Measuring QPU Performance with the Ising Model[1]

Nolan McMahon under the supervision of Dr. Barry Sanders

Abstract

There exists at least one optimization problem that can be solved more efficiently (with a smaller time to solution metric) on D-Wave's quantum computer, the 2000Q, than with highly optimized classical simulated annealers. The optimization problem is to minimize the Ising model's Hamiltonian:

$$H = -\sum_{i=1}^{N} B_i s_i - \sum_{i}^{N} \sum_{j \in \text{nn(i)}} J_{ij} s_i s_j$$

Given:

$$\vec{B} = \{0.5143, 0.9702, 0.7975, 0.5248, 0.5397, 0.1735, 0.6969, 0.6760\}$$

$$\vec{J} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9094 \\ 0 & 0 & 0 & 0 & 0.9170 & 0.6233 & 0.8326 & 0.2928 \\ 0 & 0 & 0 & 0 & 0.3610 & 0.6507 & 0 & 0.6036 \\ 0 & 0 & 0 & 0 & 0 & 0.0636 & 0.2915 & 0.2554 \\ 0 & 0.9170 & 0.3610 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.6233 & 0.6507 & 0.0636 & 0 & 0 & 0 & 0 \\ 0 & 0.8326 & 0 & 0.2915 & 0 & 0 & 0 & 0 \\ 0.9094 & 0.2928 & 0.6036 & 0.2554 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The 2000Q shows two orders of magnitude lower time to solution than classical simulated annealers. As the instance size of this problem increases the difference in the time to solution grows, suggesting that D-Wave's 2000Q is better suited to solve this problem.

[1]All references are in accordance to the format prescribed by the Proceedings of the National Academy of Sciences (PNAS)

Introduction

D-Wave is a Canadian company which has produced a 2048 qubit quantum computer called the 2000Q. The 2000Q is open for public use; D-Wave is granting one free minute of computation time per month to anyone who wants it. This open access has inevitably led to many overly embellished, grandiose claims about the potential that the 2000Q, and quantum computers in general, have to change the world. The greater purpose of this project is to create and then quantitatively substantiate a more realistic claim about the 2000Q and its potential.

In order to make a substantiatable claim, objective criteria for classifying the performance of the 2000Q need to be defined. These performance measurements then need to be compared to the alternatives that have been available up and to this point, that is, the performance of non-quantum (classical) computers. These performance measurements will be made while solving scalable problems, that is, problems that can be made arbitarily large so that we can determine whether, as the problems grow, does the 2000Q have better performance?

Background

Computational problems can be classified based on the format of their solution. For example, the problem of determining the primality of a positive integer n is a decision problem. The solution to this problem is either "Yes" (n is prime) or "No" (n is composite). Another type of problem is one which seeks to extremize some function given a list of constraints. Such problems are called optimization problems.

The problem associated with minimizing the Ising model Hamiltonian is an example of an optimization problem (and is called the Ising problem from this point forward). The Hamiltonian for a system with N spins is given by:

$$H = -\sum_{i=1}^{N} B_i s_i - \sum_{i}^{N} \sum_{j \in \mathrm{nn(i)}} J_{ij} s_i s_j \tag{1}$$

where B_i is the magnetic field at the location of spin s_i, J_{ij} is the coupling strength between spins s_i and s_j, and $nn(i)$ is the nearest neighbour function which returns a set of all the spins s_j which have a non-zero coupling strength to spin s_i. Each spin s_i has a value in the set $\{-1, +1\}$. Minimizing the value of H, then, is a process of searching through all of the combinations of $\vec{S} = \{s_1, s_2, \ldots, s_N\}$ until the assignment of the spins s_i returns the smallest result for H given the supplied values:

$$\vec{B} = \{B_1, B_2, \ldots, B_N\}; \qquad \vec{J} = \begin{pmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,N} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ J_{N,1} & J_{N,2} & \cdots & J_{N,N} \end{pmatrix}$$

The solution is defined as the lowest energy value of the Hamiltonian H. The D-Wave 2000Q is a perfect candidate for optimization problems like the Ising problem, however, the hardware does pose some significant constraints.[1]

The processor inside of the 2000Q has 2048 qubits, each of which can be assigned a numerical weight value. Some of these qubits form pairs because they share a hardware coupling, which can take on one of sixteen strengths. Taken together, the qubits and couplings can be interpreted as sets of verticies and edges (respectively) of a sparse graphical lattice called a Chimera graph. The Chimera graph has a substructure of unit cells: groups of eight vertices broken up into two groups of four verticies connected in a bipartite manner. These bipartite unit cells are then connected to identical, adjacent unit cells to form the lattice. Inside of the 2000Q, there are 256 unit cells arranged in a square lattice, ergo sixteen rows and sixteen columns of unit cells[1]. See figure 1.

In order to solve the Ising problem on the 2000Q, the problem needs to be embedded onto the hardware in a systematic fashion so that, when the result is returned, it's meaning can be properly interpreted in terms of the Ising problem. Each individual qubit can take on one of two values, therefore it makes sense for them to represent $\vec{S}$. The weight of the qubits can represent $\vec{B}$ and the strength of the couplers can represent $\vec{J}$. After the solution is found, the assignment of $\vec{S}$ can be used in equation

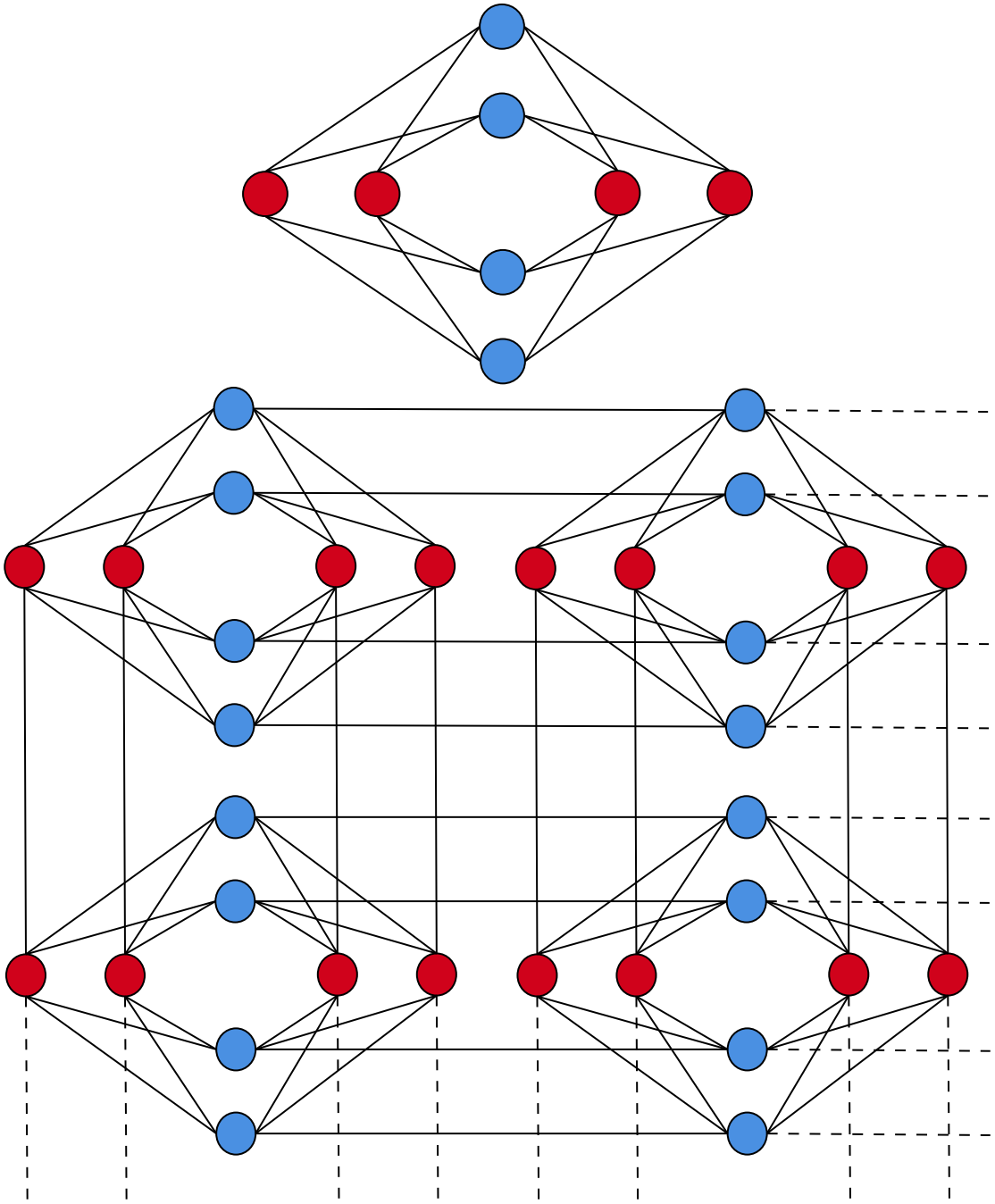

Figure 1: **Top:** A single unit cell of Chimera. Here, the circles represent qubits (verticies) and the lines represent available couplings (edges). Notice that none of the blue/red verticies share an edge with each other but each blue/red vertex is connected to every other red/blue vertex. **Bottom:** Four interconnected unit cells of Chimera. The blue/red verticies from one unit of Chimera are connected to the blue/red verticies of the neighbouring unit of Chimera. Dashed lines represent inter-Chimera unit cell connections that exist but were not drawn. D-Wave's 2000Q has sixteen rows and sixteen columns of Chimera unit cells.

(1) to determine the energy H. The existance (or rather, lack thereof) of the couplers between physical qubits on the Chimera graph produces a limitation on the selection of nearest neighbours that any one qubit has.

Now that a problem can be embedded onto the hardware, the next task is clarify how to distinguish between Ising problems. Solving the Ising problem involves finding the combination of spin values $\vec{S}$ such that H is minimized. However, the magnetic field strengths $\vec{B}$ and coupling strengths $\vec{J}$ must be given in order to do so. Therefore if $\{(\vec{B}, \vec{J})\}$ is defined to be the set of all combinations of $\vec{B}$ and $\vec{J}$, each element of the set will constitute a new problem. In order to keep track of all of these problems, an index k will be introduced such that solving the kth Ising problem involves minimizing the Ising Hamiltonian, given $(\vec{B}, \vec{J})_k$.

In object orientated programming languages, classes are used to define the properties and methods that can belong to objects. A class' constructor can be used to create an object which can be manipulated by a programmer separately from all other instantiated objects of that class. Problems $(\vec{B}, \vec{J})_k$ function analogously to classes. They can be instantiated with different properties by constructing instances. One important property of an instance is its size which is defined as the number of verticies on the Chimera graph activated by the given instance. The next question is, how does a problem increase its instance size?

Embedding the weights and strengths of problem $(\vec{B}, \vec{J})_k$ onto the D-Wave hardware instantiates the smallest instance of the problem. This embedding involves qubits in some number of units of Chimera. Define the set of Chimera units with qubits with non-zero weight under the instantiation of the smallest problem instance the unit cell of the problem $(\vec{B}, \vec{J})_k$. The instance size can be increased by replicating the embedding of the smallest instance size into the Chimera unit cells which are directly adjacent to unit cell of problem $(\vec{B}, \vec{J})_k$ and connecting the two problem unit cells together. This embedding of a problem must be connected, meaning that, starting from any non-zero weighted vertex, you could walk along edges with non-zero strength to get to any other non-zero weighted vertex. If it is not connected,

the instance size of problem $(\vec{B}, \vec{J})_k$ has not been increased, a different problem has just been instantiated! Replicating the smallest instance once will double the size of the new instance. Notice that for any given problem with a smallest instance size greater than one, not all instance sizes are realizable. All instance sizes are positive integer multiples of the size of the smallest instance.

As an example, suppose that I choose the problem that is defined by:

$$\vec{B} = \{2, 2, 2, 2, 0, -2, -2, 0\}; \qquad \vec{J} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 2 & 2 & -2 & 0 & 0 & 0 & 0 \\ 2 & -2 & -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To create the smallest feasible instance of this problem, I start by embedding the $\vec{B}$ and $\vec{J}$ values onto the Chimera graph (which has been labeled in figure 2). This particular instance can fit inside of a single unit of Chimera. When I want to increase the size of the problem to the next smallest feasible size, I duplicate the smallest instance on the unit of Chimera immediately to the right of the first one. I can now use the inter-cell couplings to connect these two units together. I can create an even bigger instance by duplicating the smallest instance below the original instance and to the right of the original instance (see figure 3). In this way, I can continue to make bigger and bigger instances until I reach the boundaries of the D-Wave hardware itself. The smallest instance has six vertices with non-zero weight. Thus, the size of this instance is six. Similarly, the second smallest instance has a size of twelve, the third smallest instance has a size of eighteen, and so on. Solving for the minimum energy of each of these instances is now just a matter of finding the combination of spins $\vec{S}$ for which the value of the Hamiltonian in equation (1) is minimized.

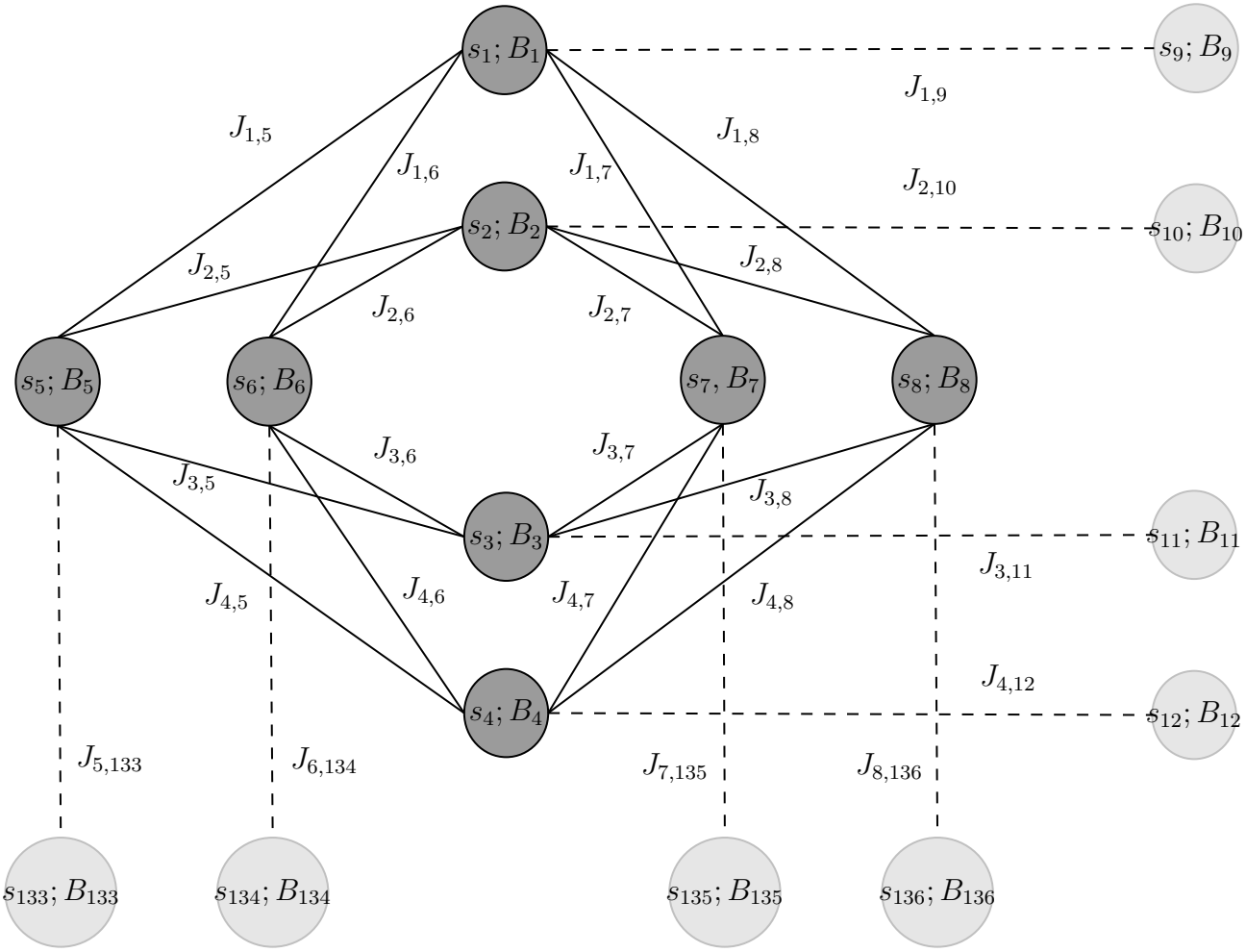

Figure 2: A labeled unit of Chimera for the purposes of embedding the example problem.

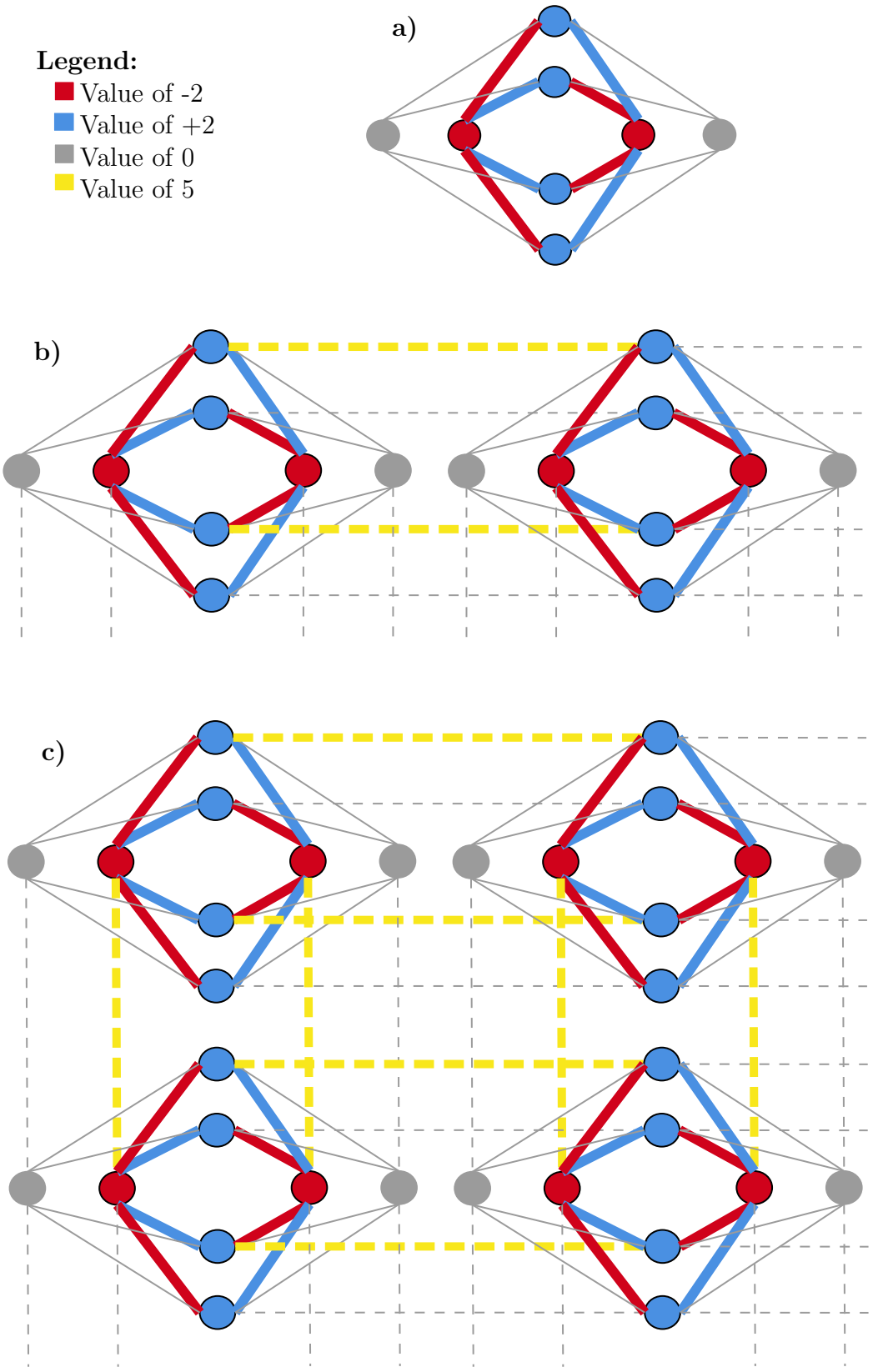

Figure 3: A demonstration of how the problem instances grow in size.

Now that a set of problems has been defined, a means by which to quantify the performance of each problem needs to be determined. There is no single best way to judge the performance. For this project, we have chosen to define the time-to-solution or TTS as our metric of choice. The general formulation of TTS is defined as follows[2]:

$$\text{TTS} = t_A \frac{\log(1-P)}{\log(1-P_s)} \tag{2}$$

Where t_A is the annealing time, P is the probability of reaching the ground state after some r annealing repetitions, and P_s is the probability of reaching the ground state on the current annealing repetition. The way in which t_A is calculated varies from solver to solver but all implementations are making measurements of the same quantity[3].

Methods

Now that the problem has been defined, how is it solved?

D-Wave's Quantum Annealer

The D-Wave 2000Q processor contains 2048 qubits and has a Hamiltonian[1]:

$$\text{H}_{\text{Ising}} = \underbrace{-\frac{A(s)}{2}\sum_i \hat{\sigma}_x^{(i)}}_{\text{initial Hamiltonian}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{i,j}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right)}_{\text{final Hamiltonian}} \tag{3}$$

where $\sigma_{x,z}^{(i)}$ are Pauli matricies operating on a qubit q_i, h_i is the weight of the ith qubit, $J_{i,j}$ is the strength of the coupler between qubits q_i and q_j, $s = \frac{t}{t_f}$ is a parameter which is a function of both the current time t and the total annealing time t_f, $A(s)$ is a monotonically decreasing function of parameter s, and $B(s)$ is a monotonically increasing function of parameter s.

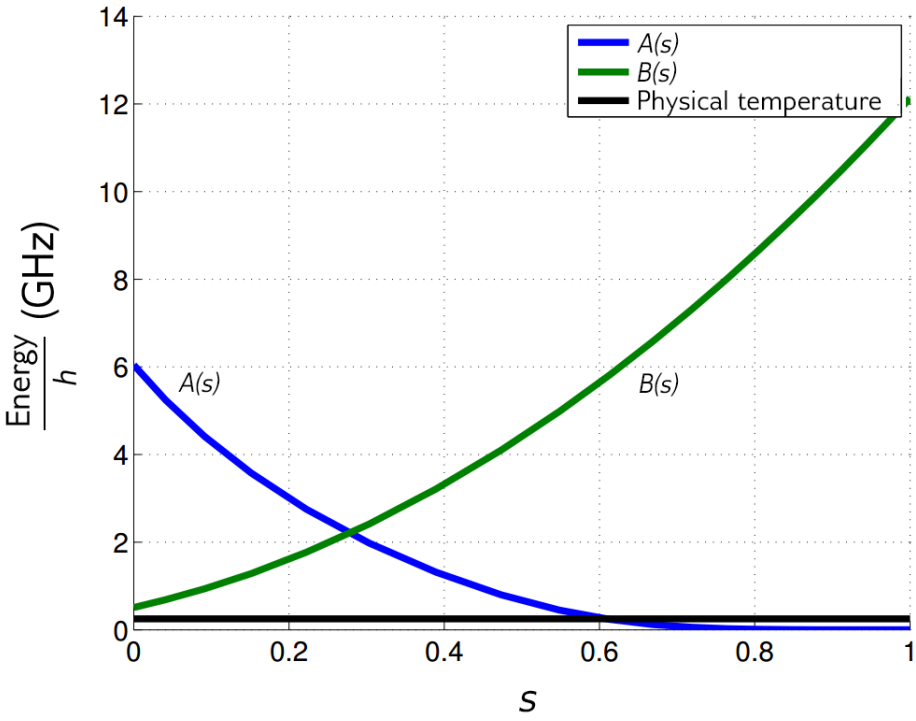

Figure 4: Annealing schedule for the 2000Q. $A(s)$ is the coefficient for the initial Hamiltonian and $B(s)$ is the coefficient for the final Hamiltonian[1].

Before the annealing process begins, $A(s) \gg B(s)$ which puts all of the qubits in a superposition of two states, zero and one, such that the Hamiltonian is minimized. As the annealing process starts, the final Hamiltonian is slowly introduced as the qubits are assigned their weights (h_i), the couplers are assigned their strengths ($J_{i,j}$) and the value of $B(s)$ increases. By changing the values of $A(s)$ and $B(s)$ slowly enough, the 2048 qubits are likely to stay in the ground state of the Hamiltonian H_{Ising} for the entire duration of the anneal, that is, until $A(s) \ll B(s)$ and the Ising Hamiltonian is represented entirely by the final Hamiltonian. Notice that the form of the final Hamiltonian is the same as the problem Hamiltonian in equation (1). As such, reading off the values of the spins in the final Hamiltonian can reveal the solution to the Ising problem.

As the annealing process runs, the energy gap between the ground state of the Ising Hamiltonian and the higher energy states decreases to some minimum. The system has a non-zero probability of transitioning from the ground state to a higher energy state throughout the entire annealing process. This probability will increase because of thermal fluctations in the system and if the annealing process is done

too quickly (the value of $A(s)$ is decreased too fast and the value of $B(s)$ is increased to fast). As such, it is important not to complete the anneal too quickly since you are more likely to leave the ground state.

The performance of the quantum annealer is measured by time to solution (TTS). It is defined as:

$$\text{TTS} = t_{\text{ar}} \frac{\ln(1-P)}{\ln(1-P_s)} \tag{4}$$

where t_{ar} is the annealing time per repetition, P is the probability of reaching the ground state after some r repetitions, and P_s is the probability of reaching the ground state on the current repetition. Throughout the entity of this paper, P is set to 0.99, that is, the probability of reaching the ground state (the minimum energy value of the Hamiltonian H after some number r repetitions of the anneal) is 99%.

In order to send problems to the 2000Q, the HTTP protocol needs to be used to communicate with D-Wave's solver API. The API is very flexible, allowing for the submission of weights and strengths, as well as several dozen other input parameters. The problem status, solution, timing information, etc. is stored in D-Wave's database and can be remotely retrieved on demand.

Simulated Annealing

Simulated annealing was modeled on the cooling of a material in a heat bath. Annealing is an iterative process. Suppose that you have some system in state s. State s has some energy associated with it E. Each annealing interaction starts with the selection of some neighbouring state s^*. A probability that the present state will be exchanged with this new state is then calculated (this is called a transition probability). The transition probability is a function of both the system's temperature T and the energy of the neighbouring state s^*[4].

$$P_{\text{Transition}}(E^*) = \begin{cases} 1, & E^* < E \\ \exp\left(\frac{-E^*}{k_B T}\right), & E^* \geq E \end{cases} \tag{5}$$

If $E^* \geq E$, a random number r in the set $[0, 1)$ is chosen. If $r < P_{\text{Transition}}(E^*)$, the current state

is changed from s to s^*. At the end of the annealing iteration, the temperature is decreased by some amount defined by the annealing schedule. The annealing schedule is a function which describes the temperature of the system as a function of the annealing iteration count.

Notice that at higher temperatures, if $E^* \geq E$, the system is more likely to undergo a transition from s to s^*. As T decreases, the system gradually settles into lower energy configurations since the probability that a transition to a higher energy will occur continues to decrease.

Simulated annealers often end up in local minima of functions that they are trying to minimize. The way around this problem is to run the annealer multiple times on each instance. The odds of finding the global minimum are not 100% so annealing multiple times gives multiple chances to reach that desirable global minimum.

Writing a simulated annealer is not a difficult task, but writing an *efficient* one most certainly is. To ensure that a comparison is made between the 2000Q and the best simulated annealer currently available, simulated annealing codes which have been optimized and recently published were used[5]. This paper came with a dozen or so solvers, each of which is optimized slightly differently in order to provide better performance on different graphical structures. The improvements that were made (over some naive implementation) include: using faster random number generators whose results are reused over multiple iterations of the annealing process, picking neighbouring states deterministically instead of randomly, etc.

The authors[5] used time to solution to measure the performance of their simulated annealing codes. They defined time to solution as:

$$\text{TTS} = \frac{SN}{f} \frac{\ln(1-P)}{\ln(1-P_s)} \tag{6}$$

where S is the number of sweeps performed in the simulation (where a sweep is defined as one pass through all of the spins in the model[6]), N is the instance size, f is the number of attempted spin

updates per second, P is the probability of reaching the ground state after some r repetitions, and P_s is the probability of reaching the ground state on the current repetition.

These simulated annealing codes were written in C++ while the rest of the codes related to this project were written in Python. Solving problem instances using these codes can be done by calling these C++ codes from Python, passing the weights, strengths, and other input parameters, and then pulling the output back into Python for analysis.

Results

Consider the following problem:

$$\vec{B} = \{0.5143, 0.9702, 0.7975, 0.5248, 0.5397, 0.1735, 0.6969, 0.6760\}$$

$$\vec{J} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9094 \\ 0 & 0 & 0 & 0 & 0.9170 & 0.6233 & 0.8326 & 0.2928 \\ 0 & 0 & 0 & 0 & 0.3610 & 0.6507 & 0 & 0.6036 \\ 0 & 0 & 0 & 0 & 0 & 0.0636 & 0.2915 & 0.2554 \\ 0 & 0.9170 & 0.3610 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.6233 & 0.6507 & 0.0636 & 0 & 0 & 0 & 0 \\ 0 & 0.8326 & 0 & 0.2915 & 0 & 0 & 0 & 0 \\ 0.9094 & 0.2928 & 0.6036 & 0.2554 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Each of the weights and strengths were sampled from a uniform random distribution between zero and one. Graphically, three smaller instances of this problem can be represented on the Chimera graph as shown in figure 5.

Notice that I am not representing every instance size. This problem has a unit size of one unit of Chimera. This means that I can construct an instance of size four using four units of Chimera arranged

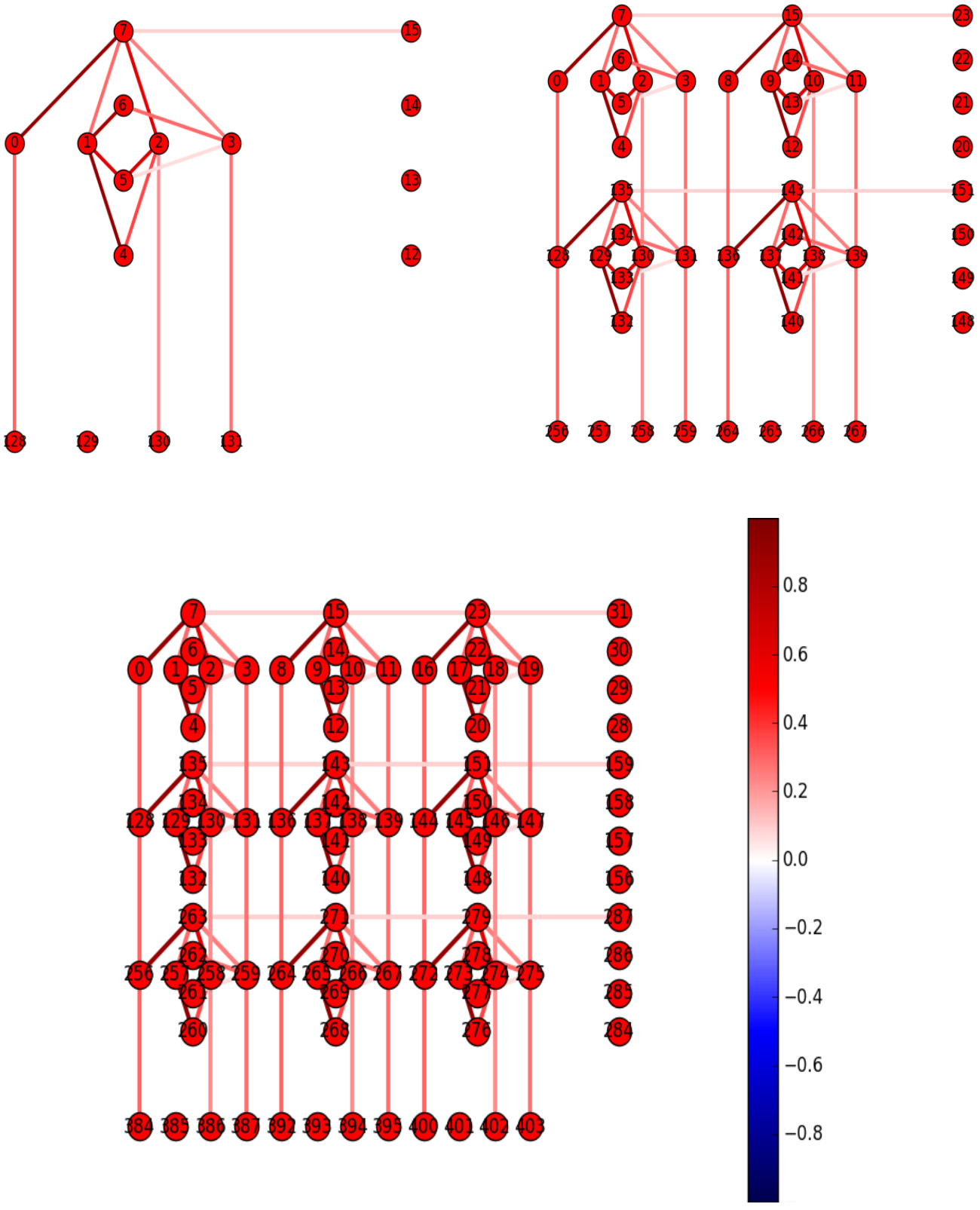

Figure 5: Graphical representations of the problem instances of the first three sizes.

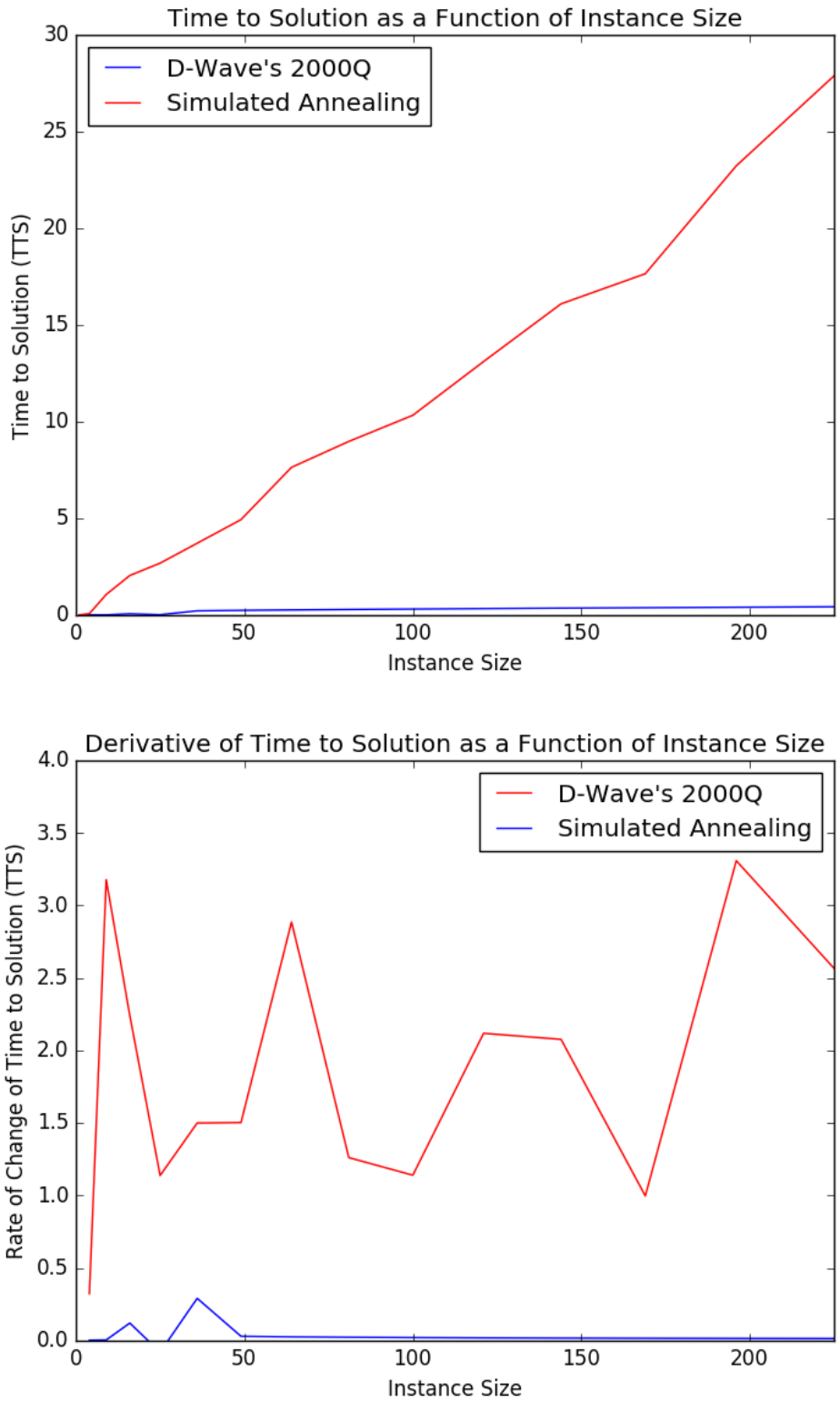

Figure 6: **Top:** Time to solution for both the simulated annealer and the 2000Q for the problem as given. **Bottom:** The slope of the above graph as a function of instance size. Notice that the slope of the simulated annealer is consistently above the slope of the 2000Q. This leads to the ever increasing gap in the time to solution seen in the above plot.

in a two by two square. I can continue in this way by constructing only instances which are square on the Chimera grid. When I solve the problem instances on both the D-Wave 2000Q and the simulated annealer, I find that the time to solution on the 2000Q is far less for instances of every size than for the simulated annealer.

In figure 6, the time to solution is plotted against the instance size. Simulated annealing shows itself to be much less efficient than the 2000Q since the gap between the time to solution for both solvers continues to grow as the instance size increases. This growth is exemplified in the second plot in figure 6 where the slope of the first plot in figure 6 is shown. The plot represents the change in the time to solution with respect to the instance size for both solvers. Smaller slope means that there is a smaller increase in the time to solution between instances of increasing size.

Conclusions

D-Wave's 2000Q was able to outperform an optimized simulated annealer on a randomly generated problem. The time to solution was roughly two orders of magnitude less and the scaling looked promising. The existence of this problem, and the lack of a concerted, guided effort to find it would suggest that either the selection of this problem was an unbelieveable stroke of luck, or that a significant percentage of all Ising problems can be solved with a smaller time to solution on the 2000Q than with simulated annealing.

Future Direction

Simulated annealing is not the necessarily the best or only classical method which is used to solve optimization problems like the Ising problem. Simulated quantum annealing is another possibility. Evaluating these Ising instances using a simulated quantum annealer may very well reveal that the 2000Q is not better than all classical algorithms. Therefore, implementing an optimized simulated quantum annealer would be a good next step in this project.

Furthermore this process depends upon the assumption that the answers that the 2000Q and the

simulated annealer return are both correct. But, both methods are inherently probabilistic. They were both run multiple times on each problem and the mode was assumed to be the correct answer, but, it would be wise to include a deterministic solver. Not as a high performing competitor, but as a validity check. One possibility that looks promising is to convert the Ising problem into a weighted max-2-sat problem, solve it, and convert the answer back as there are many good SAT problem solvers.

Acknowledgements

This work was done as part of the physics 598 honours research thesis course. I want to thank Dr. Barry Sanders for supervising my work on this project, holding many meetings to discuss my project and its status, and providing guidance to the direction of the project as a whole.

I also want to give thanks to Seyed Shakib Vedaie and Archismita Dalal for meeting with me on an almost weekly basis to propel this project forward. Their input on this project was absolutely critical to its success.

Last but not least, I want to thank Austin Nhung, a fellow physics 598 student, for his input into this project. While we were working on distinct aspects of this project, without his code, this work would not have been possible.

References

[1] Inc. DWS (2019) D-wave system documentation (https://docs.dwavesys.com/docs/latest/index.html).

[2] Troels F. Rønnow, Zhihui Wang JJSBSVIDWJMMDALMT (2014) Defining and detecting quantum speedup. *Science* 345:420–424.

[3] Vasil S. Denchev, Sergio Boixo SVINDRBVSJM, Neven H (2016) What is the computational value of finite-range tunneling? *Physical Review X* 6.

[4] Kathryn A. Dowsland JMT (2012) *Handbook of Natural Computing.* (Springer Publishing), pp. 1623–1655.

[5] S.V. Isakov1, I.N. Zintchenko TRMT (2015) Optimised simulated annealing for ising spin glasses. *Computer Physics Communications* 192:265–271.

[6] Sergio Boixo, Troels F. Rønnow SVIZWDWDALJMMMT (2014) Quantum annealing with more than one hundred qubits. *Nature Physics* 10:218–224.