Overview

My thesis is concerned with the Hamiltonians used in quantum simulation. We are interested in the computation of the ground state of such Hamiltonians. Based on the type of ground state which is either a tensor product or an entangled state, or have some other classifiable characteristics, we devise Hamiltonians that work on these ground states. We have to use perturbation theory and matrix algebra to make our Hamiltonian work. In the end if the ground state energy of the Hamiltonian is lower bounded by 1/poly(n), we can efficiently simulate the circuit represented by this Hamiltonian. To summarize, at this stage we classify how to solve a problem, not by problem type, but by solution type. There is not a single type of Hamiltonian that can solve all problems, but there are several different types of Hamiltonian to target different classes of ground states.

The Hamiltonians used to do computation are called local Hamiltonians. So what are local Hamiltonians? If for example our entire computation consists of n qubits. A local Hamiltonian does computation on only a fixed constant subset of them; say 4 qubits. In classical computing, the most common gates of logic are AND, OR and NOT. In quantum computing, the gates of logic operations are Pauli X, Y, Z and CNOT. The Hamiltonian used to model our system need not be exact. For example it might suit our specific purpose to use the Hadamard gate instead of the Z gate. Obviously the two gates are not the same, but for our specific target operation, they are the same. Furthermore our simulation needs to be accurate for a period of time. We use the Schrodinger equation to make sure of this. At the heart of our research in simulation seems to be the degenerate eigenenergies of the Hamiltonian. We are interested in what are the characteristics of a matrix that has degenerate eigenvalues. We are also interested in the ground state. The ground state is important because it is only intuitive that the correct solution should take the lowest energy to access, so that you have to try less hard to extract it at the end of your computation, than all the other states.

This semester of research was about why building the quantum computer is currently still a challenge. It is challenging because of there are currently no good error correction codes and quantum algorithms. This semester was also about how to simulate the quantum computer on a classical computer. Now I will discuss what I did during this semester. During the first two weeks of this semester, I researched quantum error correction methodologies. The exponential growth of the error correction circuit size is one of the factors that limits what problems we can solve on a quantum computer. In other words, we discovered that with error correction, the circuit size might grow faster than polynomial. Next we downloaded the jQuantum quantum simulator from off the world-wide web, to aid us in our study of the Shor's, Grover's, and Deutch Josza Algorithms. This quantum simulator represents a case of how to use the classical computer to simulate quantum mechanical concepts, for a small number of qubits— 18 qubits maximum. Richard Feynman once said that quantum mechanical concepts can not be efficiently simulated with classical computer for large qubit numbers, but 18 qubits is still a small number. Looking, at the source code for jQuantum, I got an appreciation of how a function that need to be calculated N times on a classical computer through the use

of a loop, can be done in just one step on a quantum computer. By the way, the jQuantum software can also be used to write error correction codes. Then next, we studied a Quantum Hamiltonian Complexity paper. On one hand, this paper is about models, approximations and ground states. On the other hand, this paper is about proving that the k-local Hamiltonian is QMA-complete, which would mean that it is solvable on a quantum computer. From here we make our happy down-hill slide because k-QSAT is a special case of k-local Hamiltonian, and if k-local Hamiltonian is solvable so is k-QSAT. Thus we can now solve NP-complete problems. (To distinguish k-QSAT from other k-Local Hamiltonian is that in k-QSAT the local Hamiltonians are all projectors, which are not versatile but still a good start). At the end of the semester, we looked at a polynomial time algorithm for 2-QSAT. If successfully implemented, this algorithm is the proof that k-QSAT is solvable, at least for k=2. Next semester we will study the clause to variable ratio of a barely solvable problem. Through exploring models such as the Ising model by writing codes in Java to test the satisfiability, we will gain insight into this ratio.

Quantum Error Correction

A major reason of why Quantum Computing research continues to be an arduous struggle is that there is currently no way to achieve error correction in an economical way. The advantage of Quantum Computing is supposed to be that it uses polynomial sized circuits to solve exponential sized problems. However quantum error correction causes exponential growth in circuit size. Currently, we solved the problems of detecting the errors but there is still no efficient way for correcting the errors after detecting them. The quantum simulator I used in my research, jQuantum, was invented partly for this purpose. I still have not had the time to truly sit down and think if I can come up with a better error correction code using jQuantum, but I will as soon as possible.

The problem with the current strategy is that error correction will cause the amount of gates to be disproportionately big if we want a small error threshold (for example 10^-5). In fact for a 21 qubit circuit, the number of gates would be on the order of 10^19. As you can see this polynomial sized circuit is even bigger than an exponential circuit! So probably what is currently happening in the quantum computing research labs, is that we tolerate large error rates (such as 10^-1) so that we don't waste money on the gates. In the ion trap, the gates are laser pulses, and currently computation that requires more than a couple hundred laser pulses at a time is impossible.

In general, we don't know what noise is afflicting a system, so it would be useful if a specific code C and error-correction operation R could be used to protect against an entire class of noise processes. But in many respects quantum error correction codes are quite similar to classical codes. An error is identified by measuring the error syndrome, and then correcting it as in the classical case. So far there is no major setback in how quantum computing works in theory. However in the real world, environment interacts with the quantum system and causes decoherence and other error to occur. Noise models were constructed to help us better understand the nature of quantum errors. Right now, each quantum error is modeled as the original gate followed by another quantum gate that represents the error. The most common errors are constructed from X, Y, Z pauli gates. The effects of these noises that they cause either a spin-flip error or a phase error, or

some combination of the two cases. To monitor spin-flip errors we use the quantum state $(|000\rangle + |111\rangle)/\sqrt{2}$, and to monitor phase errors, we use the quantum state $(|+++\rangle + |---\rangle)/\sqrt{2}$.

The Shor code, named after its inventor, protects against arbitrary error on a single qubit. Suppose a bit flip occurs on the first qubit, we perform measurement Z_1Z_2 comparing the first two qubits, and find that they are different. We conclude bit flip occurred for either one of them. Next we perform measurement Z_2Z_3. We find they are the same. We conclude the first qubit must have flipped. When a phase flip occurs on any one the first three qubits we find that the signs of the first and second blocks are different. The second and final stage of syndrome measurement is to compare the sign of the second and third blocks of qubits. We find that these are the same, and conclude that the phase must have flipped in the first block of three qubits. The drawback is this only work for a single qubit; with entanglement it is not going to work, because two qubits are involved.

As mentioned earlier, many respects quantum error-correction codes are similar to their classical counterparts; there is however an interesting class of quantum codes known as degenerate codes, that includes a wide variety of codes such as the Shor code and the Steane code. Degenerate codes, pack and store more information than classical code. However some of the proof techniques used classically to prove bounds on error-correction is invalid for degenerate codes. The quantum Hamming bound is a simple bound that gives insight into the properties of quantum codes; it gives the total number of errors that may occur on n or fewer qubits. This bound basically tells us whether or not a code with particular characteristics exists or not. The bad news is Hamming bound only applies to non-degenerate codes. Our first example of a large class of degenerate quantum error-correcting codes is the Calderbank-Shor-Steane codes usually known as the CSS codes. The stabilizer codes are a class of codes that is even more general than the CSS codes, and allows us to construct a wide variety of quantum codes. One of the features of stabilizer codes is that their structure enables systematic construction of procedure for encoding, decoding, and error correction. Stabilizer codes are analogous to classical linear codes. Stabilizers' power lies in the clever use of the group theory, and they describe state spaces under quantum operations. A set of generators G generate a stabilizer for the code space of, for example, the seven qubit Steane code. A linear code C encoding k bits of information into n bit code is specified by an n by k generator matrix G. Furthermore, error correction for linear codes is more clearly understood by introducing the parity check matrices H. The code is defined to be the nullspace of H, and it is subject to corruption by an error E. To construct H, we pick out n-k linearly independent vectors that are orthogonal to the columns of the generator G. Also, H has n columns.

To keep errors constantly in check is inevitably hard to do, especially because of the fact that we can not clone a quantum state. Since we can not get tensor producted states, we settle for the second choice of using entangled states. For example if the state was

originally $\alpha|0\rangle + \beta|1\rangle$, we encoded it as $\alpha|000\rangle + \beta|111\rangle$. Basically how we achieve this is through the use of CNOT gates to create many copies of our qubit—usually a total of 3 copies would be enough. If you think about this you will realize why it causes exponential growth of circuits: if you originally have 3 qubits, after a layer of gates, you will have 3^2 qubits, and after 2 layers of gates, you will have 3^3 qubits. Why this is actually not a problem is you are not going to apply gates to all of these qubits only a portion of them, and ideally you want to first fix the error, before applying more gates only to your original qubits, so that the number of gates or qubits will not grow exponentially.

jQuantum

jQuantum is a quantum simulator. Quantum simulators allow us to simulate the behavior of a quantum computer on a classical computer. jQuantum was put on the worldwide web and open to the public to download for free, because its creators wish to encourage writers of new quantum algorithms. Somehow the current Shor's and Grover's algorithms are not stable and reliable enough, and there is demand for a better quantum algorithm. jQuantum is a program written in Java. This is quite a logical choice for program language since Java's Swing library allows us to create a very user friendly and attractive looking Gui, or framework in which the program runs. If it was written in C++, it would have been much less inviting and much more difficult to use. You will have to use the command shell, and learn by heart all the commands written for each quantum gate.

To help you visualize what is happening to all your quantum states and how likely you are to get each one of them if you do a measurement, there is an array of colored squares that gives you the complex coefficients of each quantum state in the summation. From the colors of the squares you can infer the probability of the future measurement being in that state. Basically the color represents the phase and the intensity represents the amplitude. In the Grover algorithm simulation that I made, I was able to view the gradual intensification of red color in the square I am searching for, as I pass through each Grover cycle.

For the Shor simulation—I realized why this algorithm is very unrealiable and can not be trusted. Based on the results of in jQuantum, I learned why the Shor algorithm is unrealiable and why we need a new algorithm. First of all the algorithm can only factor numbers with an even integer as period. Second of all it is more reliable at factoring some numbers than others even assuming both could be factored. For example, I discovered that it is better at factor 15 than 21. This is because for 15 Q/r =128/4=32; while for 21 Q/r=512/6=85.333. Since for 21 the number Q/r is not a whole number, it introduces more ambiguity and more fluctuations in the solution.

First I want to affirm that jQuantum definitely have many virtues. I also like the fact that this program has many nice details. I liked the conceptual clarity of this program. Each quantum state is represented by a square illustrated at the bottom of the panel. So that there is no confusion between the qubit number and the quantum state number.

It is also true to the fact that you can not reverse a measurement. Also it calculates the number of Grover's iterations for you, and it rounds down rather than up, so that you will not waste gates in your circuit design. Instead of getting 100% you get 95% instead, but you will never get 105% any way, so there is no need to waste the last gate. Another detail I liked was the fact that you can reset the input qubits after you have constructed your circuit. Just think how frustrated you would be if your circuit can not be reused for different inputs.

However jQuantum is not a perfect program. Sometimes it seems more like a toy than something you can make scientific progress with. But I definitely think that it is a good start and needs only a few suggestions to make it even better. I didn't like the fact that it is not easy to correct a mistake when you are building a very complicated circuit, that is prone to mistakes, this causes huge waste of time. For example after constructing, layer 12 you suddenly found a mistake in layer 3. At this stage, jQuantum only allows you to delete all the gates from 3 to 12 to correct your error. I don't understand, why we can not just delete layer 3 and replace it with something else. I also don't like the fact that there is no mechanism for shifting the circuit if it won't fit in one screen. It is not too tiring for me to have to scroll by just a small step each time I add another gate that won't fit on the screen.

I looked at the source code folder of the jQuantum. The file that did all the calculations of the quantum gates, is called Register.java. I noticed that they used two arrays called real and imaginary to keep track of the coefficient of all the quantum states. There was extensive use of for-loops to increment through the real and imaginary arrays. Through the use of a Hashset, they were able to keep track of all the terms that will be changed when you pass your original quantum state through a quantum gate. For single qubit gates, no entangling action takes place, and you can increment your for loops by large chunks, while for entangling gates, you have to increment step by step through the entire array. The math behind the majority of the gates in this program is multiplying a vector by a matrix to get the new vector. But because we want to be as efficient as possible, we do not want the entire matrix, we only want a "local" matrix action, that acts on only one of the qubits or at most 2 or 3 at a time. That's why we use the for-loop to skip over the terms that will never be changed. Also the logic behind what to include in your Hashset can be very complicated during computation, and can involve checking whether an element is already in the Hashset. The calculations Inverse Quantum Fourier Transforms is more complicated and involves the use of the sine and cosine functions. I am personally in awe of the person who came up with how to implement the bit reversal algorithm. I am also in awe of the rest of the FFT. But after the FFT the rest of the Inverse Quantum Fourier Transform is pretty much self explanatory. Register.java is the file that does all the computations for the quantum gates. Below I am including an analysis of one of the methods in Register.java, for the Hadamard gate.

```
public void hadamard( int j ) {
   double realTmp, imaginaryTmp;

   for ( int k = 0; k < power2( size ); k += power2(j) ) {
      for ( int l = 0; l < power2(j-1); l++ ) {
         int i0 = k + l;
         int i1 = k + l + power2(j-1);

         realTmp      = real[ i0 ];
         imaginaryTmp = imaginary[ i0 ];

         real[ i0 ] = ( realTmp + real[ i1 ] ) / SQRT2;
         imaginary[ i0 ] = ( imaginaryTmp + imaginary[ i1 ] ) / SQRT2;
         real[ i1 ] = ( realTmp - real[ i1 ] ) / SQRT2;
         imaginary[ i1 ] = ( imaginaryTmp - imaginary[ i1 ] ) / SQRT2;
      }
   }
}
```

Explanation of the above function Register: hadamard()

- The function power2(int n), basically calculates the nth power of 2. Here x is the total number of qubits, and the number that is stored in x qubits is less than 2^n. In other words, the n qubits can represent any number from 0 to $2^n - 1$.
- The argument passed to the Hadamard function is j. This means we want to perform our computation on the jth qubit. Based on the convention in this program, so that it is easier to keep track of for further development, the smallest j can be is 1 not 0. j=1 is the lowest qubit or the digit that represents 2^0 in our number or state that is represented by the n qubits.
- They used two for-loops in the above code. Understanding what they are used for, takes knowledge of tensor products. If you write out what a tensor product means on paper, it will make more sense. Basically, Power2(size) is the bound based on the size of our circuit; the number of qubits is represented by variable 'size'. l and k are two increments that we have to consider so that we can be sure that we only change the elements in the quantum state that actually changed when we do operations on the jth qubit. k increments to the starting point of each block of elements lower than and including the jth qubit. l increments through each element of the block of elements. So in the end each element in the quantum state is considered, just in a different order than in counting sequence.
- The indices i0 and i1 are used for doing the Hadamard gate computation. i0 and i1 are separated by an increment that is the size of the block of qubits that are less than j. The relation between i0 and i1 is that they represent the pair of elements that the quantum gate somehow modify based on the original values.
- The rest of this algorithm is where we do a matrix multiplication to find our final quantum state.
-

Deutch Josza Algorithm

The Deutch Josza algorithm is a demonstration of using quantum mechanical concepts to achieve information processing. This algorithm takes an input function and outputs whether this function is a constant function or a balanced function; balanced is another word for exactly half 0 and half 1. This algorithm basically takes in two numbers x and y; y is interfered with f(x) through $y \oplus f(x)$ and generates an output; and then register 1, the one representing x is measured. When we measure, we should get 0 only if the function is constant, and something other constant if the function is not constant.

As part of my research leading up to my thesis, I programmed a Deutch Josza algorithm in Java. Through this task, I was able to experience how hard it is to simulate quantum systems on the classical computer. I used a lot of for-loops so that I can calculate f(x,y) for each pair of x and y within the range of my register size. The quantum computer is able to do this in one step while on the classical computer you have to do calculation for each x, y pair. What is a loop procedure on the classical computer is a physics phenomenon on the quantum computer. On the classical computer, it takes, 2^n/2+1 measurements to make sure that the function is constant, while on the quantum computer it takes only measurment. The concept of measurement itself is also difficult to simulate. Java has a Math.random() function that simulates randomness; but it does not simulate randomness based on a probability distribution other than equal probability for each state. In my program, I simply assumed that I will always get as the result the state with the highest probability. There is probably a way to fix this problem and maybe I will try it someday.

The Deutch Josza algorithm itself was easier to understand than to program. The hardest part is working with a String input and parsing the function into an array of elements. The next hardest part is writing the algorithm for solving the function. Getting the grand concept of recursion is actually simple compared to mechanisms of storing each step as you go along. A lot of things humans do intuitively are very frustrating to tell a computer how to do. I also find that what is difficult about programming in general is staying consistent so that all the variables in your program match up in the correct order. A good advice is to use copy and paste when you do your programming so that you will have less typos. But when you use copy and paste, you have to change some parts of the program and it is crucial to stay consistent.

Computational Complexity

Now allow me to explain the complexity of a problem in classical computing theory. A problem is in NP if given the correct input and output pair, you can find a polynomial algorithm that verifies the correctness of this pair. An NP-hard function satisfies the following criteria: inputting a polynomial function into the NP-hard function, gives you an NP function. A problem is NP-complete if it is both NP and NP-hard. We currently can not solve NP complete problems on a classical computer. We are thinking that even though the classical computer can not solve NP complete questions, maybe the quantum computer can. We have to first mutate the problem into quantum logic form. In this transformed form, the NP complexity is called the QMA, and the NP-hard is called QMA-hard, and equivalently the NP-complete is called the QMA-complete. From here

on, we have to use quantum algorithms to solve problems on a quantum computer, because the classical algorithms will not apply.

Now I will elucidate how the sets of computational complexity fit together into the big picture. $P \subseteq BPP \subseteq BQP \subseteq QMA \subseteq PSPACE$. NP contains all of P and some of BPP and BQP, and is contained entirely within QMA. Currently we believe the power of the quantum computer is BQP. Scientists want eventually to be able to solve problems in QMA so that all the NP problems will be solved for certain. That QHC paper covers the fact that k-LH is included within QMA.

Quantum Hamiltonian Paper

Quantum Hamiltonian Complexity is the study of quantum constraint satisfaction problems. The details of this field involve the establishment of the Cook-Levin Theorem, and the modeling of 1D low temperature quantum systems via area laws. But from a farther distance, it is the study of local quantum constraints. Local Hamiltonians are Hamiltonians that occur in nature. We are also interested in the ground state. Intuitively, the ground state can be thought of as the vector $|\psi\rangle$ which maximally satisfies all the constraints or local Hamiltonians $\{H_i\}$. First of all, based on the Schrodinger equation the evolution of quantum states occurs due to local Hamiltonians. Local Hamiltonians only affect a portion of the system out of the entire system of qubits. Therefore just as classical logic clause force its bits to lie in a subspace, quantum clause H_i also restrict the k qubits to a certain subspace. QHC asks the questions such as: which quantum systems have ground states that have classical representation? Can we use classical simulations to predict when a quantum system will exhibit interesting phenomenon such as phase transition? Can we quantify the hardness of determining certain properties of local Hamiltonians?

To describe the joint state of n qubits requires an exponential sized density matrix ρ, this is why quantum computers are much more powerful. Entanglement is another trait that distinguishes the quantum world from the classical world. Very exotic correlations can exist between two systems A and B. Looking at the density matrix describing system AB, on one end of the spectrum, they exist in a tensor product, on the other end of the spectrum they exhibit entanglement. We quantify the amount of entanglement by entropy of entanglement. Like the classical systems, quantum systems are subjected to noise. Their interactions with the environment cause noise to be injected into the system. To model this we permit probabilistic mixtures of pure states more generally referred to as mixed states. The probabilities add up to 1. An open quantum system is one that interacts with its environment. An open quantum system A can be simulated by using a closed joint system AB, evolving AB using unitary operators and subsequently tracing out part of AB.

An interesting problem that QHC is concerned with is finding the ground state energy. This problem has many interesting forms and is of QMA-hardness. Not only do we want to find the optimum solution, we want to find the approximation to the optimum solution as well, which is talked about in this QHC paper. Just as we can do everything we want with classical Hamiltonians, we also want to be able to do everything we want with quantum Hamiltonians. The paper talks about tensor networks which are used by condensed matter physicists, and also discusses the proof that LH problem is QMA-complete, which were done by computer scientists. It was easy to find out that the general LH problem is QMA-complete. But there are also special LH problems such as the Quantum SAT, Stoquastic LH, commuting LH, which were all proven QMA-complete too after much greater efforts. The terms related to QHC include: mean field theory, tensor networks, Density Matrix Renormalization Group (DMRG algorithm) and Multi-Scale Entanglement Renormalization Ansatz. How to use these are all related to how to classically simulate quantum systems.

To aid us in our exploration of quantum systems, we often use the classical computer. The last two decades have seen much effort towards classifying when a ground state can be described classically. Now we move on to describing ground states quantum mechanically, and we work with these ground states using quantum algorithms, one algorithm discussed in this paper is the DMRG. The DMRG operates on a type of tensor network called the Matrix Product State (MPS). MPS are used to efficiently classically simulate "slightly entangled" quantum computation. Generalization of MPS to higher dimensions are tensor networks called PEPS and MERA. But these are not easy to work with since, even the problem of determining whether a tensor network represent a non-zero vector is not in Polynomial-Time Hierarchy. There are other quantum algorithms that are not discussed in this paper that I read about in other articles. (The quantum adiabatic computation algorithm operates on balanced Boolean inputs that evolve in time. The Solve-Q algorithm is a 2-QSAT algorithm that operates on chain reactions and discretised cycles. During the semester of my research, I worked with k-QSAT constraint satisfaction problems. I had the experience of seeing how the Solve-Q algorithm solves a 2-QSAT constraint satisfaction problem.) As it is we are only beginning to understand how to process information using quantum mechanics and how to represent them using higher level mathematics; so quantum algorithms are few, and we need people to write more of them before we can build a quantum computer.

Computer scientists and Condensed matter physicists approach the problem of finding quantum algorithms in two different ways. Computer scientists, use constraint graphs to study constraint satisfaction problems. Condensed matter physicists use simplified models to approximate the real world particle interactions, by studying in a laboratory how a collection of particles interact with their nearest neighbors, and swapping those particles to aid in computation.

The question that is relevant to many condensed matter physicists is that given a quantum state, can we compute some local property. Some local properties are intensive and do not scale with system's size. Other local properties such as the ground state energy are extensive properties and do depend on system size. In practice there is no way to know

the actual Hamiltonian that is causing a quantum system to evolve in time—all we can do is make an educated guess. We desire to find a simplified model which ignores certain degrees of freedom, but nevertheless captures the local properties of interest. After we have the model, we want to find its ground state energy. To better understand physical systems, condensed matter physicists model them using Hamiltonians, but studying Hamiltonians is a very a challenging subject. They currently want to find efficient algorithms for approximating local properties of a Hamiltonian. Where as to a computer scientist, efficient means the algorithm theoretically runs in polynomial time; to a condensed matter physicist efficient means that it is fast and actually works well in practice. The variational principle simply means optimization over restricted set to make computation easier. Three types of restricted sets are used to model quantum states: Product States, Gaussian States and Tensor Networks. Products states are for separable states. Guassian states and tensor networks are for entangled states. A key question is: Which quantum systems have ground states that can be well-approximated by a tensor network of small bond dimension? The bond dimension needs to be small so that the circuit is polynomial sized.

Unlike computer scientists, condensed matter physicists work on building something without proving whether it is mathematically possible because most likely it is possible. Physicists have come up with models of particle systems and how they behave. Both classical and quantum simulations have been used to uncover the properties of the Hubbard model. Bosons and fermions are indistinguishable particles. Swapping bosons will not do anything to the wave function, while swapping any pair of fermions will induce a global phase of -1 on the wave function. Boson and fermions have different possible statistical distributions governing how a system of indistinguishable and non-interacting particles populates a set of discrete energy states. Hubbard proposed the Hubbard model to explain the behavior of strongly correlated materials. In particular, this model describes the behavior of electrons in solids, and capture s the transition of a system between being a conductor and an insulator. In the Hubbard model, the tuning parameters are the electron density and the repulsion strength, and the 1D model has been solved by Lieb and Wu. Basically when we remove electrons, it causes the insulator to become a conductor. Beyond 1D, the Hubbard model is very difficult to solve.

After coming up with classical models, condensed matter physicists have upgraded them to quantum versions. For example there is now both a classical and a quantum version of the Ising model. Each model is described by a many-body Hamiltonian. An interaction graph illustrates which sets of particles are constrained by a common local Hamiltonian. Solving the model means determining some property of system evolving due to the Hamiltonian. The commonly studied models are: Ising, Heisenberg, and AKLT models. The mean field theory is a variational methold over the set of product states to qualitatively extract properties of a many-body system. A many-body Hamiltonian H is difficult to solve due to coupling between particles. To circumvent this, mean-field theory constructs the mean field Hamiltonian, which is an approximation of the actual Hamiltonian. Sometimes the Hamiltonian is too symmetrical and so we add a symmetry breaking perturbation—for example the linear term you see in the Ising model. We want to get the mean field Hamiltonians so that qubits do not interact, and are therefore easier

to solve; it is usually given at a specific Temperature. Physicists have derived clever ways of encoding certain classes of entangled quantum states into compact forms called tensor networks. An n- tensor M(i1,i2,….in) for all i={0,1} which stores all 2^n amplitudes of the quantum state.

The paper then goes on to discuss the works of the computer scientists. The computer scientist's approach to coming up with new algorithms is through mathematics rather than physics, each quantum gate is represented as nothing more than a matrix, and together they form algorithms. Each algorithm operates on quantum states that can be expressed using a certain type of mathematics. For example the DMRG is an algorithm over a set of tensor networks called MPS; it is used for studying 1D quantum many-body systems, to extraordinary precision. Matrix Product state (MPS) associates each qudit with a series of matrices A each acting on one element of the qudit of dimension D. Now that we have both algorithm and quantum state, we can do quantum simulation. However, large values of qudit dimension D are not computationally feasible due to large circuit size. So we are restricted to small qudit size. Physics often restrict math in practical situations. Other types of tensor networks used to represent quantum states are PEPS and MERA. An efficient representation of a quantum system would not be useful without the ability to compute properties of the system from this format. Which is why the strength of the MERA is that the expectation values of local observables of a quantum state can be efficiently computed.

Computer scientists studied the area law. Area laws are a way of formulating the fact that entanglement in natural quantum systems occur roughly on the boundary. The open question in this field is whether area law holds in dimensions larger than 1D. Commuting LH seem to be closer in complexity to classical world of constraint satisfaction problems. Commuting 2-LH, 3-LH, 4-LH are in NP. SAT is historically the first problem to be proven NP-complete. Now we want to define a quantum generalization of k-SAT and answer whether this generalization also lie in P when k=2. The Approximate Ground-Space Projection (AGSP) is used to prove Hasting's 1D area law for gapped systems. When the state is entangled, we can not do a simple projection onto the ground space because we can not bound the amount of entanglement. However an approximate projection will work fine; this is called the AGSP. Repeated applications of AGSP to product state allow us to approximate the ground state. It so happens that to prove the area law, it suffices to have a good AGSP along with a product state with constant overlap with the ground state. The first criteria imply the second

First I have described what condense matter physicists are doing, then I described what computer scientists are doing, now I will describe how the two fields interact. They interact because it takes actually building a circuit, which is physics, to prove that a problem, a k-LH problem, is solvable on a quantum computer, which is a question only interesting to a computer scientist. Kitaev showed that 5-local Hamiltonian problem is QMA complete. In the proof, a YES instance of k-LH is proof sent to the verifier circuit as the ground state of the local Hamiltonian in question. The verifier then runs a simple local version of phase estimation to roughly determine the energy penalty incurred by the given proof. It is first assumed that we have a k-local Hamiltonian as a sum of *r* local

Hamiltonians. When we apply the quantum verification circuit V to the proof $|\psi\rangle$, and measure the answer register, the thresholds probability of a and b are separated inverse polynomial in size of data. So it decreases very fast as number of local Hamiltonians increases. Namely the probability of a flip is inverse polynomial to number of local Hamiltonians, another word for circuit size. Thus proving the k-LH is within QMA

We prove that the probability of flipping the answer register of the proof is very high when we have a large number of local Hamiltonians (same as number of terms in index register.) Let P be a promise problem in QMA. The proof register contains the proof that V verifies. Using verifier circuit V, our goal is to define a 5-local Hamiltonian H that has a small eigenvalue if and only if there exist a proof causing V to accept with high probability. In the case where H has no such proof we want to prove that there are no small eigenvalues, so in the YES case the eigenvalue is upper bounded where as in the NO case, the eigenvalue is lower bounded. Both of these bonds turns out to be inverse polynomial circuit size and thus suffice to show that 5-LH is QMA-hard.

After proving that 5-LH is QMA hard, we want to prove 2-LH is QMA hard. Kitaev, Kempe, and Regev use perturbation theory prove that 2-local Hamiltonian is QMA hard. This proof is quite complicated and has many pieces that fit together. The beginning is Kempe and Regev's result that 3-LH is also QMA complete. This is obtained via a circuit to Hamiltonian construction similar to Kitaev's except that one first uses only a single qubit to refer to the clock in the propagation Hamiltonian—this reduces the Hamiltonian's locality from 5 to 3. To Prove that 2-LH is QMA hard, the approach is to show that a Karp or mapping reduction from an arbitrary instance of 3-LH to 2-LH. Thus we have just reduced from 5 to 2. Once we define the 2-local Hamiltonian H-tilde, the spectral analysis we have just performed follows the chain of relationships: H -> Heff->sum(z)->H-tilde. This illustrates the concept that we can approximate one circuit with another, besides proving that 2-LH is QMA-hard.

.