An Evaluation of the Performance of the Quantum Adiabatic Algorithm on Random Instances of k-SAT

by

James McBride

Submitted to the Department of Physics in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

© James McBride, MMII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author

Department of Physics

Certified by

Edward Farhi
Professor
Thesis Supervisor

Accepted by

Chairman, Department Committee on Graduate Students
An Evaluation of the Performance of the Quantum Adiabatic Algorithm on Random Instances of k-SAT

by

James McBride

Submitted to the Department of Physics
on August 23, 2002, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

A quantum computational framework has been developed based on the adiabatic theorem. The theorem guarantees that a system with a time dependent Hamiltonian that is placed into its ground state will remain in its ground state provided that the Hamiltonian of the system varies slowly enough with time.

This work investigates the performance of the quantum adiabatic algorithm on random instances of k-SAT. The performance of the algorithm is examined on subsets of k-SAT that are classically easy and on subsets that are classically difficult. The evaluation attempts to determine how the typical time required to solve the problems grows with the size of the problems.

This evaluation is done by directly determining the required time from numerical integration of the dynamics of the system and by inferring this time using a result from the adiabatic theorem. This evaluation considers quantum systems composed of up to 23 bits and is performed on several large scale Beowulf clusters. As was seen in previous work, the direct integration studies show what appears to be only a quadratic growth rate in the required running time with the number of bits in problems that classically require exponential time. However, further studies show that these effects are caused by polynomial bounded matrix elements and are not indicative of the asymptotic behavior of the performance of the algorithm.

The real asymptotic scaling of the performance of the algorithm is controlled by the ground state energy gap. When this is examined directly it is not currently possible to determine whether the growth rate of the running time of the algorithm is exponential or polynomial.

Thesis Supervisor: Edward Farhi
Title: Professor
Acknowledgments

I have been very fortunate to have met some of the most amazing people while working on this degree.

First to my advisor Eddie Farhi who has helped me every step of the way. I hope at least a little bit of the truthseeker in him has worn off on me.

To my readers Prof. Goldstone and Prof. Levitov, I could not have asked for two more helpful and thoughtful people to help me put together this thesis.

Thanks to Daniel Preda who helped me along any number of times when he and I started this work together.

To Ike Chuang and Geva Patz for working with us on this project, providing a constant check that things were going well, and for letting us take over all of their computers for the last several months.

I’d also like to thank Ron Choy and Alan Edelman for providing us yet another needed source of computer time and help in getting our software up and running.

To Mike Bove and the whole community of people at the Media Lab for giving me a home away from home and exposing me to MIT life outside the physics department.

To Jack Wisdom from whom I’ve learned so much about so many things. And to Gerry Sussman who has helped me in every possible fashion.

To Kristin Burgess without whom this thesis would have never been started or finished. I would be lost without her.

And finally to my parents and sister who have always tried their hardest to keep me happy and sane, even when I had other ideas.
Contents

1 Introduction .................................................. 15
   1.1 Quantum Algorithms .................................. 16
       1.1.1 Deutsch’s Algorithm .......................... 16
       1.1.2 Grover’s Algorithm ......................... 18
   1.2 Adiabatic Quantum Computation .................... 20
       1.2.1 The Adiabatic Theorem ....................... 20
       1.2.2 Using Adiabatic Evolution to Perform Computation .... 23
   1.3 k-SAT .................................................... 24
       1.3.1 SATISFIABILITY ............................... 25
       1.3.2 2-SAT ........................................... 26
       1.3.3 3-SAT ........................................... 27
       1.3.4 (2+p)-SAT .................................... 28

2 Details ....................................................... 31
   2.1 Instance Generation .................................. 31
   2.2 Construction of the Hamiltonians ................ 34
       2.2.1 The Problem Hamiltonian \( H_p \) .............. 34
       2.2.2 The Beginning Hamiltonian \( H_b \) .......... 36
   2.3 Previous Work ......................................... 37

3 Integration of Schrödinger’s Equation ................. 41
   3.1 Details of the Integration Search Strategy ... 42
   3.2 Results ............................................... 42
3.2.1 2-SAT and 3-SAT .................................................. 42
3.2.2 Scrambled Instances ............................................... 45
3.3 Limitations .............................................................. 45

4 Minimum Energy Gap ..................................................... 49
  4.1 Running Times from the Adiabatic Theorem ...................... 49
  4.2 Details of the Eigenvalue Search Strategy ...................... 52
  4.3 Results ................................................................. 53
    4.3.1 2-SAT and 3-SAT ............................................... 53
    4.3.2 Scrambled Instances .......................................... 56

5 Conclusions .............................................................. 63

A Numerical Implementation of $H_b$ .................................. 65

B The Lanczos Method for Finding Eigenvalues of Sparse Matrices 67

C Characteristics of the Distributions of the Data .................. 71
  C.1 Integration Studies ............................................... 73
    C.1.1 3-SAT ............................................................. 73
    C.1.2 Scrambled 3-SAT .............................................. 77
  C.2 Energy Gap Studies ................................................ 79
    C.2.1 3-SAT ............................................................. 79
    C.2.2 Scrambled 3-SAT .............................................. 87
# List of Figures

1-1 Diagrammatic Solution of the Two Instances of 2-SAT Given in the Text 27

2-1 Number of Clauses vs. Number of Bits for 3 Values of p ............ 34

2-2 Performance of the Adiabatic Algorithm on EC2 and EC3 (Figures taken from [FGG00]) .................................................. 38

2-3 Performance of the Adiabatic Algorithm on Scrambled Instances of EC2 (Figures taken from [FGG00]) .............................. 39

2-4 Performance of the Adiabatic Algorithm out to 20 Bits (Figures taken from [FGG+01]) .............................................................. 40

3-1 Probability as a Function of T for Three 6 Bit Instances ............ 43

3-2 Increase in Required Runtime as a Function of Bit Number for 2-SAT and 3-SAT (top) and Scrambled Instances of 2-SAT and 3-SAT (bottom) 46

3-3 Distribution of Probabilities for 1000 Instances at 10 Bits ........... 47

4-1 Does the Adiabatic Formula Determine the Required Running Time? 50

4-2 Energy Gap vs. s for a 10 Bit Instance of 3-SAT ...................... 51

4-3 An Example of a 10 Bit Instance with a Double Minimum ............ 53

4-4 Data from Table 4.1 on a linear/linear Scale ............................. 54

4-5 Data from Table 4.1 on a linear/log₂ Scale (top) and a log₂ / log₂ Scale (bottom) ................................................................. 55

4-6 Data from Table 4.2 .............................................................. 58

4-7 Data for Bits 10-23 from table 4.2. log₂(Minimum Gap) vs Number of Bits and log₂(Minimum Gap) vs log₂(Number of Bits) ........... 59
Data for Bits 15-23 from table 4.2. $\log_2(\text{Minimum Gap})$ vs Number of Bits and $\log_2(\text{Minimum Gap})$ vs $\log_2(\text{Number of Bits})$.

Progression of the Slope of the Scrambled Data.

Distribution of Running Times for 1000 3-SAT Instances at 6 Bits.

Distribution of Running Times for 1000 3-SAT Instances at 10 Bits.

Distribution of Probabilities for 1000 3-SAT Instances at 6 Bits Run for Time $T=35.4$.

Distribution of Probabilities for 1000 3-SAT Instances at 10 Bits Run for Time $T=71.4$.

Distribution of Running Times for 1000 Scrambled 3-SAT Instances at 6 Bits.

Distribution of Probabilities for 1000 Scrambled 3-SAT Instances at 6 Bits Run for Time $T=57.3$.

Distribution of Minimum Gaps for 3700 3-SAT Instances at 10 Bits.

Distribution of $\log_2(\text{Minimum Gap})$ for 3700 3-SAT Instances at 10 Bits (Same data as figure C-7).

Distribution of Minimum Gaps for 3700 3-SAT Instances at 15 Bits.

Distribution of $\log_2(\text{Minimum Gap})$ for 3700 3-SAT Instances at 15 Bits (Same data as figure C-9).

Distribution of Minimum Gaps for 931 3-SAT Instances at 20 Bits.

Distribution of $\log_2(\text{Minimum Gap})$ for 931 3-SAT Instances at 20 Bits. (Same data as figure C-11).

Distribution of Minimum Gaps for 592 3-SAT Instances at 22 Bits.

Distribution of $\log_2(\text{Minimum Gap})$ for 592 3-SAT Instances at 22 Bits. Same data as figure C-13.

Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 10 Bits.

Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 15 Bits.
C-17 Distribution of Minimum Gaps for 214 Scrambled 3-SAT Instances at
20 Bits ......................................................... 89

C-18 Distribution of Minimum Gaps for 139 Scrambled 3-SAT Instances at
22 Bits ......................................................... 90
### List of Tables

3.1 2-SAT and 3-SAT Data Summary ........................................ 44
3.2 Scrambled 2-SAT and Scrambled 3-SAT Data Summary ............. 44
4.1 2-SAT and 3-SAT Minimum Energy Gap Data ....................... 54
4.2 Scrambled 2-SAT and Scrambled 3-SAT Minimum Energy Gap Data 56
C.1 Statistics of the Distribution of Running Times for 1000 3-SAT Instances at 6 Bits .................................................. 73
C.2 Statistics of the Distribution of Running Times for 1000 3-SAT Instances at 10 Bits .................................................... 74
C.3 Statistics of the Distribution of Probabilities for 1000 3-SAT Instances at 6 Bits Run for Time T=35.4 ............................. 75
C.4 Statistics of the Distribution of Probabilities for 1000 3-SAT Instances at 10 Bits Run for Time T=71.4 .............................. 76
C.5 Statistics of the Distribution of Running Times for 1000 Scrambled 3-SAT Instances at 6 Bits ........................................... 77
C.6 Statistics of the Distribution of Probabilities for 1000 Scrambled 3-SAT Instances at 6 Bits Run for T=57.3 ........................ 78
C.7 Statistics of the Distribution of Minimum Gaps for 3700 3-SAT Instances at 10 Bits ....................................................... 79
C.8 Statistics of the Distribution of Minimum Gaps for 3700 3-SAT Instances at 15 Bits .......................................................... 81
C.9 Statistics of the Distribution of Minimum Gaps for 931 3-SAT Instances at 20 Bits ............................................................. 83
C.10 Statistics of the Distribution of Minimum Gaps for 592 3-SAT Instances at 22 Bits .............................................................. 85
C.11 Statistics of the Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 10 Bits .................................................. 87
C.12 Statistics of the Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 15 Bits .................................................. 88
C.13 Statistics of the Distribution of Minimum Gaps for 214 Scrambled 3-SAT Instances at 20 Bits .................................................. 89
C.14 Statistics of the Distribution of Minimum Gaps for 139 Scrambled 3-SAT Instances at 22 Bits .................................................. 90


Chapter 1

Introduction

All computations performed by conventional computers are classical computations. The computer's memory, abstractly thought of as a large collection of ones and zeros, is in a definite state during all parts of the computation. During the computation, sections of the memory may be read and modified. Traditionally the calculation is thought to proceed by the action of a number of classical gates which read a small number of memory locations and possibly modify a few others. Throughout the computation each bit is stored as a classical variable; its complete state can be queried at any point during the computation to make sure that the computation is proceeding correctly without interfering with the progress of the calculation.

Quantum computation replaces the classical memory storage device with a device that can be interacted with at the quantum level. Abstractly this is often thought of as replacing the classical ones and zeros with quantum spins referred to as qubits (an upward pointing spin represents a zero and a downward pointing spin represents a one). This replacement still allows the possibility that the memory exists in a definite state of ones and zeros, but because the spins are quantum mechanical the memory can now also be in a superposition of these states. Additionally, the computation can now be thought of as the action of a sequence of quantum mechanical operators instead of classical gates.

This replacement of classical ones and zeros with quantum spins and classical gates with quantum operators produces a system that is computationally more powerful.
This increased power has been demonstrated by the construction of several quantum algorithms that are more powerful than their classical counterparts. Two of these algorithms will be discussed in the next section.

More recently, a method of construction of quantum algorithms has been presented that relies on the adiabatic theorem. This thesis explores the performance of such algorithms on sets of problem that are known classically to be computationally easy and on sets of problem that are known classically to be computationally difficult. A problem is considered to be computationally easy if the required running time grows like a polynomial in the size of the problem. If the required running time grows faster than any polynomial in the size of the problem, the problem is considered hard.

Unlike most quantum computational algorithms, these adiabatic algorithms naturally operate in continuous time. Instead of proceeding with the operation of a sequence of quantum gates on a prepared initial state, a time dependent Hamiltonian is constructed that slowly evolves that state into one from which the answer to the computational problem can be read. This feature allows the performance to be evaluated using many of the traditional tools used to study the dynamics of quantum systems.

The remainder of this chapter presents a brief overview of the most well know quantum algorithms. A proof of the adiabatic theorem is then given and followed by an explanation of how the results of that theorem can be used to produce quantum algorithms. The chapter ends with a discussion of the set of computational problems known as SATISFIABILITY. All of the problems used to evaluate the performance of the adiabatic algorithms are drawn from this set.

\section{1.1 Quantum Algorithms}

\subsection{1.1.1 Deutsch’s Algorithm}

Deutsch's algorithm solves a simple problem. Imagine that you have a function $f$ that takes a single bit as input and returns a single bit as output, and you would like
to determine whether \( f \) is a constant function. In order to determine this classically it would be necessary to call \( f \) twice, once on each of its possible inputs. However, if \( f \) can be applied quantum mechanically this can be accomplished with a single application. The discussion of this algorithm also provides an opportunity to present the notation often used in quantum computation in the context of a relatively simple application.

In order to apply the function \( f \) quantum mechanically we imagine having an operator with the following behavior

\[
U_f : |x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle. \tag{1.1}
\]

Here \( \oplus \) is the bitwise \textbf{XOR}. If the spin that makes up the \( |y\rangle \) portion of the state is initially set to zero, the action of \( U_f \) is to leave the result, \( f(x) \), stored on that spin. Further if the spin that makes up the \( |x\rangle \) portion of the state is put into a superposition over the possible values of \( x \), the action of \( U_f \) causes the spin that makes up the \( |y\rangle \) component of the state to be in a superposition of all the possible output values of \( f \). The ability to create such a superposition with one application of the operator that represents \( f \) is often called quantum parallelism.

Taking the function \( f \) from Deutsch's problem as an example, if the \( |y\rangle \) portion of the initial state is placed into the state \( \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \), then

\[
U_f : |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \rightarrow |x\rangle \frac{1}{\sqrt{2}}(|f(x)\rangle - |1 \oplus f(x)\rangle) \\
= |x\rangle (-1)^{f(x)} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{1.2}
\]

Now placing the \( |x\rangle \) bits in a superposition, the action of the operator is

\[
U_f : \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \rightarrow \frac{1}{\sqrt{2}} \left[ (-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle \right] \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{1.4}
\]
If \( f \) is constant, the resulting state is
\[
\left\{ \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \  \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right\} \text{ if } f = 0
\]

or
\[
-\left\{ \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \  \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right\} \text{ if } f = 1.
\]  

However if \( f \) is not constant, the resulting state is
\[
\left\{ \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \  \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right\} \text{ if } f(0) = 0 \text{ and } f(1) = 1
\]

or
\[
-\left\{ \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \  \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right\} \text{ if } f(0) = 1 \text{ and } f(1) = 0.
\]

Measuring the \( x \) bit in the basis made from \( \left\{ \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle), \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right\} \) will determine with certainty if \( f \) is constant after a single application of the operator that applies \( f \).

While Deutsch’s algorithm solves the problem of determining whether \( f \) is constant, it does not do so any more efficiently than is classically possible. This is because the single application of \( U_f \) requires two calls to the classical function \( f \). The extra call is required because \( f \) must be implemented in a reversible fashion if it is to be represented by a unitary operator.

### 1.1.2 Grover’s Algorithm

Grover’s algorithm solves the problem of searching through an unstructured list. Classically one would have to look, on average, at half of the elements in the list before finding the one element in the list that is being sought, so the classical algorithm is \( O(N) \). An algorithm is \( O(N) \) if the number of steps in the algorithm grows no faster than linearly with \( N \). In this case \( N \) is the size of the list to be searched. Grover’s
algorithm accomplishes this task in $O(\sqrt{N})$ steps.

To see how the algorithm works\(^1\), imagine that the items in the list are numbered $1 \ldots N$ and that the search takes the form of locating the unique input $w \in \{1 \ldots N\}$ to a function $f : \{1 \ldots N\} \rightarrow \{0, 1\}$ such that $f(w) = 1$. The function $f$ takes the value 0 for all other inputs. Using the technique described in section 1.1.1 it is possible to construct an operator with the following action

$$U_f : |x\rangle \rightarrow (-1)^{f(x)} |x\rangle.$$  

(1.9)

If the sought item is number $w$ then $f(w) = 1$, and the action of $U_f$ can be written

$$U_f = 1 - 2 |w\rangle \langle w|.$$  

(1.10)

One additional operator is needed in the operation of the algorithm,

$$U_s = 2 |s\rangle \langle s| - 1 \quad \text{where} \quad |s\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle.$$  

(1.11)

The action of the combined operator $U_G = U_s U_f$ in the subspace spanned by $|w\rangle$ and $|r\rangle = \frac{1}{\sqrt{N-1}} (\sqrt{N} |s\rangle - |w\rangle)$ can be written as

$$U_G = U_s U_f = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}.$$  

(1.12)

In this expression $\cos \theta = 1 - \frac{2}{N}$.

The algorithm operates by repeatedly applying $U_G$ to an initial state $|s\rangle$ which is a superposition over all the possible inputs to $f$. The action of $U_G$ rotates the state $|r\rangle$ into $|w\rangle$, and for large values of $N$ each application of $U_G$ rotates that state by an angle of $\theta \approx \frac{2}{\sqrt{N}}$. Since $|w\rangle$ and $|r\rangle$ are perpendicular and for large $N$ the state $|s\rangle$ is nearly aligned with $|r\rangle$, after $\frac{\pi}{4} \sqrt{N}$ applications of $U_G$ the initial state has been rotated into $|w\rangle$, and a measurement of the spins will yield the location of the sought

\(^1\)This explanation was originally given by Farhi et al., quant-ph/9612026.
item with high probability. Since, as discussed in section 1.1.1, each application of $U_G$ requires two calls of the function $f$, the algorithm is able to search through the entire unstructured list with only $O(\sqrt{N})$ calls of $f$. It has been proven that the speed of this algorithm cannot be improved.\footnote{Bennett \textit{et al.}, “Strengths and Weaknesses of Quantum Computing” SIAM Journal on Computing Vol 26, #5, pp 1510-1523 and quant-ph/9711070.}

1.2 Adiabatic Quantum Computation

1.2.1 The Adiabatic Theorem

The adiabatic theorem gives a precise statement of a reasonable idea: if the dynamics of a quantum mechanical system are governed by a Hamiltonian that varies slowly with time, then a system placed in the $n^{th}$ eigenstate will remain in the $n^{th}$ eigenstate under the action of that Hamiltonian provided that the $n^{th}$ eigenstate is nondegenerate at all points during the evolution.

The formalization of this idea in the adiabatic theorem gives an estimate of the transition probability to other states during the evolution.\footnote{This proof follows the one given in Liboff, “Introductory Quantum Mechanics” Third edition, Addison Wesley 1997.} Begin by writing the total Hamiltonian as a time dependent piece and a time independent piece

$$H(t) = H_0 + H'(t).$$

(1.13)

The time dependent eigenvectors of $H_0$ are

$$|\psi_1(t)\rangle = |\psi_1(t = 0)\rangle e^{-iE_1 t/\hbar} \equiv |\psi_1\rangle e^{-iE_1 t/\hbar}.$$ 

(1.14)

Any state $|\psi(t)\rangle$ can be expanded in terms of the eigenvectors of $H_0$ as

$$|\psi(t)\rangle = \sum_{i=1}^{N} c_i(t) |\psi_i(t)\rangle.$$ 

(1.15)
If the system begins in the $j^{th}$ eigenstate of $H_0$, first order perturbation theory gives the value of these expansion coefficients at a time $t$ as

$$c_k(t) = \frac{1}{i\hbar} \int_{t'=0}^{t'} \langle \psi_k(t') | H'(t') | \psi_j(t') \rangle \, dt' \quad \text{for} \quad k \neq j. \quad (1.16)$$

The time dependence of this equation can be written explicitly as

$$c_k(t) = \frac{1}{i\hbar} \int_{t'=0}^{t'} \langle \psi_k | H'(t') | \psi_j \rangle \, e^{i(E_k - E_j)/\hbar \, dt'} \quad \text{for} \quad k \neq j. \quad (1.17)$$

If the Hamiltonian changes slowly with time, the time derivative of $H'(t)$ is small. To incorporate this fact into the expansion, equation 1.17 can be integrated by parts

$$c_k(t) = -\frac{1}{E_k - E_j} \left\{ \langle \psi_k | H'(t') | \psi_j \rangle \, e^{i(E_k - E_j)/\hbar \, t'} \right|_{t'=0}^{t'=t} - \int_{t'=0}^{t' t} \langle \psi_k | \frac{\partial H'(t')}{\partial t'} | \psi_j \rangle \, e^{i(E_k - E_j)/\hbar \, dt'} \right\} \quad \text{for} \quad k \neq j. \quad (1.18)$$

If the time dependence of $H'(t)$ is small, then the second term in equation 1.18 can be neglected. Further, if $H'(t)$ is zero at the beginning of the interval $c_k(t)$ can be approximated as

$$c_k(t) \approx -\frac{\langle \psi_k | H'(t) | \psi_j \rangle}{(E_k - E_j)} \, e^{i(E_k - E_j)/\hbar} \quad \text{for} \quad k \neq j. \quad (1.19)$$

Since the system starts in the $j^{th}$ eigenstate and the perturbing Hamiltonian changes slowly with time we can write

$$|\psi(t)\rangle \approx |\phi\rangle \, e^{-iE_j t/\hbar}. \quad (1.20)$$

However $|\psi(t)\rangle$ can also be written as

$$|\psi(t)\rangle = |\psi_j(t)\rangle + \sum_{i \neq j} c_i(t) \, |\psi_i(t)\rangle. \quad (1.21)$$
Combining these two results and using equation 1.19 gives

\[ |\phi\rangle = |\phi_j\rangle + \sum_{i \neq j} \frac{\langle \psi_i | \mathbf{H}'(t) | \psi_j \rangle}{E_j - E_i} |\phi_i\rangle. \] (1.22)

This is identical to the result that would have been obtained from considering a time independent perturbation to the system. This says that provided the second term in equation 1.18 can be ignored, the eigenstates of the system change with time as if the perturbation were time independent. A system initially placed in the \( j \)th eigenstate will remain in the \( j \)th state even as the state evolves.

Looking at the second term in equation 1.18, it is possible to derive a constraint on how quickly \( \mathbf{H}'(t) \) may change with time. The magnitude of this term can be approximated as

\[
\left| \int_{t'=0}^{t'=t} e^{i (E_k - E_j)/\hbar} \langle \psi_k | \frac{\partial \mathbf{H}'(t')}{\partial t'} | \psi_j \rangle \, dt' \right|
\approx \left| \langle \psi_k | \frac{\partial \mathbf{H}'(t)}{\partial t} | \psi_j \rangle \right| \left| \int_{t'=0}^{t'=t} e^{i (E_k - E_j)/\hbar} \, dt' \right|
\approx \frac{2\hbar}{(E_k - E_j)} \left| \langle \psi_k | \frac{\partial \mathbf{H}'(t)}{\partial t} | \psi_j \rangle \right| \sin \left( \frac{t(E_k - E_j)}{2\hbar} \right). \] (1.23)

The term can be neglected as long as

\[
\left| \frac{E_k - E_j}{\hbar} \langle \psi_k | \mathbf{H}'(t) | \psi_j \rangle \right| \gg 2 \left| \langle \psi_k | \frac{\partial \mathbf{H}'(t)}{\partial t} | \psi_j \rangle \right|. \] (1.24)

In what follows it will be necessary to estimate the probability that a system initially placed in the \( j \)th eigenstate does not stay in that state during the evolution. This can be found by looking at the magnitude of the coefficients in equation 1.18.

\[
|c_k(t)|^2 \approx \left( \frac{1}{E_k - E_j} \right)^2 \left| \langle \psi_k | \mathbf{H}'(t) | \psi_j \rangle \right|^2 + \left( \frac{2\hbar}{E_k - E_j} \right)^2 \left| \langle \psi_k | \frac{\partial \mathbf{H}'(t)}{\partial t} | \psi_j \rangle \right|^2 \sin^2 \left( \frac{t(E_k - E_j)}{2\hbar} \right) - \frac{2\hbar}{E_k - E_j} \sin \left( \frac{E_k - E_j}{2\hbar} \right) \cos \left( \frac{E_k - E_j}{2\hbar} \right) \frac{\partial}{\partial t} \left( \langle \psi_k | \mathbf{H}'(t) | \psi_j \rangle \right)^2 \] (1.25)
The first term in this formula represents the expansion of the new eigenstate in the original eigenbasis. The second term gives the probability to make a transition out of this state. The final term, because of the sine and cosine, will average out to zero over a long enough time period. Averaging the sine squared term gives the probability of transitioning from state $j$ to state $k$

$$\text{Prob}_{j \rightarrow k} \approx \frac{2\hbar^2}{(E_k - E_j)^4} \left| \langle \psi_k | \frac{\partial H'(t)}{\partial t} | \psi_j \rangle \right|^2. \quad (1.26)$$

Transitions are then prohibited provided that the time variation of the Hamiltonian satisfies

$$\max_{t' \in [0,t]} \left| \langle \psi_k(t') | \frac{\partial H'(t')}{\partial t'} | \psi_j(t') \rangle \right|^2 \ll \frac{\min_{t' \in [0,t]} (E_j(t') - E_k(t'))^4}{4\hbar^2}. \quad (1.27)$$

### 1.2.2 Using Adiabatic Evolution to Perform Computation

All quantum computation works by evolving a system from a prepared state into one from which the answer to the computational problem of interest can be read. Before the algorithm can begin, a Hamiltonian must be constructed that represents the problem to be solved. Imagine that the problem of interest is a problem where an item that uniquely satisfies some criteria must be selected out of a collection of $N$ items. A Hamiltonian would be constructed that operates in an $N$-dimensional Hilbert space. The possible items would then be represented by the eigenvectors of the Hamiltonian which make up a basis for the Hilbert space. In that basis the action of the Hamiltonian must somehow pick out the item of interest. Typically this Hamiltonian, referred to as the problem Hamiltonian $H_p$, is constructed so that solutions to the problem are represented by the state vectors with minimum energy.

Another $N$-dimensional Hamiltonian must also be constructed. This Hamiltonian, referred to as the beginning Hamiltonian $H_b$, should have a ground state that can be easily prepared because it is from that state that the system will begin to evolve. The particulars of the problem may also influence the choice of $H_b$. For instance, when constructing a beginning Hamiltonian to solve a problem where any of $N$ possible
outcomes are apriori equally likely, $H_b$ should be chosen to have a ground state that is an equal superposition of all the eigenvectors of $H_p$.

These two time independent Hamiltonians are then combined to form the Hamiltonian that will evolve the prepared ground state of $H_b$ into the ground state of $H_p$. This is done by creating a time dependent Hamiltonian that interpolates from one to the other

$$H_T(t) = \left( \frac{t}{T} \right) H_p + \left( \frac{T - t}{T} \right) H_b.$$  

(1.28)

The parameter $T$ controls the rate of interpolation. In order for the computation to succeed, the interpolation must be done slowly enough so that there is a large probability that the system remains in its ground state throughout the evolution. To understand how this constrains $T$, rewrite $H_T(t)$ as $H(s)$ where $s \equiv t/T$. Then

$$\frac{d}{ds} H(s) = T \frac{d}{dt} H_T(t).$$  

(1.29)

The most likely transition would be to the first excited state. From section 1.2.1 this means that $T$ must obey the constraint

$$T \gg \frac{\max_{s \in [0,1]} \left| \langle \psi'_1(s) | \frac{dH(s)}{ds} (s) | \psi'_0(s) \rangle \right|}{\min_{s \in [0,1]} \left[ E'_1(s) - E'_0(s) \right]^2}.$$  

(1.30)

In this equation the primed variables refer to the eigenvalues and eigenvector of $H(s)$ which are related to the eigenvectors and eigenvalues of $H_T(t)$ through $E'_i(s) = E_i(sT)$ and $|\psi'_i(s)\rangle = |\psi_i(sT)\rangle$.

1.3 k-SAT

In this thesis, the performance of the adiabatic algorithm is evaluated by solving instances of $k$-SAT. While an enormous amount of work has gone into studying the performance of classical algorithms on these problems, this section will only attempt to provide the reader with enough background to explain how these problems fit into
the conventional computation complexity hierarchy and why the problems are an appropriate choice for the exploration of the performance of the adiabatic algorithm.

1.3.1 SATISFIABILITY

SATISFIABILITY problems are solved by finding an assignment to a given set of binary variables that satisfies a logical formula which is provided when the problem is posed. In all of the cases that will be examined here these logical formulae can be expressed as a logical conjunction (AND) of individual clauses. Each clause in turn is composed of the logical disjunction (OR) of a number of the variables and/or their negations.

In what follows logical OR will be represented by \( \lor \) and logical AND will be represented by \( \land \). The binary variables will be \( x_1 \ldots x_n \) and their negations will be \( \bar{x}_1 \ldots \bar{x}_n \). An example of such a logical formula is

\[
(x_3) \land (\bar{x}_6 \lor x_4) \land (\bar{x}_5 \lor x_2 \lor x_1).
\] (1.31)

The first clause \( (x_3) \) contains a single variable. In order for it to be satisfied \( x_3 \) must have the value 1 (or TRUE). The next clause \( (\bar{x}_6 \lor x_4) \) requires that one or both of the variables \( x_6 \) and \( x_4 \) have the value 0 (or FALSE). The final clause contains three variables. In order for this clause to be true, one (or more) of the following must be true: \( x_5 = 0, x_2 = 1, \) or \( x_1 = 1 \). Since the formula is composed of clauses joined by logical ANDs, all three clauses must be true for the formula itself to be true. In a general SATISFIABILITY problem there is no restriction on the number of clauses or the number of variables that can occur in each clause. If an assignment exists that makes the formula true the formula is said to be satisfiable, otherwise it is said to be unsatisfiable. Algorithms which solve instances of SATISFIABILITY determine whether the given instance is satisfiable.

For an \( n \) bit instance of SATISFIABILITY there are \( 2^n \) possible assignments to the variables. These possible assignments can be enumerated by the binary representation of the integers between 0 and \( 2^n - 1 \) where \( x_1 \) corresponds to the least significant bit and \( x_n \) the most significant bit. Using this enumeration, the solution to an instance
of SATISFIABILITY can be represented as an integer. For example a 3 bit instance that is solved by \( x_1 = 0, \ x_2 = 1, \text{ and } x_3 = 1 \) is solved by 6.

### 1.3.2 2-SAT

2-SAT problems are SATISFIABILITY problems where each clause in the formula contains exactly two variables. The variables may appear negated and variables may repeat in a single clause so long as each clause contains two variables. There is no limit to the number of clauses that may occur in the formula. Two such formulae are:

\[
(x_1 \lor x_2) \land (x_1 \lor \bar{x}_2) \land (\bar{x}_1 \lor x_2) \quad \text{and} \quad (x_1 \lor x_2) \land (x_1 \lor \bar{x}_2) \land (\bar{x}_1 \lor x_2) \land (\bar{x}_1 \lor \bar{x}_2).
\]

The first of these is satisfiable, and the second is not.

There is an \( O(n) \) algorithm that can decide whether any instance of 2-SAT is satisfiable. For SATISFIABILITY problems the size of the problems is the number of bits \( n \) from which the variables in the clauses are drawn. This \( O(n) \) algorithm is a tremendous improvement over checking all possible assignments since the number of possible assignments is \( 2^n \).

The first step in the algorithm consists of constructing a graph with \( 2n \) vertices. Each vertex represents either a variable \( (x_i) \) or the negation \( (\bar{x}_i) \) of a variable. Then for each clause, two directed paths are drawn. For the clause \( (a \lor b) \), one path is drawn from \( \bar{a} \) to \( b \), and the other is drawn from \( \bar{b} \) to \( a \). Figure 1-1 shows the result for the two examples above. If a directed path exists from a variable to the negation of that variables, the variable cannot have the value TRUE. If a directed path exists from the negation of a variable to the variable, the variable cannot have the value FALSE. This is because each directed path represents a logical implication. In the example clause \( (a \lor b) \), if \( a \) has the value FALSE then \( b \) must have the value TRUE. The directed path connecting \( \bar{a} \) and \( b \) represents this implication. A path connecting a variable and that variable’s negation is then a logical contradiction. Since the total number of paths cannot exceed twice the number of clauses, checking each variable will take
no more than $O(nc)$ steps, where $c$ is the number of clauses.

1.3.3 3-SAT

Another subset of the SATISFIABILITY problems is 3-SAT. These problems differ from the 2-SAT problem only in that each clause in the logical formula must contain exactly three variables. A 5 bit example of such a formula is:

$$(x_1 \lor x_3 \lor \overline{x}_5) \land (x_1 \lor \overline{x}_4 \lor x_2) \land (x_2 \lor \overline{x}_1 \lor x_4). \quad (1.34)$$

The addition of the extra variable in each clause has a tremendous impact of the difficulty of these problems. There are no known algorithms that are able to solve them efficiently. For all known algorithms, the amount of time required to solve random instances of 3-SAT increases exponentially with the number of bits involved in the formula when the number of bits is large. 3-SAT problems fall within the computational complexity class NP-COMPLETE.

To say that a problem can be solved in polynomial time mean that the maximum number of steps required to solve the problem can be bounded by a polynomial in the size of the problem. Problems that can be solved in polynomial time on conventional computers are in the class $\mathsf{P}$. Problems that can be solved in polynomial time on a nondeterministic computer are in the class $\mathsf{NP}$ (This defines the class $\mathsf{NP}$). A nondeterministic computer is one that can pursue multiple possibilities simultaneously. To give an example, the possible assignments of $n$ binary variables can be arranged into a binary tree that has depth $n$. Even though this tree has $2^n$ leafs, a nondeterministic
computer would be able to search it in \( O(n) \) steps because it can pursue both branches at any node simultaneously. Such machines do not exist; they are a theoretical construct which complexity theorists have used to quantify the difficulty of certain sets of problems. Equivalently, problems in \( NP \) have the property that individual possible solutions can be checked in polynomial time despite the fact that the total space of possible solutions can be exponentially large. While problems in \( P \) are also in \( NP \), it has not been proven that there are any problems in \( NP \) that cannot be solved in polynomial time on a conventional computer.

Each \( NP\text{-COMPLETE} \) problem has the property that an algorithm that is able to solve it in polynomial time on a conventional computer will also be able to solve all problems in the class \( NP \) in polynomial time. Much work has gone into determining whether such algorithms exist. This effort has been expended because such problems occur frequently in practice, and as more is understood about the class of \( NP\text{-COMPLETE} \) problems, algorithms that solve problems from this class more and more efficiently are produced.

### 1.3.4 (2+p)-SAT

(2+p)-SAT is a set of problems designed to sit in between 2-SAT and 3-SAT. The logical formulae in instances of (2+p)-SAT are generated randomly and can have both clauses containing two variables and clauses containing three variables. Each clause in the formula is chosen by first picking a random number uniformly between zero and one. If this number is less than \( p \), a three variable clause is added to the formula. Otherwise, a two variable clause is added. The variables that occur in the clauses are also typically selected randomly. For \( p = 0 \) the instances that result from this procedure are randomly generated 2-SAT instances, and with \( p = 1 \) the instances are randomly generated 3-SAT instances. For \( p \) between zero and one the formulae usually involve both two variable and three variable clauses.

In order to generate instances of these problems, the number of clauses must somehow be fixed. Previous studies\(^4\) show that for instances of 3-SAT if the number

\(^4\)Cheeseman et al., "Where the really hard problems are" Proceedings of IJCAI-91.
of clauses is fewer than 4.25 times the number of bits, the chance that the instance is satisfiable is high. If the number of clauses is chosen to be greater than 4.25 times number of bits, the chance that the instance is satisfiable is low. Coincident with this transition is a qualitative change in the time required by classical algorithms to decide if these instances are satisfiable. Sets of instances where the number of clauses is roughly 4.25 times the number of bits show an exponential increase in the amount of time required to solve typical instances as the number of bits increases. Away from this region the amount of time required to solve a typical instance increased less than exponentially with the number of bits. The study mentioned below fixes the number of clauses by requiring the largest number of clauses at any bit number that yields a set of instances where half of the instances are satisfiable. Our studies fix this number in a different way which will be explained in section 2.1.

For $p > 0$ these problems form a set of problems that is technically contained within the NP-COMPLETE set of problems. However, these problems have recently received a lot of attention because for certain ranges of $p$, typical problems are not difficult to solve.\footnote{Monasson et al., "Phase Transition and Search cost in the 2+-p-SAT Problem" 1996 Phys Comp96.} For $p < 0.4$ the computational cost has been seen to increases only nearly linearly with $p$. It is not until $p > 0.6$ that the cost begins to rise rapidly.

These results are important for two reasons. First, they show that complexity classification, which is done on a worst case basis, is not always a good indication of the difficulty of typical problems drawn from a set. Second, the rapid transition from marginal ease to marginal difficulty as $p$ moves from 0.4 to 0.6 provides the designer of new algorithms a perfect place to test the performance of those algorithms.
Chapter 2

Details

This chapter focuses on the details of the implementation of the adiabatic algorithm used to solve random instances of \((2+p)\text{-SAT}\). First the method of \((2+p)\text{-SAT}\) instance generation is discussed. Then the construction of the Hamiltonians used by the adiabatic algorithm in the solution of theses instances is described.

2.1 Instance Generation

This work explores the performance of the adiabatic algorithm on random instances of \((2+p)\text{-SAT}\) problems. We are interested in the typical performance of the algorithm, rather than the worst case performance, but we have also tried to ensure that the set of problems we choose to work with is the most difficult set that we can generate randomly. The biggest effect of this choice was to restrict our attention to instances that are satisfiable by exactly one assignment to the variables. These are amongst the problems that are the most difficult for classical algorithms. If there are many satisfying assignments, one can be found quickly by random search. If an instance is strongly over constrained, classical algorithms can also quickly see that there is no solution consistent with all the constraints. If the number of constraints is not large enough to allow the space of possible solutions to be searched quickly, the algorithms require large amounts of time to perform the search. The performance is worst when a single solution exists, but very few clauses help in directing the search. From
the studies discussed in section 1.3.4 we also know that classical algorithms typically have the most difficulty with problems that have a number of clauses near the number that signals the transition from typical problems being satisfiable to typical problems being unsatisfiable. In order to generate instances that have this property, we generate instances by randomly generating clauses until all but one satisfying assignment is eliminated. If the addition of a clause eliminates all the satisfying assignments, that instance is thrown out, and the generation process starts over.

We also restrict the clauses that make up the logical formulae to clauses that do not repeat a variable. This means that if $x_2$ appears as one of the variables in a clause, neither $x_2$ nor $\overline{x}_2$ can occur anywhere else in that clause. This constraint simply prevents the problems from becoming artificially easier than they normally would be.

The generation of an $n$ bit instance of $(2+p)$-SAT begins with the generation of the first clause. To generate a clause, first a random variable is chosen uniformly from the interval $[0, 1]$. If this number is less than $p$ a 3-SAT clause is generated as described below. Otherwise a 2-SAT clause is generated as described below. The variables that make up a clause are randomly chosen one at a time from the list $\{x_1, x_2, \ldots, x_n\}$. If a variable is chosen that has already been selected for this clause, that choice is rejected. If a 3-SAT clause is being generated three unique variables are selected. If a 2-SAT clauses is being generated, two unique variables are selected. Once the variables are selected, they are individually negated with 50% probability. All the clauses are constructed using this procedure. When a clause has been constructed it is added to the formula. The number of satisfying assignments is then counted. If the number of satisfying assignments is 1 the instance is accepted. If no satisfying assignments are found, all the clauses are thrown out and the entire process starts over. If more than 1 satisfying assignment is found, another clause is generated.

The process of generating an $n$ bit instance is captured by the following algorithm:

1: let S be the number of satisfying assignments
2: set $S=2^n$
3: clear the logical formula
4: while S>1 do
5:  choose \( x \) uniformly from the interval \([0,1]\)
6:  if \( x > p \) then
7:    generate a 2-SAT clause
8:  else
9:    generate a 3-SAT clause
10:  end if
11:  add clause to formula
12:  search for satisfying assignments
13:  set S to the number of satisfying assignments found
14: end while
15: if S=1 then
16:  accept instance
17: else
18:  RESTART
19: end if

Since each 2-SAT clause accepts three of the possible four assignments to the variables contained in it and each 3-SAT clause accepts seven out of the possible eight assignments to the variables contained in it, the expected number of clauses, \( c(p) \), can be estimated

\[
\langle c(p) \rangle = -n \left[ \frac{1-p}{\log_2(3/4)} + \frac{p}{\log_2(7/8)} \right] \approx n \left[ 2.4(1-p) + 5.2(p) \right].
\] (2.1)

As can be seen in figure 2-1, equation 2.1 overestimates the number of clauses, but accurately captures the fact that the expected number of clauses grows linearly with \( n \) for all values of \( p \).
2.2 Construction of the Hamiltonians

As discussed in section 1.2.2 in order to use the adiabatic algorithm to solve problems, the two Hamiltonians that are interpolated between must be constructed. As will be discussed in section 2.3 the choices made in constructing these operators can greatly influence the performance of the algorithm. Throughout this chapter, and for the rest of this thesis, the work will be presented in units such that $\hbar = 1$.

2.2.1 The Problem Hamiltonian $H_p$

An n bit instance of $(2+p)$-SAT has $2^n \equiv N$ possible assignments of the variables. The instance will then require an N dimensional Hilbert space if all of the possible assignments of the variables are to be mapped onto linearly independent vectors. This space will be constructed from the tensor product space of n qubits. A basis for this

\[ f(x) = 0.11 + 2.16x \]
\[ g(x) = 0.29 + 2.86x \]
\[ h(x) = -1.3e-12 + 4.5x \]

Figure 2-1: Number of Clauses vs. Number of Bits for 3 Values of $p$
space can be found by taking all the products of the form

\[
|z_1\rangle |z_2\rangle |z_3\rangle \ldots |z_n\rangle
\]

where each of the \(z_i\)'s is 0 for a spin pointing upward in the \(\hat{z}\) direction and 1 for a spin pointing downward in the \(\hat{z}\) direction. For a single bit then,

\[
|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.
\]  

(2.3)

This numbering allows these vectors to be labeled numerically starting with zero, \(\{|0\rangle, |1\rangle, \ldots, |N - 1\rangle\}\). This number is chosen so that the integer label for each vector, when written in binary, gives the \(\hat{z}\) direction of each spin and one of the \(N\) possible assignments to the variables.

The problem Hamiltonian is then constructed as a sum of terms. Each term corresponds to one clause in the logical formula associated with the \((2+p)-\text{SAT}\) instance. Each one of these terms is diagonal in this basis. The action of the term is to return a 1 if the particular assignment associated with the vector being operated on violates that clause and a 0 if it does not.

Then, the action of \(H_p\) as a whole on the vectors in this basis is to return the vector multiplied by the number of clauses violated by the assignment represented by that vector. \(H_p\) is diagonal in this basis and, since the instances are generated with unique satisfying assignments, has a unique state with energy 0. Further, all the other states have energy at least 1. Since a typical assignment to the variables violates 1/8 of the 3-SAT clauses and 1/4 of the 2-SAT clauses, the energy of a typical state will be of the same order as the average number of clauses. This number is given by equation 2.1.
2.2.2 The Beginning Hamiltonian : $H_b$

The beginning Hamiltonian is also composed of a sum of terms. Each term $H_b^i$ acts on a single bit and takes the form

$$H_b^i = \frac{1}{2}(1 - \sigma_x^i) \quad \text{with} \quad \sigma_x^i = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \quad (2.4)$$

Then

$$H_b^i \ket{x_i = x} = x \ket{x_i = x} \quad (2.5)$$

where

$$\ket{x_i = 0} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad \ket{x_i = 1} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \quad (2.6)$$

Each time a variable occurs in each of the clauses making up the logical formula a term of this type is added to $H_b$. For example, the 3-SAT formula

$$(x_1 \lor x_3 \lor \bar{x}_5) \land (x_1 \lor \bar{x}_4 \lor x_2) \land (x_2 \lor \bar{x}_1 \lor x_4) \quad (2.7)$$

produces a beginning Hamiltonian

$$H_b = 3H_1^b + 2H_2^b + 3H_3^b + 2H_4^b + 5H_5^b. \quad (2.8)$$

The symbol $x_i$ used to denote the value of the $i^{th}$ bit in a logical formula is not related to the symbol $\ket{x_i = x}$ which denotes that the $i^{th}$ qubit is in an eigenstate of the operator $\sigma_x^i$ with eigenvalue $x$.

For every instance with a unique satisfying assignment, the ground state of $H_b$ is the tensor product of all the bits in the state $\ket{x_i = 0}$. Because this is an equal superposition of a single bit pointing up and down, the ground state corresponds to an equal superposition of all the basis vectors in the eigenbasis of the problem.
Hamiltonian.\textsuperscript{2}

2.3 Previous Work

In addition to the paper where the idea of adiabatic quantum computation was originally presented\textsuperscript{3} two numerical studies have been performed to evaluate the performance of the algorithm on instances of NP-COMPLETE problems. Both studies draw problems from a set named EXACT COVER. The problems are a subset of the SATISFIABILITY problems discussed earlier, but the clauses that are conjoined to form the logical formulae are of a slightly different type than they are in k-SAT problems. However, instances were generated with the same constraints presented above, and the beginning and problem Hamiltonians were constructed similarly.

The first of these\textsuperscript{4} studied the performance on instances of EXACT COVER 2 (EC2) and EXACT COVER 3 (EC3). Much like 2-SAT and 3-SAT, the first of these sets can be solved classically in polynomial time while the second cannot. Farhi et al. also explored the performance of the algorithm when the diagonal elements of $H_p$ were randomly reordered. This scrambling destroys any connection between $H_b$ and $H_p$. Once this has happened the problem is much more difficult to solve, and even the quantum algorithm is known to require an exponentially increasing amount of time to solve these problems as their size grows.

The performance on instances of EC3 and EC2 was explored by generating 50 instances at each bit number from 7 to 15. For each instance the amount of time ($T$ in equation 1.28) required to remain in the ground state (and hence find the solution to the problem) with probability 1/8 was determined by numerically integrating Schrödinger’s equation. The median times for the set of 50 instances were then used to quantify the performance.

As can be seen in figure 2-2 the amount of time required to solve the instances of EC2 seems to only increase linearly with the number of bits. The time required to

\textsuperscript{2}A discussion of the numerical implementation of $H_b$ is given in appendix A.

\textsuperscript{3}quant-ph/0001106.

\textsuperscript{4}quant-ph/0007071.
Figure 2-2: Performance of the Adiabatic Algorithm on EC2 and EC3 (Figures taken from [FGG00])
Figure 2-3: Performance of the Adiabatic Algorithm on Scrambled Instances of EC2 (Figures taken from [FGG00])

solve the EC3 instance is well fit by a quadratic, but is also well fit by an exponential.

The effect of scrambling was studied by determining the time required to remain in the ground state with probability 1/8 after scrambling $H_p$ for 100 instances at each bit number from 7 to 12. The results are shown in figure 2-3. As expected, the behavior is clearly exponential in this case.

Their next study$^5$ extended the results for EC3 out to 20 bits solving 100 instances at each bit number. The results are very consistent with those found in the previous study. In fact, as can be see in figure 2-4 the quadratic fit found in the initial study continues to fit the data out to 20 bits.

$^5$quant-ph/0104129.
Figure 2-4: Performance of the Adiabatic Algorithm out to 20 Bits (Figures taken from [FGG+01])
Chapter 3

Integration of Schrödinger’s Equation

Continuing in the same vein as the previous investigations on this topic, this work began by attempting to numerically integrate Schrödinger’s equation to determine at what rate the time parameter $T$ in adiabatic evolution has to be increased as the number of bits increases in order to obtain a fixed probability of ending up in the ground state at the end of the evolution. The determination of the parameter $T$ is a direct measure of something that would be needed to use the algorithm in practice. These algorithms are inherently probabilistic, and it is likely that they will have to be used several times to guarantee that they have produced the correct result. But in order to run the algorithm even once the interpolation time $T$ must be chosen. This time could be chosen by evaluating the performance of the algorithm on a set of benchmark problems. This evaluation would proceed in the same fashion as the following investigation proceeds. The difference, of course, is that the real benchmarks would be determined by running a real quantum computer, not a classical computer program attempting to simulate the behavior of a quantum computer.
3.1 Details of the Integration Search Strategy

For each instance that was generated, the minimum time to interpolate from $H_b$ to $H_p$ that allowed the state to remain in the ground state with 50% probability was determined. This was done by first integrating\(^1\) for two times that are much lower than the expected T, around 5% and 10% of the running median value for T, and determining the probability to be in the ground state. From these two values, the next trial running time was chosen by linearly extrapolating these results. This process of extrapolation continued until the probability to remain in the ground state was 50%.

To avoid over extrapolation, the extrapolated time was never allowed to change by more than half of the current estimate of the median value of T. For some instances an extrapolation from even small values of T could predict the correct value of T required to end in the ground state with 50% probability; however as is shown in figure 3-1 this extrapolation was not always accurate.

3.2 Results

3.2.1 2-SAT and 3-SAT

The performance of the adiabatic algorithm on the instances of k-SAT studied here is very similar to its performance on instances of Exact Cover. The behavior of the required time to achieve probability 50% for instances of 2-SAT is well fit by a straight line (figure 3-2). The data taken in this evaluation are summarized in table 3.1. The error column in the table is the sampling error on the median runtime. For instances of 3-SAT with bit numbers below 10, the median required runtime to achieve probability 50% appears linear (see figure 3-2), but the performance at higher bit number confirms that the interpolation time T grows faster than linearly with the number of bits.

\(^1\)While many different methods of integration were tried, nothing performed consistently better than the 4/5 Cash Karp Runge-Kutta stepper presented in "Numerical Recipes." The routines presented there had to be modified to handle complex numbers, and the standard step size adjuster was replaced by one that was more optimized for this problem.
The performance was also examined on instances of \((2+p)\text{-SAT}\) for \(p = .5\). For this value of \(p\), the amount of time required by classical algorithms to solve instances increases exponentially. The performance of adiabatic algorithm appears to be growing just faster than linearly at this value of \(p\). Classically the transition from easy to hard with \(p\) can only be seen at much higher bit numbers than we were able to explore, so this near linear behavior could be an artifact of working at too low a bit number to see the algorithm's ultimate performance.

The distribution of probabilities that results from running the algorithm on a large set of instances for the predetermined time required to reach median probability 50\% was examined (see appendix C for details). The distribution is a lot less tight than was found in previous studies (see figure 3-3). The distribution of probabilities that resulted from 1000 instances at 10 bits contained probabilities that were as large as 0.98 and as small as 0.005.
<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Number of Points</th>
<th>2-SAT Median Time</th>
<th>2-SAT Error</th>
<th>3-SAT Median Time</th>
<th>3-SAT Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1000</td>
<td>5.35</td>
<td>0.24</td>
<td>12.6</td>
<td>0.77</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>8.22</td>
<td>0.48</td>
<td>20.6</td>
<td>1.4</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>10.4</td>
<td>0.7</td>
<td>28.1</td>
<td>1.8</td>
</tr>
<tr>
<td>6</td>
<td>1000</td>
<td>12.6</td>
<td>1.0</td>
<td>35.4</td>
<td>1.2</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>14.6</td>
<td>0.8</td>
<td>46.5</td>
<td>3.2</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>16.0</td>
<td>1.3</td>
<td>55.0</td>
<td>3.6</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>18.6</td>
<td>0.6</td>
<td>61.9</td>
<td>5.7</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>20.8</td>
<td>1.7</td>
<td>71.4</td>
<td>8.1</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
<td>22.2</td>
<td>1.9</td>
<td>82.1</td>
<td>7.4</td>
</tr>
<tr>
<td>12</td>
<td>1000</td>
<td>24.9</td>
<td>2.3</td>
<td>94.6</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>1000</td>
<td>25.5</td>
<td>3.1</td>
<td>107</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>1000</td>
<td>28.3</td>
<td>2.4</td>
<td>117</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 3.1: 2-SAT and 3-SAT Data Summary

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Number of Points</th>
<th>2-SAT Median Time</th>
<th>2-SAT Error</th>
<th>3-SAT Median Time</th>
<th>3-SAT Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1000</td>
<td>6.23</td>
<td>0.38</td>
<td>12.5</td>
<td>0.52</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>10.6</td>
<td>0.51</td>
<td>22.5</td>
<td>1.2</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>16.6</td>
<td>1.4</td>
<td>36.3</td>
<td>3.2</td>
</tr>
<tr>
<td>6</td>
<td>1000</td>
<td>27.4</td>
<td>2.7</td>
<td>57.3</td>
<td>6.8</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>50.5</td>
<td>7.4</td>
<td>97.4</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>85.8</td>
<td>9.8</td>
<td>164</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>168</td>
<td>19</td>
<td>311</td>
<td>42</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>284</td>
<td>36</td>
<td>485</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 3.2: Scrambled 2-SAT and Scrambled 3-SAT Data Summary
3.2.2 Scrambled Instances

Scrambled 3-SAT instances are instances that are constructed in the following way: first an instance of 3-SAT is generated, then $H_b$ and $H_p$ are created, lastly the elements are $H_p$ are randomly reordered. As mentioned in section 2.3, even a quantum algorithm will require an exponential amount of time to solve these instances. Figure 3-2 shows that such behavior is evident in both scrambled 3-SAT instances and scrambled 2-SAT instances.

3.3 Limitations

In order to find the time required to achieve probability $1/2$ for a single 14 bit instance in cases where that time turned out to be near the final median, about one hour of computer time was required on a Pentium III 1.7 GHz machine. However, this is not a good estimation of the rate at which the times were found. Some instances required times that were much larger than the median time. For 14 bit instances of 3-SAT the median runtime to achieve probability $1/2$ was found to be 117. In a sample of one thousand 14 bit instances over 100 instances required the adiabatic interpolation time to be greater than 2,000 and several required it to be over 800,000. Substantially more computer time was required to integrate these instances.
Figure 3-2: Increase in Required Runtime as a Function of Bit Number for 2-SAT and 3-SAT (top) and Scrambled Instances of 2-SAT and 3-SAT (bottom)
Distribution of Probabilities at the Median Run Time for 1000 Instances at 10 Bits

Figure 3-3: Distribution of Probabilities for 1000 Instances at 10 Bits
Chapter 4

Minimum Energy Gap

In order to extend the study of the performance of the adiabatic algorithm to higher bit numbers it is necessary to find a method of evaluating the performance that is less computationally demanding than the ones discussed in the previous chapter. Using the results of the adiabatic theorem, such an evaluation can be made by finding the minimum energy gap between the ground state and the first excited state during the interpolation. In our experience, the minimum gap can be located in one tenth of the time required to find $T$ and requires half of the memory.

4.1 Running Times from the Adiabatic Theorem

In section 1.2.2, formula 1.30 states that the interpolation time required for successful interpolation from $H_b$ to $H_p$ is bounded by

$$T \gg 2\hbar \frac{\max_{s \in [0,1]} \left| \left\langle \psi_1'(s) \, \frac{\partial H(s)}{\partial s} \, \psi_0'(s) \right\rangle \right|}{\min_{s \in [0,1]} \left[ E_1'(s) - E_0'(s) \right]^2}.$$  \hfill (4.1)

The method presented in the previous chapter attempted to determine $T$ by integrating the evolution of the state. However, if the right hand side of this equation is a fair substitute for $T$, it can be found simply by determining the eigenvectors and eigenvalues of the Hamiltonian during the interpolation. Figure 4-1 shows a comparison between the running time $T$ as found by the methods discussed in the previous
Figure 4-1: Does the Adiabatic Formula Determine the Required Running Time?

\[
\left| \frac{\langle \psi'_1(s_{min}) | \frac{dH(s)}{ds} | \psi'_0(s_{min}) \rangle}{|E'_1(s_{min}) - E'_0(s_{min})|^2} \right| \tag{4.2}
\]

where \( s_{min} \) is the location of the minimum energy gap. For each point, an instance (either a 10 bit or 12 bit instance) is generated and the time required to achieve probability 1/2 is found by integration. Then the location where \( (E'_1(s) - E'_0(s)) \) is minimum is found. At this point the matrix element in the numerator is also evaluated. As seen in the figure 4-1, the time found by direct integration and the value of expression 4.2 are highly correlated.

The reason that this formula is so accurate despite the fact that the numerator is was not maximized is that \( (E'_1(s) - E'_0(s)) \) is typically small in one narrow region of the interpolation. This can be seen in figure 4-2, which shows the energy gap as a function of \( s \) for a typical 10 bit instance of 3-SAT. Since the gap is small in only
one place, it is most likely that a transition will occur in this region if one is going to occur at all. The transition probability is then governed by the gap and the matrix element in this region. Because there is such a high degree of correlation between the integration time and the time found from expression 4.2, expression 4.2 can be used to evaluate the performance of the algorithm.

In addition to being less computationally demanding, there is an additional reason to use the energy gaps to evaluate the performance of the algorithm. The matrix elements of the Hamiltonian grow as a polynomial with the number of bits. Numerically we work in the basis provided by $H_p$. In this basis $H_p$ is diagonal, and since the diagonal elements represent the number of clauses violated by a particular assignment of the bits in the logical formula, they are approximately the number of clauses divided by 8 for 3-SAT (or divided by 4 for 2-SAT). Similarly, typical matrix elements of $H_b$ are approximately the number of clauses in the logical formula from which it was created. Since the number of clauses only grows linearly with $n$, the matrix elements
of $H(s)$ are bounded by a polynomial in $n$. The most important thing to determine about the running times is whether they grow exponentially, and since matrix elements of $H(s)$ cannot cause this behavior, if it happens it must happen because the energy gaps become exponentially small.

### 4.2 Details of the Eigenvalue Search Strategy

This section discusses the method of finding the location of the minimum gap. A discussion of the numerical methods used in this search can be found in appendix B. In what follows, the function $g(s)$ refers to the energy gap between the ground state and the first excited state as a function of the interpolating variable $s$. Specifically,

$$g(s) = \langle \psi'_1(s) | H(s) | \psi'_1(s) \rangle - \langle \psi'_0(s) | H(s) | \psi'_0(s) \rangle.$$  \hspace{1cm} (4.3)

The derivative of $g(s)$ is given by

$$\frac{dg}{ds}(s) = \langle \psi'_1(s) | \frac{dH(s)}{ds} | \psi'_1(s) \rangle - \langle \psi'_0(s) | \frac{dH(s)}{ds} | \psi'_0(s) \rangle \hspace{1cm} (4.4)$$

$$= \langle \psi'_1(s) | (H_p - H_b) | \psi'_1(s) \rangle - \langle \psi'_0(s) | (H_p - H_b) | \psi'_0(s) \rangle. \hspace{1cm} (4.5)$$

A single evaluation of $g$ also determines, with little extra numerical work, whether that value of $s$ is to the left or right of the minimum. Using standard techniques for one dimensional minimization we are typically able to find the minimum to within a tolerance of $10^{-5}$ with 9 evaluations of $g$.

Without an exhaustive search on every instance it is impossible to determine if the located minimum is the global minimum. This could potentially be a problem because instances have been found that have multiple local minima. In our experience, they occur about 1% of the time. One such instance is shown in figure 4-3. To examine the effect of these instances we generated 1000 instances at 10 bits and determined the minimum gap by the technique described above and also by an exhaustive search. For the instances that were determined to have double minimum our minimizer found the true minimum over 80% of the time. Even in those instances where the true minimum
Figure 4-3: An Example of a 10 Bit Instance with a Double Minimum

was not found, all these instances ended up being placed on the correct side of the median. Since the performance of the algorithm is evaluated by determining the median minimum gap for a large number of instances at each bit number, the presence of the double minimum was ignored in further studies.

4.3 Results

4.3.1 2–SAT and 3–SAT

Figure 4-4 shows the median gaps for sets of 2–SAT and 3–SAT. For 2–SAT, sets of instances at bit numbers 10 to 18 were examined, and for 3–SAT, sets of instances at bit numbers 10 to 23 were examined. Figure 4-5 also shows this data on a $\log_2$/linear scale and a $\log_2$/log$_2$ scale. The data is summarized in table 4.1.

The most interesting question that can be asked about the behavior of the median minimum gap is whether its rate of decrease is polynomial in $n$ or whether it is
Figure 4-4: Data from Table 4.1 on a linear/linear scale

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Number of Points</th>
<th>2-SAT Median Minimum Gap</th>
<th>2-SAT Error</th>
<th>Number of Points</th>
<th>3-SAT Median Minimum Gap</th>
<th>3-SAT Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1000</td>
<td>0.414</td>
<td>0.032</td>
<td>3700</td>
<td>0.349</td>
<td>0.014</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
<td>0.407</td>
<td>0.028</td>
<td>3700</td>
<td>0.327</td>
<td>0.019</td>
</tr>
<tr>
<td>12</td>
<td>1000</td>
<td>0.391</td>
<td>0.022</td>
<td>3700</td>
<td>0.287</td>
<td>0.014</td>
</tr>
<tr>
<td>13</td>
<td>1000</td>
<td>0.368</td>
<td>0.022</td>
<td>3700</td>
<td>0.275</td>
<td>0.018</td>
</tr>
<tr>
<td>14</td>
<td>1000</td>
<td>0.357</td>
<td>0.026</td>
<td>3700</td>
<td>0.237</td>
<td>0.015</td>
</tr>
<tr>
<td>15</td>
<td>1000</td>
<td>0.362</td>
<td>0.029</td>
<td>3700</td>
<td>0.229</td>
<td>0.015</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>0.344</td>
<td>0.030</td>
<td>3700</td>
<td>0.210</td>
<td>0.015</td>
</tr>
<tr>
<td>17</td>
<td>1000</td>
<td>0.327</td>
<td>0.027</td>
<td>3700</td>
<td>0.191</td>
<td>0.014</td>
</tr>
<tr>
<td>18</td>
<td>1000</td>
<td>0.329</td>
<td>0.026</td>
<td>2008</td>
<td>0.176</td>
<td>0.017</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td>1626</td>
<td>0.171</td>
<td>0.021</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>931</td>
<td>0.156</td>
<td>0.032</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td>1069</td>
<td>0.133</td>
<td>0.022</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td>529</td>
<td>0.128</td>
<td>0.034</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>268</td>
<td>0.126</td>
<td>0.048</td>
</tr>
</tbody>
</table>

Table 4.1: 2-SAT and 3-SAT Minimum Energy Gap Data
Figure 4-5: Data from Table 4.1 on a linear/$\log_2$ Scale (top) and a $\log_2 / \log_2$ Scale (bottom)
<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Number of Points</th>
<th>2-SAT Median Minimum Gap</th>
<th>2-SAT Error</th>
<th>Number of Points</th>
<th>3-SAT Median Minimum Gap</th>
<th>3-SAT Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>0.119</td>
<td>0.041</td>
<td>200</td>
<td>0.168</td>
<td>0.027</td>
</tr>
<tr>
<td>11</td>
<td>50</td>
<td>0.107</td>
<td>0.031</td>
<td>200</td>
<td>0.137</td>
<td>0.019</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>0.0916</td>
<td>0.015</td>
<td>200</td>
<td>0.111</td>
<td>0.12</td>
</tr>
<tr>
<td>13</td>
<td>50</td>
<td>0.0668</td>
<td>0.072</td>
<td>200</td>
<td>0.0945</td>
<td>0.010</td>
</tr>
<tr>
<td>14</td>
<td>50</td>
<td>0.0642</td>
<td>0.013</td>
<td>200</td>
<td>0.0774</td>
<td>0.0085</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>0.0511</td>
<td>0.0066</td>
<td>200</td>
<td>0.0613</td>
<td>0.046</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>0.0406</td>
<td>0.0077</td>
<td>200</td>
<td>0.0492</td>
<td>0.0039</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>0.0307</td>
<td>0.0047</td>
<td>200</td>
<td>0.0377</td>
<td>0.0022</td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>0.0254</td>
<td>0.0061</td>
<td>200</td>
<td>0.0288</td>
<td>0.0021</td>
</tr>
<tr>
<td>19</td>
<td>50</td>
<td>0.0188</td>
<td>0.0023</td>
<td>200</td>
<td>0.0225</td>
<td>0.0013</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>0.0130</td>
<td>0.0022</td>
<td>200</td>
<td>0.0165</td>
<td>0.00080</td>
</tr>
<tr>
<td>21</td>
<td>164</td>
<td>0.0124</td>
<td>0.00050</td>
<td>30</td>
<td>0.00960</td>
<td>0.0011</td>
</tr>
</tbody>
</table>

Table 4.2: Scrambled 2-SAT and Scrambled 3-SAT Minimum Energy Gap Data

The median minimum gaps are smaller for 3-SAT instances than they are for 2-SAT instances. This is expected since classically 3-SAT is a more difficult than 2-SAT. However, the data do not give a good indication of the asymptotic behavior of the median minimum gap for large n. In fact, the 2-SAT data also appear linear on the log2/linear plot which makes it impossible to rule out exponential behavior even in the case of a classically simple problem.

### 4.3.2 Scrambled Instances

The data for the scrambled instances is summarized in table 4.2. The distribution of minimum energy gaps is much tighter for scrambled instances than it is for unscrambled instances, so a much smaller sample is required at each bit number to achieve a similar error.

Since scrambling the diagonal elements of \( H_p \) destroys any structure that may be present in the problems in virtue of the clauses, even a quantum algorithm will require exponential time to solve this problem. Looking at the figure 4-7, such behavior is evident. For the bits we examined, the median minimum gap is decreasing faster than
exponentially since the data points appear to bow downward around the exponential fit.

The search bound for Grover's algorithm referred to in section 1.1.2 covers all quantum mechanical algorithms attempting to solve the unstructured search problem. Since any quantum algorithm must require a running time that grows like $\sqrt{N}$, the median minimum gap must be decreasing faster than $(1/2)^{n/2}$. As can be seen by examining the slope of the fit to the scrambled data in figures 4-7 and 4-8, the best exponential fit to the data has the median minimum gap decreasing more slowly than this. This is a clear sign that the asymptotic behavior of the algorithm is not seen.

This slope is decreasing, and by fitting subsets of the data the progression of this slope can be seen. The results of fitting subsets of the data are shown in figure 4-9. The first subset contains the data from 10 bits to 12 bits. The second subset contains the data from 11 bits to 13 bits. This grouping is continued until the final subset which contains the data from 20 bits through 22 bits. From the figure, it appears that the asymptotic performance of the algorithm will not be observable before at least 25 or 30 bits.
Median Minimum Gap vs. Bits for 3SAT and Scrambled 3SAT

Figure 4-6: Data from Table 4.2
Figure 4-7: Data for Bits 10-23 from table 4.2. \( \log_2(\text{Minimum Gap}) \) vs Number of Bits and \( \log_2(\text{Minimum Gap}) \) vs \( \log_2(\text{Number of Bits}) \)
Figure 4-8: Data for Bits 15-23 from table 4.2. \( \log_2(\text{Minimum Gap}) \) vs Number of Bits and \( \log_2(\text{Minimum Gap}) \) vs \( \log_2(\text{Number of Bits}) \)
Figure 4-9: Progression of the Slope of the Scrambled Data
Chapter 5

Conclusions

This work evaluated the performance of the quantum adiabatic algorithm in its ability to solve random instances of $k$-SAT in two ways. The first method of evaluation determined how much time was required to solve instances of $k$-SAT by directly integrating the equations governing the dynamics of the system. The method was also used to examine the performance of the algorithm on scrambled instances of $k$-SAT where it is known that the algorithm’s running time must grow exponentially. Using this method we investigated the performance on problems involving up to 14 bits. In the case of 2-SAT, a classically trivial problem, only linear growth in the running time of the algorithm is seen out to 14 bits. On instances of 3-SAT, a classically difficult problem, the amount of time required by the algorithm grows faster than linearly, but is well described by a quadratic for instances of this size. On the scrambled instances of 2-SAT and 3-SAT out to 10 bits, exponential growth is clearly apparent.

The width of the distribution of running times increases with the number of bits. This requires that a larger sample of instances be investigated at each bit number in order to keep the error of the estimate on the required running time fixed. This effect, together with the increase in computer time required to simulate the dynamics of larger and larger systems, makes it impractical to continue the study with these numerical tools.

To extend the study further a second method of evaluating the performance of the adiabatic algorithm was developed. This method of evaluation searches for the
minimum energy gap between the ground state and the first excited state. It is a consequence of the adiabatic theorem that for the problems investigated here this energy gap controls the asymptotic behavior of the algorithm. Because this energy gap is a non-dynamical feature of the system, the amount of time required to compute it does not grow with the required running time of the adiabatic algorithm T. By using routines based on the Lanczos method of determining a small number of the eigenvalues of a large sparse matrix, we investigated systems containing up to 23 bits. The same classes of problems were investigated with this method as was mentioned above.

The scaling of the gap, and hence the running time of the algorithm, for instances of 2-SAT and 3-SAT is well described either by an exponential or by a polynomial. This makes it impossible to draw any conclusions about the ultimate performance of the algorithm on these types of problems. Further, this study makes it clear that the polynomial growth seen in previous studies is an artifact of a matrix element that can be bounded by a polynomial and is not indicative of the asymptotic growth rate of the running time of the algorithm.

Faster than exponential behavior is seen in the performance of the algorithm on scrambled instances of 2-SAT and 3-SAT. There is a strict bound on the number of steps required by a quantum computer to solve these problems. From this bound we see that the asymptotic performance of the algorithm cannot yet be seen. By looking at the rate at which the exponential is growing, the asymptotic behavior will not be apparent before at least 25 or 30 bits. Extrapolating the amount of computer time used to process instances out to 30 bits, it will require the use of at least a teraflop machine to process one 30 bit instance per day. This will make further numerical investigation, without analytic advances, fruitless.
Appendix A

Numerical Implementation of $H_b$

Memory limitations prohibit the storage of matrices much larger than those that control the dynamics of 15 qubit systems in non-sparse form. A dense matrix of single precision floating point numbers from such a system requires over 4 gigabytes of storage. For the problems studied here $H_p$ poses no such problems because it is diagonal. $H_b$, however, is not, although it is fairly sparse. In the $H_p$ basis, $H_b$ couples each state to only $n$ other states where $n$ is the number of qubits. This can reduce storage costs substantially. Just over 1 gigabyte is required to store the nonzero elements of $H_b$ in a conventional sparse form for a 22 qubit system.

This memory constraint is avoided by recomputing the action of $H_b$ each time it is applied. As discussed in section 2.2.2, $H_b$ is a sum of terms, one coming from each bit

$$H_b = \sum_{i=1}^{n} b_i \frac{1}{2}(1 - \sigma_z^i).$$  \hspace{1cm} (A.1)

Here $b_i$ is the total number of times that the variable $x_i$ occurs in the logical formula. The off diagonal action of each term then is just the action of $\sigma_z^i$ which flips the $i^{th}$ bit in the label of the state. For example, consider the beginning Hamiltonian generated by the problem discussed in section 2.2.2

$$H_b = 3H_b^1 + 2H_b^2 + H_b^3 + 2H_b^4 + H_b^5.$$  \hspace{1cm} (A.2)
The action of the operator on the state with label 0 is

\[
H_b |0\rangle = H_b (|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle) = \frac{8}{2} |0\rangle - \frac{3}{2} |1\rangle - \frac{2}{2} |2\rangle - \frac{1}{2} |4\rangle - \frac{2}{2} |8\rangle - \frac{1}{2} |16\rangle ,
\]

(A.3)

and on the state with label 2 is

\[
H_b |2\rangle = H_b (|0\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle \otimes |0\rangle) = \frac{8}{2} |2\rangle - \frac{3}{2} |3\rangle - \frac{2}{2} |6\rangle - \frac{1}{2} |10\rangle - \frac{2}{2} |18\rangle .
\]

(A.6)

These operations, since they are essentially bitwise XOR acting on the label of the vector, can be applied very efficiently. And since only the \(b_i\)'s must be stored, the storage cost shrinks to almost nothing.
Appendix B

The Lanczos Method for Finding Eigenvalues of Sparse Matrices

Just as memory constraints prevent the storage of Hamiltonians with many more than 15 bits, they also constrain the calculation of eigenvalues through the standard QR algorithm.\textsuperscript{1} And even if the matrix can be initially stored in a sparse form, the matrix will not remain sparse throughout the algorithm. The QR algorithm diagonalizes a matrix through repeated application of orthogonal matrices designed to rotate the initial matrix into its diagonal form. Intermediate stages of this process will certainly destroy any sparsity that might be present in the original matrix.

In order to continue to take advantage of sparse structure, an eigenvalue finding method is required that involves only the operation of multiplying a vector by the matrix that is being diagonalized. For real symmetric matrices the Lanczos method\textsuperscript{2} does just this.

The operation of this algorithm is similar to the operation of another numerical method that is used to find the largest eigenvector of a system called the power method. In the power method a random vector is chosen and is repeatedly multiplied by the input matrix. To see the effect of this, assume the matrix $A$ is the matrix whose largest eigenvalue is being sought. The expansion of the initial vector $|v\rangle$ in

\footnotesize
\textsuperscript{1}Numerical Recipes section 11.6.
\textsuperscript{2}Parlett, The Symmetric Eigenvalue Problem, Series in Computational Mathematics, Prentice-Hall 1980
the eigenbasis of $A$ is

$$
A |v\rangle = A \left( \lambda_1 |a_1\rangle + \lambda_2 |a_2\rangle + \lambda_3 |a_3\rangle + \ldots + \lambda_N |a_N\rangle \right) \quad (B.1)
$$

$$
= \lambda_1 a_1 |a_1\rangle + \lambda_2 a_2 |a_2\rangle + \lambda_3 a_3 |a_3\rangle + \ldots + \lambda_N a_N |a_N\rangle. \quad (B.2)
$$

Here $a_i$ are the eigenvalues of $A$, and $|a_i\rangle$ are its eigenvectors. Repeated application gives

$$
A^k |v\rangle = \lambda_1 a_1^k |a_1\rangle + \lambda_2 a_2^k |a_2\rangle + \lambda_3 a_3^k |a_3\rangle + \ldots + \lambda_N a_N^k |a_N\rangle. \quad (B.3)
$$

As $A$ is applied, the vector becomes more and more aligned with the eigenvector that has the largest eigenvalue.

While the method might be useful for some problems where the single largest eigenvalue is desired, in the problems that this thesis examines we wish to determine the two smallest eigenvalues of a system. The Lanczos method provides a way to achieve this. It begins in the same way as the power method: a random vector is multiplied by the matrix being investigated, but instead of continuing to act on the resultant vector a sequence of vectors is produced from the following algorithm:

1: choose a starting vector $|v_0\rangle$
2: $|v'_0\rangle = A |v_0\rangle$
3: $|v_1\rangle = |v'_0\rangle - \langle v_0 | v'_0 \rangle |v_0\rangle$
4: for $i=2$ to $n-1$ do
5: $|v'_i\rangle = A |v_{i-1}\rangle$
6: $|v_i\rangle = |v'_i\rangle - \langle v_{i-1} | v'_i \rangle |v_{i-1}\rangle - \langle v_{i-2} | v'_i \rangle |v_{i-2}\rangle$
7: end for

For large enough $n$, the extremal eigenvalues (those of both largest and smallest magnitude) of $A$ will be approximated by the eigenvalues of the matrix $\langle v_i | A | v_j \rangle$. Because of the recursion relation used to generate the vectors, if $A$ is real and symmetric this matrix is tridiagonal, so it can rapidly be diagonalized by conventional means.
This algorithm becomes even more powerful when it is combined with restarting. Restarting is a procedure where a modest value of $n$ is chosen ($n \ll \text{size of } A$) and the vectors $|v_0\rangle \ldots |v_{n-1}\rangle$ are created. The eigenvalues of $\langle v_i | A | v_j \rangle$ are then found and the few that are closest to the eigenvalues being sought are selected. The eigenvectors associated with those eigenvalues and the vectors $|v_i\rangle$ associated with them are used as the first several vectors in the procedure described above. Since the accuracy of any eigenvalue can always be checked directly, the process of restarting can continue until the desired accuracy is achieved. This allows one to find a small number of eigenvalues from enormous matrices by storing only a few vectors. Using this procedure we were able to find the two smallest eigenvalues of a $2^{23} \times 2^{23}$ matrix with $n=6$. 
Appendix C

Characteristics of the Distributions of the Data

The statistical analyses in this thesis are performed to determine the typical behavior of the quantum adiabatic algorithm on random instances of problems drawn from certain classes. For this reason, when examining distributions of runtimes or minimum energy gaps the median is the most meaningful statistic. However, there is much more information in these distributions than is represented by this one number. This appendix presents the full distributions for a subset of the data discussed in this thesis. Each distribution is presented in the form of a histogram containing 100 bins.

From the integration studies, the full distributions of runtimes required to achieve probability 50% is presented for 1000 instances of 3-SAT at 6 bits and 10 bits. Then the distributions that result from running 1000 instances at the median running time required to achieve probability 50% is presented at 6 bits and 10 bits. These distributions are also presented for scrambled 6 bit instances of 3-SAT.

From the energy gap studies, the distributions of minimum gaps for instances of 3-SAT and scrambled 3-SAT are presented at 10, 15, 20, and 22 bits. The distributions of minimum gaps for unscrambled instances have tails that extend several orders of magnitude below the median value of the distributions. To see these tails, the distributions of \( \log_2(\text{Minimum Gap}) \) are also presented.

Accompanying each histogram is a table giving a few more of the statistics of
the distributions. In these tables the median error is found by ordering the data in ascending order and subtracting the value of the data point at the location of the median minus the square root of the size of the distribution from the data point at the location of the median plus the square root of the size of the distribution. For example take a distribution of 100 number and order them into a list. The median is at location 50. The error on the median is found by subtracting the value at location 40 in the list from the value at location 60. Likewise the numbers in the table labeled $i^{th}$ percent are found by looking at that location in the list. For the example list 90$^{th}$ percent means the number at location 90 in the ordered list.

Finally it should be pointed out that the distribution in the section on integration studies are not the distributions from which the plots in section 3.2 were drawn. When this study was performed, running times were not found for instances that proved to required times that exceeded the current median by more than a factor of 2. This did not effect the determination of the median as can be seen by comparing the data found here with the data in chapter 3. For the distributions presented here all of the running times have been determined no matter how large they are.
C.1 Integration Studies

C.1.1 3-SAT

![Distribution of Running Time to Achieve Probability 50\% for 1000 3SAT Instances at 6 Bits](image)

Figure C-1: Distribution of Running Times for 1000 3-SAT Instances at 6 Bits.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Instances</th>
<th>Median</th>
<th>Median Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1000</td>
<td>35.13</td>
<td>1.89</td>
</tr>
<tr>
<td>10th %</td>
<td></td>
<td>20.70</td>
<td>90th %</td>
</tr>
<tr>
<td>1st %</td>
<td></td>
<td>13.39</td>
<td>99 %</td>
</tr>
<tr>
<td>Minimum</td>
<td></td>
<td>10.62</td>
<td>Maximum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1401</td>
</tr>
</tbody>
</table>

Table C.1: Statistics of the Distribution of Running Times for 1000 3-SAT Instances at 6 Bits
Figure C-2: Distribution of Running Times for 1000 3-SAT Instances at 10 Bits.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>10</th>
<th>Instances</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>72.54</td>
<td>Median Error</td>
<td>8.129</td>
</tr>
<tr>
<td>10&lt;sup&gt;th&lt;/sup&gt; %</td>
<td>37.70</td>
<td>90&lt;sup&gt;th&lt;/sup&gt; %</td>
<td>293.7</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; %</td>
<td>25.02</td>
<td>99&lt;sup&gt;th&lt;/sup&gt; %</td>
<td>2278</td>
</tr>
<tr>
<td>Minimum</td>
<td>19.46</td>
<td>Maximum</td>
<td>8034</td>
</tr>
</tbody>
</table>

Table C.2: Statistics of the Distribution of Running Times for 1000 3-SAT Instances at 10 Bits
Figure C.3: Distribution of Probabilities for 1000 3-SAT Instances at 6 Bits Run for Time T=35.4

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Median</th>
<th>Instances</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4868</td>
<td>Median Error</td>
<td>0.03186</td>
</tr>
<tr>
<td>10th %</td>
<td>0.2645</td>
<td>90th %</td>
<td>0.7072</td>
</tr>
<tr>
<td>1st %</td>
<td>0.1437</td>
<td>99th %</td>
<td>0.8329</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.08883</td>
<td>Maximum</td>
<td>0.9127</td>
</tr>
</tbody>
</table>

Table C.3: Statistics of the Distribution of Probabilities for 1000 3-SAT Instances at 6 Bits Run for Time T=35.4
Figure C.4: Distribution of Probabilities for 1000 3-SAT Instances at 10 Bits Run for Time T=71.4

Table C.4: Statistics of the Distribution of Probabilities for 1000 3-SAT Instances at 10 Bits Run for Time T=71.4
C.1.2 Scrambled 3-SAT

Distribution of Running Time to Achieve Probability 50%
for 1000 Scrambled 3SAT Instances at 6 Bits

![Distribution of Running Time](image)

Figure C-5: Distribution of Running Times for 1000 Scrambled 3-SAT Instances at 6 Bits.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Median</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>57.81</td>
<td>1000</td>
</tr>
<tr>
<td>10th %</td>
<td>29.8151</td>
<td>Median Error</td>
</tr>
<tr>
<td>1st %</td>
<td>19.29</td>
<td>5.913</td>
</tr>
<tr>
<td>Minimum</td>
<td>19.46</td>
<td>174.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>99th %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>682.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8035</td>
</tr>
</tbody>
</table>

Table C.5: Statistics of the Distribution of Running Times for 1000 Scrambled 3-SAT Instances at 6 Bits
Figure C-6: Distribution of Probabilities for 1000 Scrambled 3-SAT Instances at 6 Bits Run for Time $T=57.3$

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Median</th>
<th>10$^{th}$ %</th>
<th>1$^{st}$ %</th>
<th>Minimum</th>
<th>Instances</th>
<th>Median Error</th>
<th>90$^{th}$ %</th>
<th>99$^{th}$ %</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.4958</td>
<td>0.2184</td>
<td>0.08415</td>
<td>0.02255</td>
<td>1000</td>
<td>0.03795</td>
<td>0.7751</td>
<td>0.9276</td>
<td>0.9795</td>
</tr>
</tbody>
</table>

Table C.6: Statistics of the Distribution of Probabilities for 1000 Scrambled 3-SAT Instances at 6 Bits Run for $T=57.3$
C.2 Energy Gap Studies

C.2.1 3-SAT

Figure C-7: Distribution of Minimum Gaps for 3700 3-SAT Instances at 10 Bits

Table C.7: Statistics of the Distribution of Minimum Gaps for 3700 3-SAT Instances at 10 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Instances</th>
<th>Median Error</th>
<th>Median</th>
<th>10th %</th>
<th>90th %</th>
<th>1st %</th>
<th>99th %</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3700</td>
<td>0.0137</td>
<td>0.3493</td>
<td>0.1462</td>
<td>0.6524</td>
<td>0.0462</td>
<td>0.9754</td>
<td>0.0052</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Figure C-8: Distribution of $\log_2$(Minimum Gap) for 3700 3-SAT Instances at 10 Bits (Same data as figure C-7)
Figure C-9: Distribution of Minimum Gaps for 3700 3-SAT Instances at 15 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>15</th>
<th>Instances</th>
<th>3700</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.2296</td>
<td>Median Error</td>
<td>0.0149</td>
</tr>
<tr>
<td>10\text{th} %</td>
<td>0.0438</td>
<td>90\text{th} %</td>
<td>0.5041</td>
</tr>
<tr>
<td>1\text{st} %</td>
<td>0.0035</td>
<td>99\text{th} %</td>
<td>0.8068</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0001</td>
<td>Maximum</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table C.8: Statistics of the Distribution of Minimum Gaps for 3700 3-SAT Instances at 15 Bits
Figure C-10: Distribution of $\log_2(\text{Minimum Gap})$ for 3700 3-SAT Instances at 15 Bits (Same data as figure C-9)
Figure C-11: Distribution of Minimum Gaps for 931 3-SAT Instances at 20 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>20</td>
</tr>
<tr>
<td>10th %</td>
<td>0.0120</td>
</tr>
<tr>
<td>1st %</td>
<td>0.0005</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.00005</td>
</tr>
<tr>
<td>Median Error</td>
<td>0.0319</td>
</tr>
<tr>
<td>90th %</td>
<td>0.4302</td>
</tr>
<tr>
<td>99th %</td>
<td>0.7137</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table C.9: Statistics of the Distribution of Minimum Gaps for 931 3-SAT Instances at 20 Bits
Figure C-12: Distribution of $\log_2(\text{Minimum Gap})$ for 931 3-SAT Instances at 20 Bits. (Same data as figure C-11)
Figure C-13: Distribution of Minimum Gaps for 592 3-SAT Instances at 22 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>22</th>
<th>Instances</th>
<th>592</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.1279</td>
<td>Median Error</td>
<td>0.034</td>
</tr>
<tr>
<td>10th %</td>
<td>0.0059</td>
<td>90th %</td>
<td>0.3839</td>
</tr>
<tr>
<td>1st %</td>
<td>0.0002</td>
<td>99th %</td>
<td>0.6379</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0001</td>
<td>Maximum</td>
<td>0.7341</td>
</tr>
</tbody>
</table>

Table C.10: Statistics of the Distribution of Minimum Gaps for 592 3-SAT Instances at 22 Bits
Figure C-14: Distribution of $\log_2$(Minimum Gap) for 592 3-SAT Instances at 22 Bits. Same data as figure C-13.
C.2.2 Scrambled 3-SAT

Figure C-15: Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 10 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>10 Instances</th>
<th>Median Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.1683</td>
<td>0.0271</td>
</tr>
<tr>
<td>10th %</td>
<td>0.0805</td>
<td>0.2621</td>
</tr>
<tr>
<td>1st %</td>
<td>0.0318</td>
<td>0.3906</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0307</td>
<td>0.4407</td>
</tr>
</tbody>
</table>

Table C.11: Statistics of the Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 10 Bits
Figure C-16: Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 15 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>15</th>
<th>Instances</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.0613</td>
<td>Median Error</td>
<td>0.0046</td>
</tr>
<tr>
<td>10th %</td>
<td>0.0461</td>
<td>90th %</td>
<td>0.0847</td>
</tr>
<tr>
<td>1st %</td>
<td>0.0338</td>
<td>99th %</td>
<td>0.1271</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0329</td>
<td>Maximum</td>
<td>0.1288</td>
</tr>
</tbody>
</table>

Table C.12: Statistics of the Distribution of Minimum Gaps for 200 Scrambled 3-SAT Instances at 15 Bits
Figure C-17: Distribution of Minimum Gaps for 214 Scrambled 3-SAT Instances at 20 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Instances</th>
<th>10th %</th>
<th>Median Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.0165</td>
<td>0.0133</td>
<td>0.0008</td>
</tr>
<tr>
<td>1st %</td>
<td>0.0123</td>
<td>0.0205</td>
<td>0.0258</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0096</td>
<td>Maximum</td>
<td>0.0291</td>
</tr>
</tbody>
</table>

Table C.13: Statistics of the Distribution of Minimum Gaps for 214 Scrambled 3-SAT Instances at 20 Bits
Figure C-18: Distribution of Minimum Gaps for 139 Scrambled 3-SAT Instances at 22 Bits

Table C.14: Statistics of the Distribution of Minimum Gaps for 139 Scrambled 3-SAT Instances at 22 Bits

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Instances</th>
<th>Median</th>
<th>Median Error</th>
<th>90th %</th>
<th>99th %</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>139</td>
<td>0.0095</td>
<td>0.0008</td>
<td>0.0111</td>
<td>0.0128</td>
<td>0.0136</td>
</tr>
</tbody>
</table>
Bibliography


93


[MS96a] K. Maschhoff and D. Sorensen, A portable implementation of ARPACK for distributed memory parallel architectures, 1996.


