently mitigate the bias. The final results show that the probability of finding the ground state energy decreases on average with the number of qubits for fixed annealing time and temperature.

D-Wave 2X

In the first study, with the ferromagnetic bonds, it was determined that the field bias present in the first chip was also present in the second chip. However, analysis showed that the bias was within the stated 3% analog control error for the D-Wave 2X. In the second study, the investigation of uniform random spanning trees was continued. A trend similar to the one found for the D-Wave 2 was revealed: the probability of finding the ground state decreased with the size of the graph. However, it was found that the D-Wave 2X could, in general, handle larger graphs better than the D-Wave 2.

Comparison

Arguably the most important result arising from this study will now be presented and discussed; that is, the increase in the probability that the D-Wave 2X will find the ground state of a uniform random spanning tree compared with the D-Wave 2. As mentioned before, this increase is mainly attributed to the lower operating temperature of the D-Wave 2X [Albash].

The first piece of evidence for this increase can be found in Figure 4.1 below. It shows the percentage of submissions that found the ground state zero to one hundred times out of one thousand anneals on the 7x7 subgraph for both the D-Wave 2 and the D-Wave 2X. As can be seen for the D-Wave 2, approximately forty percent of all submissions never found the ground state. However, for the D-Wave 2X this percentage drops dramatically to only 0.56 percent. Most of the submissions found the result only once, accounting for about 8.89 percent of all ten thousand submissions. Obviously, there

is a vast improvement between the two chips, with most of the results now lying in the integral over all “success” states instead of at zero.

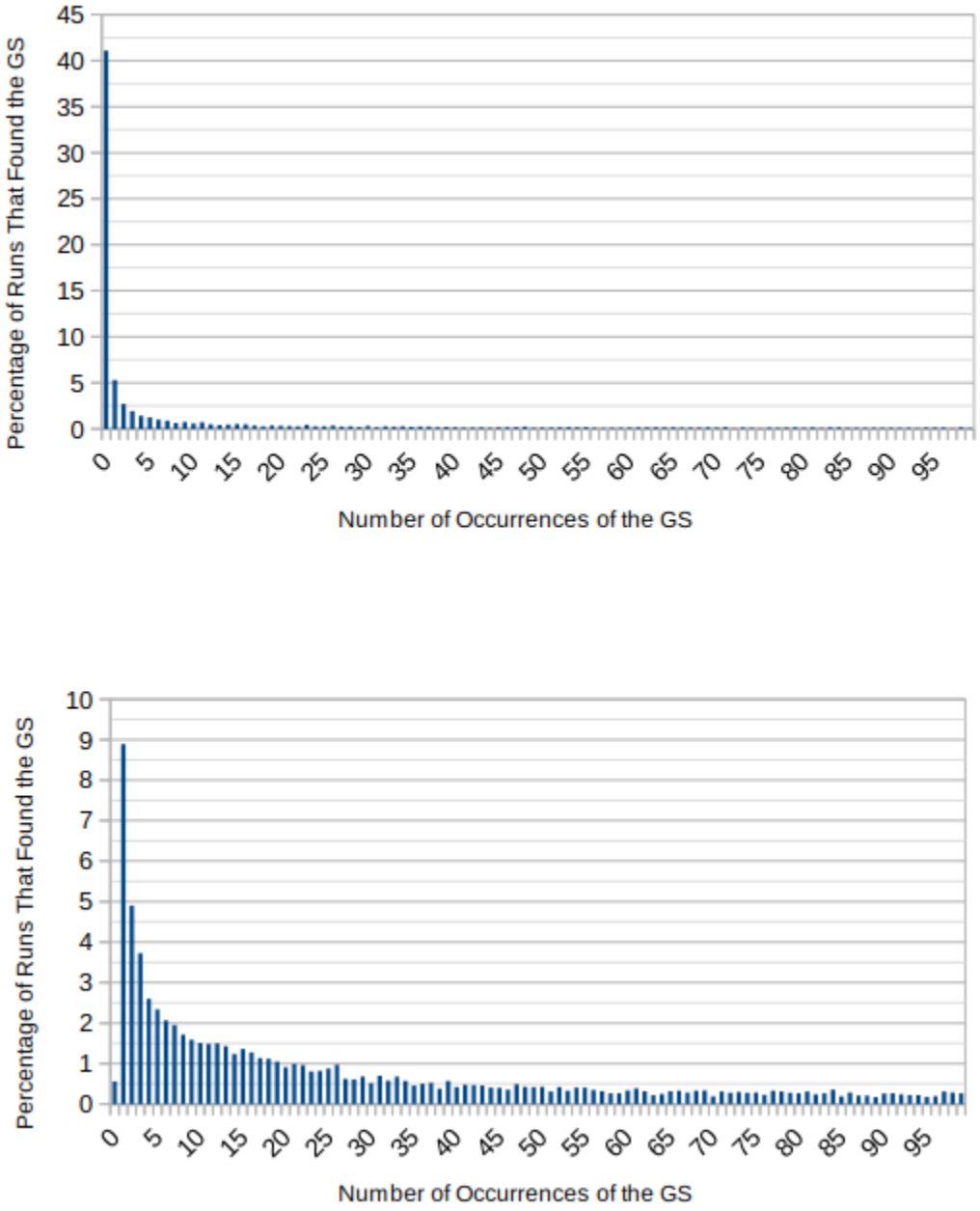

Figure 4.1 Comparison of the 7x7 Subgraphs for the (a) D-Wave Two and (b) D-Wave 2X

A second piece of evidence for the improvement between the two chips can be found by comparing the 8x8 subgraphs. The results for both the D-Wave 2 and the D-Wave 2X are found in Figure 4.2 below. A trend emerges that is similar to the one observed when comparing the 7x7 subgraphs. For the D-Wave 2, about sixty-seven percent of all ten thousand submissions never found the ground state. For the D-Wave 2X, this percentage drops to about fifteen percent. Although this still represents a plurality of the results, it no longer dominates. The remainder of the results are now spread out among the integral of "success" measurements. The next highest result is for one ground state found out of one thousand anneals and represents approximately fourteen percent.

In a final analysis, we compare the percent of submissions that found the ground state with the percent that did not. The result of this analysis is presented in Figure 4.3. When compared with Figures 4.1 and 4.2, the result is comparing the number of submissions that never found the ground state with the rest of the integral over all "success" instances where the ground state was found at least once. For instance, the data point for the 8x8 subgraph on the D-Wave 2X shows that about eighty-five percent of all submissions found the ground state at least once out of one thousand anneals. This percentage agrees with the data shown in Figure 4.2 above, which shows that about fifteen percent of the submissions never found the ground state even once.

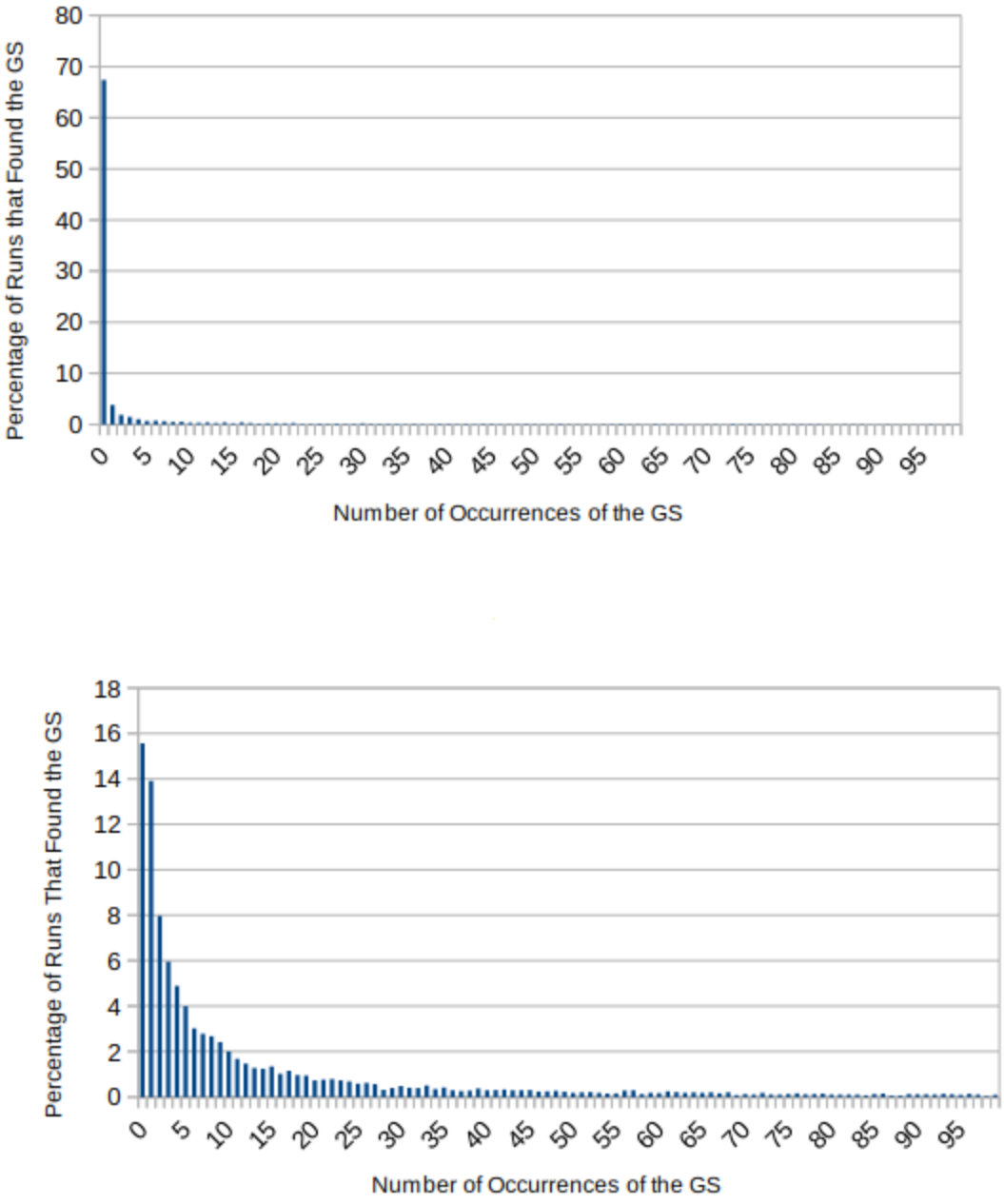

Figure 4.2 Comparison of the 8x8 Subgraphs for the (a) D-Wave Two and (b) D-Wave 2X

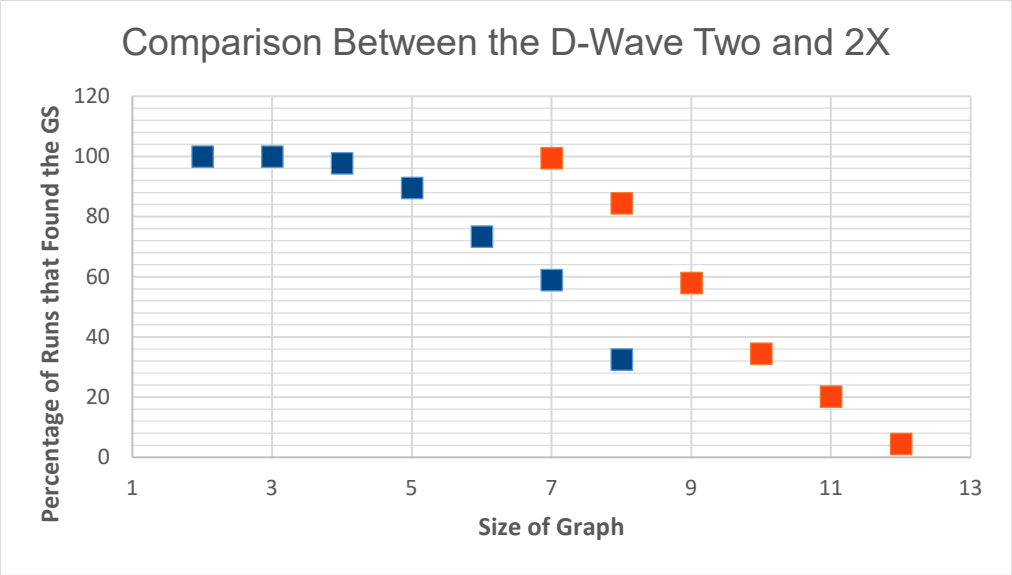

Figure 4.3 Success Probabilities for both the D-Wave 2 and the D-Wave 2X

Since data was not collected on the D-Wave 2X for any subgraph smaller than the 7x7 subgraph, the only overlap between the two data sets is the 7x7 and 8x8 subgraphs. Data was not collected for anything smaller than the 7x7 since the trend in Figure 4.3 indicates that the ground state is found close to one hundred percent of the time for everysubgraph smaller than the 7x7. However, as has been presented in this chapter, these sets of data provide ample evidence of improvement from the older to the newer generation of chip.

CHAPTER V

RECOMMENDATIONS FOR FUTURE WORK

Two possibilities for future work that build on the findings presented in this thesis will now be presented and briefly discussed. The first of these addresses the biases discovered early on in this study and discussed previously. It is suggested that these biases be carefully measured and corrected. The second possibility concerns the impact that a tree's structure might have on the ability of the D-Wave computers to solve for the ground states of the spanning trees.

Correcting for Biases

As discussed earlier, it was discovered in preliminary tests that portions of the chip suffer from a bias in the magnetic field. As one would expect, these biases have an adverse impact on the performance of the quantum annealers and can also distort the results. An obvious avenue for further work arises from this problem: carefully measure and correct for these biases. Not surprisingly, others have also noted this problem and have already done work to address it [Perdomo-Ortiz]. The experiments and tests presented in this thesis could then be redone using the corrected biases. It is expected that correcting for these biases will positively impact performance, increasing the success probabilities of the D-wave chips.

Post-processing

Another avenue for investigation involves applying classical error-correction methods to the solutions provided by the D-Wave quantum annealer. This technique for improving the results is called post-processing. Although there exist quantum error-correcting codes for the Chimera architecture of the D-Wave processor [Pudenz], as well as quantum stabilizer codes [Young], these require many qubits to implement, in addition to those needed to encode the problem itself. For a chip with limited qubits, the result is fewer qubits on which to encode a problem, leading to a decrease in the size of problems that the processor can encode and solve. An example of such a method is shown in Figure 5.1 [Pudenz]. Applying classical error-correcting techniques in post-processing is currently the more feasible alternative.

Several classical methods have been proposed [King]. One such method is to apply the idea of majority logic decoding, or "majority vote," to chains of qubits in the solution. For instance, a branch of a spanning tree would constitute a string or chain of qubits. If, in a solution, one qubit in this branch was "up" while the rest were "down," then by majority vote that qubit would be changed to "down," thus correcting the solution and its associated energy. Ties would be broken randomly.

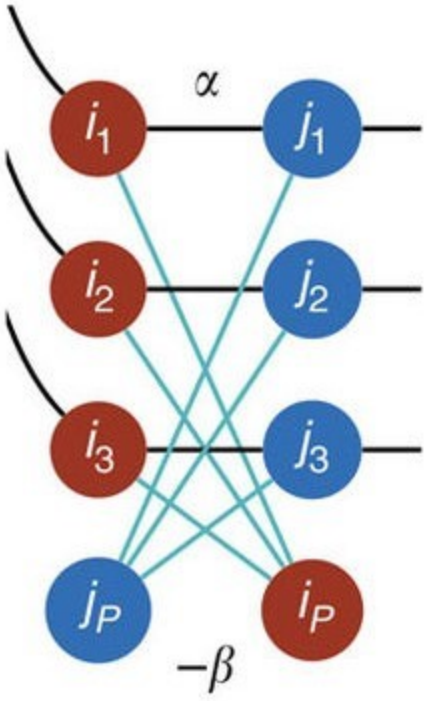

Figure 5.1 A physical implementation of a quantum error-correction code.

NOTE: The qubits labeled 1, 2, and 3 are the qubits that encode the problem, while the qubits labeled *P* are the qubits used for error-correction.

Another proposed method is termed *greedy descent*. The basic idea underpinning this technique is to randomly flip the qubits in a given solution in an effort to reach a local minimum solution. If flipping a qubit reduces the energy of the solution, we leave it flipped, but if it does not, we unflip it. Repeating this process enough times increases the probability of the solution reaching a local minimum if it is not already at one.

Furthermore, any classical heuristic method can be used to compare and refine solutions, as well as initial problems. Some popular heuristic methods already being used include simulated annealing, tabu search, parallel tempering, and Boltzmann sampling.

Analyzing Size of Trees

A final suggestion for building upon this work would be to understand the impact, if any, that a tree's structure has on the success probabilities. There are two principal aspects of a tree's structure that could be investigated: the number of branches and the length of the branches. Each of these cases is discussed in further detail below. It is hypothesized that both are inversely correlated with performance.

Number of Branches

The first aspect of a tree's structure is its number of branches. In order to count the number of branches, several ideas need to be introduced. The first is that of a *root*, which is a vertex that can be thought of as the "starting point" of a given tree; any vertex can be designated as the root. A tree in which a vertex has been chosen as the root is called a *rooted tree*. Two natural orientations can be assigned to a rooted tree, either *away from* the root or *toward* the root; such a tree is called a *directed rooted tree*. A second idea that is needed is that of a parent vertex. The *parent* of any given vertex V is defined to be that vertex which is connected to V and lies on the path directed toward the root. Since there are no loops in a spanning tree, it follows that every vertex has a unique parent. A *child* of a given vertex V is a vertex of which V is the parent. The final idea needed is that of the degree of a vertex. The *degree* of a vertex V is defined simply to be the number of edges connected to that vertex and is denoted $\deg(V)$. Hence, every vertex in a tree has a degree of at least one. Vertices of degree one are called *external*, *outer*, *terminal*, or *leaf* vertices, while those of degree two or more are called *internal*, *inner*, or *branch* vertices.

With these ideas, one can define exactly what a branch is and devise an algorithm to count them. For example, one could define *branch* to mean a child of any vertex of degree three or more, corresponding to the intuitive idea of a branch. A natural way to count the number of branches B defined this way would be to add up the degrees of all the vertices $V_1, V_2, \ldots, V_N$ of degree three or greater and then to subtract off the number N of these vertices:

$$B = \deg(V_1) + \deg(V_2) + \ldots + \deg(V_N) - N, \quad \deg(V_i) \geq 3. \tag{5.1}$$

Thus, we would say that a given graph G branches N times and has B branches. Since N and B say something about the structure of a graph, we could compare the success probabilities of trees with different N or B or both and try to determine the effect that these particular aspects of a tree's structure has on the quantum computer's ability to find the ground state energies and configurations. Finally, there already exist many tree-search algorithms that could be used to traverse a tree starting at an arbitrary root, identifying the vertices of degree three or more in the process.

Length of Branches

The second aspect a tree's structure is the length of its branches. There already exist two ideas of length in the literature. The first is the *height* of a given vertex V, defined to be the length of the longest path from V to a terminal vertex. The second is called the *depth* of a vertex, which is the length of the path connecting that vertex to the root. In both cases, the *length of a path* is simply the number of edges or vertices along that path. Either of these conceptions could be used to define the length of the branches as defined in the section above. For example, an intuitive way to define the length of a

particular branch would be to equate it with the length of the path from the root to the corresponding terminal vertex. In that case, the length of the branch is simply the depth of the corresponding terminal vertex. With working definitions in hand, one could study and quantify the impact a tree with a few, long branches has on the performance of a quantum annealer and compare that with the impact that a tree with many, short branches has. It is suspected that increasing either the number of branches or the length of the branches will negatively impact performance.

BIBLIOGRAPHY

Aaronson, *et al*. “Petting Zoo.” *Complexity Zoo*, University of Waterloo, 1 Aug. 2015, https://complexityzoo.uwaterloo.ca/Petting_Zoo.

Albash and Lidar. “Adiabatic quantum computation.” *Reviews of Modern Physics*, vol. 90, no. 1, American Physical Society, 29 Jan. 2018, 015002, doi:10.1103/RevModPhys.90.015002.

Albash, *et al*. “Temperature Scaling Law for Quantum Annealing Optimizers.” *Physical Review Letters*, vol. 119, no. 11, American Physical Society, 15 Sept. 2017, 110502, doi:10.1103/PhysRevLett.119.110502.

Aldous. “The random walk construction of uniform spanning trees and uniform labeled trees.” *SIAM Journal on Discrete Mathematics*, vol. 3 no. 4, Society for Industrial and Applied Mathematics, Nov. 1990, pp. 450-465, doi:10.1137/0403039.

Bennett and Wiesner. "Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states.” *Physical Review Letters*, vol. 69, no. 20, American Physical Society, 16 Nov. 1992, pp. 2881-2884, doi:10.1103/PhysRevLett.69.2881.

Bookatz. “QMA-complete problems.” *Quantum Information & Computation*, vol. 14, no. 5-6, Rinton Press, Apr. 2014, pp. 361-383, http://www.rintonpress.com/journals/qiconline.html.

Born and Fock. "Beweis des Adiabatensatzes". *Zeitschrift für Physik*, vol. 51, no. 3–4, Springer, Mar. 1928, pp. 165–180, doi:10.1007/BF01343193.

Broder. “Generating random spanning trees.” *30th Annual Symposium on Foundations of Computer Science*, Oct. 30 – Nov. 1, 1989, Research Triangle Park, NC, Institute for Electrical and Electronics Engineers, 1989, pp. 442-447, doi:10.1109/SFCS.1989.63516.

Cobham. “The intrinsic computational difficulty of functions.” *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress*, edited by Yehoshua Bar-Hillel, North-Holland Publishing Company, 1965, pp. 24-30, doi:10.2307/2270886

Das and Chakrabarti. “*Colloquium*: Quantum annealing and analog quantum computation.” *Reviews of Modern Physics*, vol. 80, no. 3, American Physical Society, 5 Sept. 2008, pp. 1061-1081, doi:10.1103/RevModPhys.80.1061

Deutsch. "Quantum theory, the Church-Turing principle and the universal quantum computer.” *Proceedings of the Royal Society A*, vol. 400, no. 1818, Royal Society Publishing, 8 Jul. 1985, pp. 97–117, doi:10.1098/rspa.1985.0070.

Dieks. "Communication by EPR devices.” *Physics Letters A*, vol. 92, no. 6, Elsevier, 22 Nov. 1982, pp. 271-272, doi:10.1016/0375-9601(82)90084-6.

D-Wave. “Press Releases.” *D-Wave Systems*, https://www.dwavesys.com/news/press-releases.

D-Wave 2. “The D-Wave Two Quantum Computer: Technology Overview.” D-Wave Systems Inc., May 2013, https://www.dwavesys.com/.

D-Wave 2X. “The D-Wave 2X Quantum Computer: Technology Overview.” D-Wave Systems Inc., Sept. 2015, https://www.dwavesys.com/sites/default/files/D-Wave%202X%20Tech%20Collateral_0915F.pdf.

Ekert. “Quantum cryptography based on Bell’s theorem.” *Physical Review Letters*, vol. 67, no. 6, American Physical Society, 5 Aug. 1991, pp.661-663, doi:10.1103/PhysRevLett.67.661.

Feynman. “Simulating Physics with Computers.” *International Journal of Theoretical Physics*, vol. 21, no. 6-7, Springer, Jun. 1982, pp. 467-488, doi:10.1007/BF02650179.

Finilla, *et al*. "Quantum annealing: a new method for minimizing multidimensional functions." *Chemical Physics Letter*, vol. 219, no. 5-6, Elsevier, 18 Mar. 1994, pp. 343-344, doi:10.1016/0009-2614(94)00117-0

Holevo. "Bounds for the Quantity of Information Transmitted by a Quantum Communication Channel.” *Problemy Peredachi Informatsii*, vol. 9, no. 3, Russian Academy of Sciences, Jul. 1973, pp. 3–11, http://mi.mathnet.ru/eng/ppi903.

Ingarden. "Quantum information theory." *Reports on Mathematical Physics*, vol. 10, no. 1, Elsevier, Aug. 1976, pp. 43–72, doi:10.1016/0034-4877(76)90005-7.

Kadowaki and Nishimori. "Quantum annealing in the transverse Ising model." *Physical Review E*, vol. 58, no. 5, American Physical Society, 1 Nov. 1998, pp. 5355-5363, doi:10.1103/PhysRevE.58.5355.

Katzgraber, *et al*. “Glassy Chimeras Could Be Blind to Quantum Speedup: Designing Better Benchmarks for Quantum Annealing Machines.” *Physical Review X*, vol. 4, no. 2, American Physical Society, 10 Apr. 2014, 021008, doi:10.1103/PhysRevX.4.021008.

Katzgraber and Novotny. [unpublished].

King and McGeoch. “Algorithm engineering for a quantum annealing platform.” *arXiv*, Cornell University Library, 18 Oct. 2014, 2628, arXiv:1410.2628v2 [cs.DS].

Kirchhoff. “Über die Auflösung der Gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer Ströme geführt wird." *Annalen der Physik und Chemie*, vol. 148, no. 12, Wiley-VCH, 1847, pp. 497-508, doi:10.1002/andp.18471481202.

Melchert, *et al*. “Site- and bond-percolation thresholds in $K_{n,n}$-based lattices: Vulnerability of quantum annealers to random qubit and coupler failures on chimera topologies.” *Physical Review E*, vol. 93, no. 4, American Physical Society, 25 Apr. 2016, 042128, doi:10.1103/PhysRevE.93.042128.

Mizel, *et al*. “Simple Proof of Equivalence between Adiabatic Quantum Computation and the Circuit Model.” *Physical Review Letters*, vol. 99, no. 7, American Physical Society, 16 Aug. 2007, 070502, doi:10.1103/PhysRevLett.99.070502.

Nielsen and Chuang. *Quantum Computation and Quantum Information (10th Anniversary Edition)*. Cambridge University Press, 2000.

Novotny, *et al*. “Quantum decoherence and thermalization at finite temperature withing the canonical-thermal-state ensemble.” *Physical Review A*, vol. 93, no. 3, American Physical Society, 7 Mar. 2016, 032110, doi:10.1103/PhysRevA.93.032110.

Peiniger and Piel. “A Superconducting Nb3Sn Coated Multicell Accelerating Cavity.” *IEEE Transactions on Nuclear Science*, vol. 32, no. 5, Institute of Electrical and Electronics Engineers, Oct. 1985, pp. 3610-3612, doi:10.1109/TNS.1985.4334443.

Perdomo-Ortiz, *et al*. “Determination and correction of persistent biases in quantum annealers.” *Scientific Reports*, vol. 6, Nature Publishing Group, 19 Jan. 2016, 18628, doi:10.1038/srep18628.

Poplavskii. "Thermodynamical models of information processing.” *Uspekhi Fizicheskikh Nauk*, vol. 115, no. 3, Russian Academy of Sciences, 1975, pp. 465–501, doi:10.3367/UFNr.0115.197503d.0465.

Pudenz, *et al*. “Error-corrected quantum annealing with hundreds of qubits.” *Nature Communications*, vol. 5, Nature Publishing Group, 6 Feb. 2014, 3243, doi:10.1038/ncomms4243.

Savitch. "Relationships between nondeterministic and deterministic tape complexities." *Journal of Computer and System Sciences*, vol. 4, no. 2, Elsevier, Apr. 1970, pp. 177–192, doi:10.1016/S0022-0000(70)80006-X.

Shore. "Scheme for reducing decoherence in quantum computer memory." *Physical Review A*, vol. 52, no. 4, American Physical Society, 1 Oct. 1995, pp. R2493-R2496, doi:10.1103/PhysRevA.52.R2493.

Steane. "Error Correcting Codes in Quantum Theory." *Physical Review Letters*, vol. 77, no. 5, American Physical Society, 29 Jul. 1996, pp. 793-797, doi:10.1103/PhysRevLett.77.793.

Wilson. "Generating Random Spanning Trees More Quickly than the Cover Time." *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*, May 22-24, 1996, Philadelphia, PA, Association for Computing Machinery, 1996, pp. 296-303, doi:10.1145/237814.237880.

Wootters & Zurek. "A single quantum cannot be cloned." *Nature*, vol. 299, no. 5886, Nature Publishing Group, 28 Oct. 1982, pp. 802–803, doi:10.1038/299802a0.

Yao. "Quantum Circuit Complexity." *Proceedings of 1993 IEEE 34th Annual Symposium on Foundations of Computer Science*, 3-5 Nov., Palo Alto, CA, Institute of Electrical and Electronics Engineers, 1993, doi:10.1109/SFCS.1993.366852.

Young, *et al*. "Adiabatic quantum optimization with the wrong Hamiltonian." *Physical Review A*, vol. 88, no. 6, American Physical Society, 11 Dec. 2013, 062314, doi:10.1103/PhysRevA.88.062314.

Vinci, *et al*. "Quantum annealing correction with minor embedding." *Physical Review A*, vol. 92, no. 4, American Physical Society, 9 Oct. 2015, 042310, doi:10.1103/PhysRevA.92.042310.

APPENDIX A

COMPUTER PROGRAM

```
%chimera.m
%Defines a graph on an n-qubit chip and assigns Ising parameters
%By: Spencer Hall
%Mississippi State University and Forschungzentrum Julich

clear all
clc

display('chimera.m')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%List of user-defined variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%conntype (Local or remote connection? l/r) %checked
%url (SAPI url)
%token (API token)
%solvername (name of solver to use) %checked
%qsage (Create QSage solver? y/n) %checked

%x (number of columns on chip)
%y (number of rows on chip)
%qnw (zero-based indices of qubits not working (vector))
%cnw (zero-based indices of couplers not working (two-column vector))
%x0 (column-coordinate of upper left corner of subgraph)
%y0 (row-coordinate of upper left corner of subgraph)
%ulc (qubit in upper left corner of subgraph)
%a (column-dimension of the subgraph)
%b (row-dimension of the subgraph)
%NOTE: {x0,y0} and {ulc,a,b} are mutually exclusive

%answer_mode
%max_answers
%num_reads
%annealing_time
%programming_thermalization
%readout_thermalization
%auto_scale

%I (interaction weights for couplers) %checked
%lI (lower bound for random weights) %checked
%uI (upper bound for random weights) %checked
```

```
%H (external magnetic field biases for qubits) %checked
%lH (lower bound for random biases) %checked
%uH (upper bound for random biases) %checked

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Test the library and MATLAB path
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%test to see if the library and MATLAB path are set up correctly
display(['SAPI version: ', sapiVersion])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Connect to a solver and display its properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%conntype = input('Local or remote connection (l/r)? ', 's');
conntype = 'r';
%force an acceptable input for the variable conntype
while and(strcmp(conntype,'l')~=1, strcmp(conntype,'r')~=1)
   conntype = input(' You must type l or r: ');
end

%create a SAPI connection handle
if conntype == 'l'
   %create a local SAPI connection handle
   conn = sapiLocalConnection;
else
   %input and test the url
   %url = input('SAPI url: ', 's');
   url = 'https://qfe.nas.nasa.gov/sapi/';
   %urlread(url);

   %create a remote SAPI connection handle
         %token = input('API token: ', 's');
   token = 'USRA-126a17b3b2c250ad99bb506815bdf94ce0d91247';
   conn = sapiRemoteConnection(url, token);
end

%list the SAPI connection's available solvers
solverNames = sapiListSolvers(conn);
```

```
display('Available solvers: ')
disp(solverNames)

%solvername = input('Solver name: ', 's');
solvername = 'C12';
%force an acceptable input for the variable solvername
flag = 0;
while flag == 0
   for i = 1:length(solverNames)
      if strcmp(solvername,solverNames(i)) == 1
         flag = 1;
      end
   end
   if flag == 0
      solvername = input(' The solver name must match one of the available solvers: ', 's');
   end
end

%create a SAPI solver handle
solver = sapiSolver(conn, solvername);

%create a QSage solver
%qsage = input('Would you like to create a QSage solver (y/n)? ', 's');
qsage = 'n';
%force an acceptable input for the variable qsage
flag = 0;
while flag == 0
   if qsage == 'y'
      QSageSolver = sapiSolveQSage(solver);
      flag = 1;
   elseif qsage == 'n'
      flag = 1;
   else
      input(' You must type y or n: ');
   end
end

%properties of the connected solver
props = sapiSolverProperties(solver);
display('Solver properties: ')
disp(props)

%properties of the QSage solver
if qsage == 'y'
   qsageprops = sapiSolverProperties(QSageSolver);
```

```
    display('QSage properties: ')
    disp(qsageprops)
end

%extract the properties
annealing_time_range = props.annealing_time_range;
chip_id = props.chip_id;
couplers = props.couplers;
default_annealing_time = props.default_annealing_time;
default_programming_thermalization = props.default_programming_thermalization;
default_readout_thermalization = props.default_readout_thermalization;
h_range = props.h_range;
j_range = props.j_range;
num_qubits = props.num_qubits;
parameters = props.parameters;
programming_thermalization_range = props.programming_thermalization_range;
qubits = props.qubits;
server_version = props.server_version;
supported_problem_types = props.supported_problem_types;

%store the properties
%properties = [annealing_time_range, chip_id, couplers, default_annealing_time,
default_programming_thermalization, default_readout_thermalization, h_range, j_range,
num_qubits, parameters, programming_thermalization_range, qubits, server_version,
supported_problem_types];

%save the solver's properties to a file
%save properties.dat props

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the topology of the chip and the section on which to compute
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%size of the chip
%x = input('Number of columns of K44 arrangements: ');
x = 12;
%y = input('Number of rows of K44 arrangements: ');
y = 12;

%number of qubits
n = 8*x*y;

%non-working qubits
```

```
%qnw = input('Non-working qubits, counting from 0 (vector): ');
qnw = [26; 49; 52; 53; 98; 103; 105; 110; 168; 175; 201; 208; 215; 304; 338; 341; 372;
402; 405; 410; 413; 447; 455; 480; 485; 534; 539; 540; 576; 599; 602; 606; 642; 647;
717; 833; 853; 854; 883; 895; 905; 911; 936; 938; 943; 997; 1006; 1014; 1033; 1034;
1036; 1042; 1047; 1094; 1096]; %(counting from 0) for C12
qnw = qnw + 1; %count from 1

%non-working couplers
%cnw = input('Non-working couplers (two-column matrix): ');
cnw =[344,351]; %(counting from 0) for C12
cnw = cnw + 1; %count from 1

display('Define the rectangle on which to compute.')
%x0 = input(' Column-coordinate of upper-left-corner K44 group: ');
%y0 = input(' Row-coordinate of upper-left-corner K44 group: ');
%ulc = input(' Qubit in the upper-left corner of the rectangle: ');
ulc = 1;
%a = input(' Column-dimension of the rectangle: ');
a = 12;
%b = input(' Row-dimension of the rectangle: ');
b = 12;

%ulc = 1 + 8*x*(y0-1) + 8*(x0-1);
lrc = ulc + 8*x*(b-1)+ (8*a - 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Specify the run-time parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Note: Default answer mode is 'histogram'.
%Note: Default number of reads is 10.
%Note: Default maximum number of answers is 1 000 for 'histogram' and num_reads for
all answers ('raw').
%Note: Default annealing time is 20 us.
%Note: Default post programming thermalization time is 1 000 us.
%Note: Default post readout thermalization time is 0 us.
%Note: Default automatic scaling is true.

%answer_mode = input('Answer mode (histogram or raw): ', 's');
answer_mode = 'histogram';
%num_reads = input('Number of reads (element of [1,1000]): ');
num_reads = 1000;
if strcmp(answer_mode,'histogram') == 1
```

```
    %max_answers = input('Maximum number of answers (element of [1,1000]): ');
    max_answers = 1000;
else
    max_answers = input('Maximum number of answers (element of [1,number of reads]):
');
end

display('NOTE: Maximum job duration excluding readout time is 1000000.0 us.')
%annealing_time = input(' Annealing time per read (element of [1,20000] in us): ');
annealing_time = 20;
%programming_thermalization = input(' Post programming thermalization time per read
(element of [0,10000] in us): ');
programming_thermalization = 1000;
%readout_thermalization = input(' Post readout thermalization time per read (element of
[0,10000] in us): ');
readout_thermalization = 0;

%auto_scale = input('Automatic scaling (true or false): ');
auto_scale = false;

%store the parameters, including the date and time
%parameters = [datenum(clock); answer_mode; num_reads; max_answers;
annealing_time; programming_thermalization; readout_thermalization; auto_scale;]';

%save the annealing parameters to a file
%save parameters.dat parameters -ASCII

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%BEGIN
PROGRAM%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the graph and assign the interactions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create a null matrix for the graph/interactions (adjacency matrix)
J = zeros(n,n); %preallocate

%I = input('Interaction (element of [-1,1], glass, or random): ', 's');
I = -1;
```

```
%define the Edwards-Anderson glass interaction, or . . .
if strcmp(I,'glass') == 1
   gI = [-1 1];

%. . . define the bounds of the random distribution, or . . .
elseif strcmp(I,'random') == 1
   %instructions
   display(' Define the bounds of the random distribution.')

   lI = input('  Lower bound (element of [-1,1]): ');
   %check that the lower bound is possible
   while or(lI < j_range(1), lI > j_range(2))
      lI = input('   Lower bound must be an element of [-1,1]): ');
   end

   uI = input('  Upper bound (element of [lower bound,1]): ');
   %check that the upper bound is possible
   while or(uI < lI, uI > j_range(2))
      uI = input('   Upper bound must be an element of [lower bound,1]): ');
   end

%. . . check that the given interaction is possible
else
   while or(I < j_range(1), I > j_range(2))
      I = input(' Interaction must be an element of [-1,1]): ');
   end
end

%define the chimera graph of a fully working, n-qubit chip by constructing its adjacency
matrix and assign the interactions
for row = 1:y
   for col = 1:x
      for k1 = 1:4 %red vertices in a K44 arrangement
         for k2 = 5:8 %blue vertices in a K44 arrangement

            %define the intra-connectivity of each K44 arrangement (bipartite, fully
connected) and assign the interactions
            i = k1 + 8*(col-1) + 8*x*(row-1); %red vertex/qubit
            j = k2 + 8*(col-1) + 8*x*(row-1); %blue vertex/qubit
            if strcmp(I,'glass') == 1
               J(i,j) = gI(randi(2)); %edge/interaction
            elseif strcmp(I,'random') == 1
               J(i,j) = lI + (uI-lI)*rand; %edge/interaction
            else
```

```
            J(i,j) = I; %edge/interaction
        end

        %define the row inter-connectivity (column intra-connectivity) and assign the
interactions
        if row ~= y %check for the bottom boundary
            if strcmp(I,'glass') == 1
                J(i,i+8*x) = gI(randi(2)); %edge/interaction
            elseif strcmp(I,'random') == 1
                J(i,i+8*x) = lI + (uI-lI)*rand; %edge/interaction
            else
                J(i,i+8*x) = I; %edge/interaction
            end
        end

        %define the column inter-connectivity (row intra-connectivity) and assign the
interactions
        if col ~= x %check for the right boundary
            if strcmp(I,'glass') == 1
                J(j,j+8) = gI(randi(2)); %edge/interaction
            elseif strcmp(I,'random') == 1
                J(j,j+8) = lI + (uI-lI)*rand; %edge/interaction
            else
                J(j,j+8) = I; %edge/interaction
            end
        end

      end
    end
  end
end

%inactivate the non-working qubits
if isempty(qnw) ~= 1
  for i = 1:length(qnw)
    J(:,qnw(i)) = 0;
    J(qnw(i),:) = 0;
  end
end

%inactivate the non-working couplers
dim = size(cnw);
if isempty(cnw) ~= 1
  for i = 1:dim(1)
    J(cnw(i,1),cnw(i,2)) = 0;
```

```
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Create
subgraph%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%inactivate qubits 1 through ulc-1
if ulc ~= 1
    for i = 1:ulc-1
        J(:,i) = 0;
        J(i,:) = 0;
    end
end

%inactivate qubits lrc+1 through n
if lrc ~= n
    for i = lrc+1:n
        J(:,i) = 0;
        J(i,:) = 0;
    end
end

%inactivate the rest of the qubits on the left and right sides of the rectangle
if a ~= x
    if b ~= 1
        for k = 1:b-1
            for i = ulc + 8*a + 8*x*(k-1):ulc + 8*x*k - 1
                J(i,:) = 0;
                J(:,i) = 0;
            end
        end
    end
end

%save the interactions to a file
save J.dat J -ASCII

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Assign the magnetic fields
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H = input('Magnetic field (element of [-2,2], glass, random): ', 's');
```

```
%define the Edwards-Anderson glass interaction, or . . .
if strcmp(H,'glass') == 1
   gH = [-1 1];

%. . . define the bounds of the random distribution, or . . .
elseif strcmp(H,'random') == 1
   %instructions
   display(' Define the bounds of the random distribution.')

   lH = input('  Lower bound (element of [-2,2]): ');
   %check that the lower bound is possible
   while or(lI < h_range(1), lI > h_range(2))
      lH = input('   Lower bound must be an element of [-2,2]): ');
   end

   uH = input('  Upper bound (element of [lower bound,2]): ');
   %check that the upper bound is possible
   while or(uI < lI, uI > h_range(2))
      uH = input('   Upper bound must be an element of [lower bound,2]): ');
   end

%. . . check that the given field is possible
else
   while or(H < h_range(1), H > h_range(2))
      H = input(' Field must be an element of [-2,2]): ');
   end
end

%assign the magnetic fields
h = zeros(n,1); %preallocate
if strcmp(H,'glass') == 1
   for i = 1:n
      h(i) = gH(randi(2));
   end
elseif strcmp(H,'random')
   h = lH + (uH-lH).*rand(n,1);
else
   for i = 1:n
      h(i) = H;
   end
end

%inactivate non-working qubits
if isempty(qnw) ~= 1
   for i = 1:length(qnw)
```

```
        h(qnw(i)) = 0;
    end
end

%save the magnetic fields to a file
save h.dat h -ASCII

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%END
PROGRAM%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Submit the problem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%solve the Ising problem, passing in the parameters
answer = sapiSolveIsing(solver, h, J, 'programming_thermalization',
programming_thermalization, 'num_reads', num_reads, 'max_answers', max_answers,
'readout_thermalization', readout_thermalization, 'answer_mode', answer_mode,
'auto_scale', auto_scale, 'annealing_time', annealing_time);

%Solve the Ising problem asynchronously
%result = sapiAsyncSolveIsing(solver, h, J, 'programming_thermalization',
programming_thermalization, 'num_reads', num_reads, 'max_answers', max_answers,
'readout_thermalization', readout_thermalization, 'answer_mode', answer_mode,
'auto_scale', auto_scale, 'annealing_time', annealing_time);
%while ~sapiAsyncDone(result)
%    pause(1);
%end
%answer = sapiAsyncResult(result);

%solve a QUBO problem, passing in the parameters
%answer = sapiSolveQubo(solver, Q);

%Solve a QUBO problem asynchronously
%result = sapiAsyncSolveQubo(solver, Q);
%while ~sapiAsyncDone(result)
%    pause(1);
%end
%answer = sapiAsyncResult(result);
```

```
%record the approximate time of submission (this time recorded will be later than the
actual time)
timesub = datenum(clock);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Format and save the returned answer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%extract the returned data from answer
solutions = answer.solutions;
energies = answer.energies;
num_occurrences = answer.num_occurrences;
timing = answer.timing;

%store the lowest-energy solution
%gs = solutions(:,1);

%find and store the errors in the lowest-energy solution
j = 0;
for i = 1:n
   if solutions(i,1) ~= 1
      j = j + 1;
      errors(j,1) = i;
      errors(j,2) = solutions(i,1);
   end
end

%format the returned data
energies = energies';
num_occurrences = num_occurrences';
timing = timing';

%store the lowest energy, its number of occurrences, and the date & time
%lowest(1,1) = energies(1);
%lowest(1,2) = num_occurrences(1);
%lowest(1,3) = datenum(clock);

%check to see if the there are two solutions associated with the lowest energy
%if length(energies) ~= 1
%   if energies(1) == energies (2)
%      lowest(1,2) = num_occurrences(1) + num_occurrences(2);
%   else
```

```
%       lowest(1,2) = num_occurrences(1);
%    end
%end

%save the solver's name, the run-time parameters, and the returned data to files
%save output.dat solvername properties parameters timesub solutions energies
num_occurrences timing -ASCII
%save output.dat solvername timesub solutions energies num_occurrences -ASCII
%save lowest.dat lowest -ASCII

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the Graph
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create a null matrix for the graph (adjacency matrix)
adj = zeros(n,n); %preallocate

%define the graph by constructing its adjacency matrix
for i = 1:n
   adj(i,i) = 1; %include reflexive loops so that every vertex/qubit is plotted
   for j = i:n
      if J(i,j) ~= 0
         adj(i,j) = 1;
      end
   end
end

%inactivate the non-working qubits
for i = 1:length(qnw)
   adj(qnw(i),qnw(i)) = 0;
end

%associate each vertex/qubit with a Cartesian coordinate
coord = zeros(n,2);
for row = 1:y
   for col = 1:x
      for k1 = 1:4
         i = k1 + 8*(col-1) + 8*x*(row-1);
         j = (k1+4) + 8*(col-1) + 8*x*(row-1);
         coord(i,1) = col+2*(col-1);
         coord(i,2) = k1+5*(row-1);
         coord(j,1) = col+2*(col-1)+1;
         coord(j,2) = k1+5*(row-1);
```

```
        end
    end
end

%plot the graph
gplot(adj,coord)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Save the problem in a format for cutting and pasting into the web interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%this must be the first line
first_line = [num_qubits length(qubits)+length(couplers)];
save web.dat first_line -ASCII

%format the bias data
k = 1;
flag = 0;
for i=1:n
    for j = 1:length(qnw)
        if qnw(j) == i
            flag = 1; %do not write the qnw---the website doesn't like that
        end
    end
    if flag == 0
        web_qubits(k,1)=i-1;
        web_qubits(k,2)=i-1;
        web_qubits(k,3)=h(i);
        k = k + 1;
    end
    flag = 0; %set the flag back to 0 for the next round
end
save web.dat web_qubits -append -ASCII

%format the coupler data
k=1;
for i=1:n
    for j=i:1152
        if J(i,j)~=0
            web_couplers(k,1) = i-1;
            web_couplers(k,2) = j-1;
            web_couplers(k,3) = J(i,j);
            k = k + 1;
```

```
        end
    end
end
save web.dat web_couplers -append -ASCII
```