Chapter 6

Quantum optimization and applied algorithms

6.1 Introduction

Quantum computing has advanced significantly in recent years, and while no conclusive quantum advantage has been demonstrated at the time this document has been written, real world applications appear to be on the horizon. There are several potential areas for near term applications of quantum computing, of which optimization is one promising area. For this chapter, we will investigate how quantum tools can be used to solve classical optimization problems.

The first step in quantum optimization is to map the problem to a physical model which can be encoded on a quantum computer. In practice, the most common choice is the Ising model, which is known to be able to map all NP-hard[1] problems:

$$H_{\text{Ising}} = \sum_{k=1}^{n} \sum_{j=k+1}^{n} J_{kj} \sigma_k^z \sigma_j^z + \sum_{j=1}^{n} h_j \sigma_j^z \tag{6.1}$$

with $\sigma_j^z = \left(\bigotimes_{k=1}^{j-1} I_2 \right) \otimes \sigma^z \otimes \left(\bigotimes_{k=j+1}^{n} I_2 \right)$, where

$$I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{6.2}$$

is the 2×2 identity matrix and

$$\sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{6.3}$$

The $\otimes$ symbol indicates a tensor product, which is reviewed in Appendix 1 along with the $\bigotimes_{k=1}^{j-1}$ notation. The coupling strengths J_{kj} form a matrix, and the field strengths h_j form a vector specifying a particular Ising model. Optimization problems can be encoded in the matrix J and the vector h in ways which will be explained in section 6.3. Since H_{Ising} is a Hamiltonian and the operators σ_j^z have no physical dimensions,

[1] Non-deterministic polynomial-time hard (often mistakenly referred to as 'non-polynomial hard'). The exact definition of this problem class is not important for this project, but this is the 'hardest' class of conventional optimisation problems, and it is suspected (but not proven) that no efficient algorithms exist for this class of problems.

The J_{kj}'s and h_j's should have the physical dimensions of an energy. However, it is customary, in this area of Physics, to treat this Hamiltonian as having no physical dimensions either, which simplifies the description. Accordingly, the J_{kj}'s and h's will be assumed to be pure numbers in the following.

When J and h encode a problem, they encode it in such a way that the minimum energy with respect to H_{Ising} is the best solution. When represented as a matrix, H_{Ising} is a $2^n \times 2^n$ diagonal matrix where each diagonal entry gives the energy of a classical bitstring from $|00...0\rangle$ to $|11...1\rangle$. (By classical bitstring, we mean a state which is labelled by a binary number and can represent the state of a classical system.) Calculating each of the entries on the diagonal, hence the corresponding energies, might not require much computational efforts. However, writing the entire matrix out in this way is only possible for very small problems. To see why, consider a 100-bit optimization problem, which is actually still quite small compared to what one typically encounters in the 'real world'. The number of energies will be $2^{100} \sim 10^{30}$. In Physics terms, this is more than the number of water molecules in a large tanker truck full of water. Counting the molecules in such a truck is clearly not possible. Finding the solution to such an optimization problem by just calculating all the energies and taking the lowest is similarly impossible: even if you had a fast computer which could check one solution every nanosecond and started at the beginning of the universe, you would currently be less than one thousandth of the way to being done.

6.2 The Quantum Adiabatic Algorithm

Clearly, there are better ways (i.e., *algorithms*) to solve optimization problems than just checking every possibility. Also, in most cases you just need a good solution, not the *best* solution. Coming up with clever classical ways to solve optimization problems (e.g., genetic algorithms, swarm algorithms and many others) is a huge area of active research, but these problems could also potentially be solved with the help of quantum mechanics. So far, all we have shown is how to state an optimization problem as a Hamiltonian, we have not shown how to add quantum effects. Consider the effect of adding single bit flip operations to H_{Ising}, to obtain the Hamiltonian of a *transverse field Ising model*:

$$H(t) = -A(t)\sum_j \sigma_j^x + B(t)H_{\text{Ising}} \tag{6.4}$$

where $A(t)$ and $B(t)$ are time-dependent controls, and $\sigma_j^x = \left(\bigotimes_{k=1}^{j-1} I_2\right) \otimes \sigma^x \otimes \left(\bigotimes_{k=j+1}^{n} I_2\right)$ where

$$\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{6.5}$$

is the usual Pauli matrix. (The functions $A(t)$ and $B(t)$ are often given physical units of energies, so as to give $H(t)$ the physical dimensions of an energy; however, for simplicity we won't do this. Accordingly, we will take the "time" t to be dimensionless, too.)

If $B = 0$ and $A > 0$, then the ground state $|\phi_0(A > 0, B = 0)\rangle$ of Eq. (6.4) will just be the highest energy state of each σ^x individually. In other words,

$$|\phi_0(A > 0, B = 0)\rangle = \bigotimes_{j=1}^{n} |+\rangle\,, \tag{6.6}$$

where

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{6.7}$$

On the other hand, when $B > 0$ and $A = 0$, then by definition $|\phi_0(A = 0, B > 0)\rangle$ will be the solution to the optimization problem which has been encoded into J and h in Eq. (6.1).[2]

It is important to note that $H(t)$ acts and $|\phi_0\rangle$ lives in a space of 2^n-component column vectors. This space can be spanned by the 2^n-component column vectors $|0\rangle$, $|1\rangle$, $|2\rangle$,..., $|2^n - 1\rangle$, where $|i\rangle$ is a column vector of zeros except the i-th element which is 1. In terms of these vectors,

$$|\phi_0(A > 0, B = 0)\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle . \tag{6.8}$$

Note, also that the exact ground state of H_{Ising} is one of these vectors.

Based on the statements in the previous paragraph, and some basic quantum mechanics, there is a general quantum algorithm for optimization problems, assuming a physical system which can implement the transverse field Ising model with sufficient controls is available. The *adiabatic theorem of quantum mechanics* states that a quantum system initially in the ground state of a time-dependent Hamiltonian remains in the ground state of the Hamiltonian as this operator changes, *provided the change is slow enough* (some other technical criteria — which transverse Ising systems always satisfy — must also be met). If we start in $|\phi_0(A > 0, B = 0)\rangle$ and change A and B *slowly enough* until we end with $B > 0$ and $A = 0$, we will find a final state which is almost equal to $|\phi_0(A = 0, B > 0)\rangle$, which is the solution to our problem. [It would normally be equal in the limit where $A(t)$ and $B(t)$ vary infinitely slowly. For finite time scales, the final state will be a superposition of $|\phi_0(A = 0, B > 0)\rangle$ with a number of other states, but for a sufficiently slow variation $|\phi_0(A = 0, B > 0)\rangle$ will have a much larger weight in this superposition than any of the other states.]

Solving an optimization problem along these lines would involve implementing $H(t)$ on a quantum platform, start with $B(t) = 0$ and $A(t) > 0$ and the state $|\phi_0(A > 0, B = 0)\rangle$ at $t = 0$, slowly vary $A(t)$ and $B(t)$ from $t = 0$ to $t = t_{\text{fin}}$, and measure the state at that time in the $|j\rangle$-basis (i.e., use a measuring apparatus which would find the system in one of the $|j\rangle$ states).

This techniques is known as *adiabatic quantum computing* (AQC) or the *quantum adiabatic algorithm* (QAA). Note that, while AQC definitely works, if you change the Hamiltonian slowly enough, it won't be useful in practice if it takes much longer than classical computers solving the same problem. In practice we would still be happy if it doesn't find the absolute best solution but finds a better solution than classical computers could obtain, or if it finds a solution a classical computer can also obtain but finds it much faster.

6.3 Encoding Optimization problems into Hamiltonians

To explain how optimization problems can be mapped to the J and h terms in Eq. (6.1), we will consider a simple case, known as the maximum independent set problem. The easiest way to think of maximum independent set is to think of a *graph* with vertices connected by edges, for example, the one shown in Fig. 6.1. The goal of finding the maximum independent set[3] is to colour in as many vertices as possible such that no two coloured vertices share an edge.

[2] If there is a tie for the best solution, then $|\phi_0(A = 0, B > 0)\rangle$ could be a superposition of the corresponding states.

[3] Not to be confused with *maximal* independent set, which is not a hard problem. A maximal independent set is one that cannot be extended by colouring one more vertex without having two vertices sharing an edge. Fig. 6.1(b) is an example of maximal independent set.

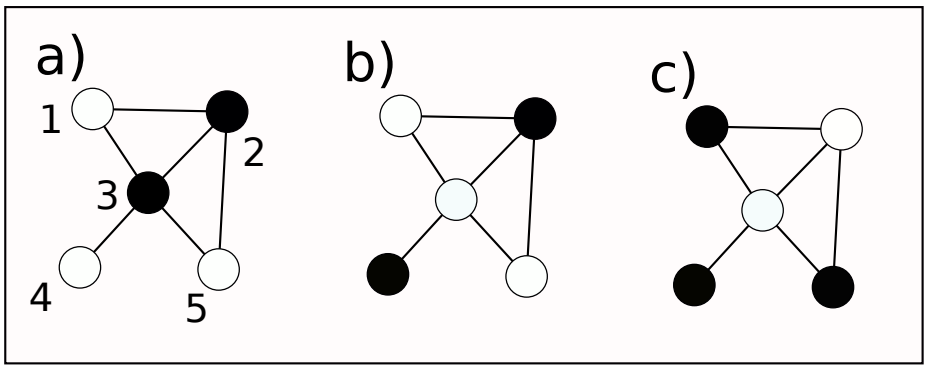

Figure 6.1: A graph with different sets of vertices coloured in black. a) A set of vertices which are not independent. b) A set of vertices which are independent, but is not the maximum independent set. c) The maximum independent set of vertices for this graph. The numbering scheme used in Eq. (6.10) is shown in a).

While finding maximum independent set may seem like a silly colouring problem, it could actually come up in the real world. For instance, imagine a stock broker wants to build a large diverse portfolio of stocks, including pairs of stocks whose values are likely to be highly correlated (maybe stocks of two different companies which produce the same product or rely on the same resource). This problem maps to a graph, where each vertex is a stock and the edges represent high correlation between stocks. Finding the largest portfolio of uncorrelated stocks in this example is exactly the maximum independent set problem.

To express the maximum independent set problem as an Ising model, we first need to think how to encode the concept of an independent set. To do this, first construct a two qubit Hamiltonian which penalizes (higher energy) the $|11\rangle$ state relative to the $|00\rangle$, $|01\rangle$, and $|10\rangle$ states. This constraint is achieved by

$$H_2 = -\sigma^z \otimes I_2 - I_2 \otimes \sigma^z + \sigma^z \otimes \sigma^z = -\sigma_1^z - \sigma_2^z + \sigma_1^z \sigma_2^z, \tag{6.9}$$

where the second form drops the identity operators and the $\otimes$ symbols for brevity. This Hamiltonian ensures the lowest energy state does not contain two ones.

Consider a graph defined by the upper triangular adjacency matrix M, such that a 1 entry indicates an edge between corresponding vertices and a 0 entry indicates no edge (see Appendix 2). If the qubits represent vertices, and the qubit state $|1\rangle$ means that this vertex is coloured, we can use operators like the Hamiltonian of Eq. (6.9) to penalise adjacent vertices that are both coloured. By defining $J = M$ and $h_k = -\sum_j (M_{kj} + M_{jk})$, a Hamiltonian of the form of Eq. (6.1) will enforce independent sets. In other words, when expressed as a bitstring, if the ones in a state form an independent set on the graph defined by M, they will get one energy, but they will get a higher energy if they do not. Try writing it out by hand for a three qubit example, such as o–o–o, see figure 6.3 and Appendix 3.

Finally, to construct a *maximum* independent set Hamiltonian, rather than just an independent set Hamiltonian, we need to give a lower energy to states with more qubits in the $|1\rangle$ state. To do this, we now set $h_k = -(\sum_j M_{kj} + M_{jk}) + \kappa$ (a Greek kappa, not k) where $0 < \kappa < 1$.

There are many more complicated ways of encoding optimization problems into Ising Hamiltonians, but maximum independent set is one of the simplest, and gives a flavour of how this process works.

6.4 Time-dependent Hamiltonian simulation

For the milestone, you will be simulating the quantum adiabatic algorithm solving a small instance of maximum independent set. To start with, consider the graph depicted in Fig. 6.1, which is defined by the following adjacency matrix (see Appendix 2)

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{6.10}$$

First, construct H_{Ising} for this graph, and verify that the lowest energy state is indeed the maximum independent set $|10011\rangle = |19\rangle$. In vector form, this is a column vector of length 2^5 with a one in the 19^{th} position $(18\,\text{zeros}\,, 1, 13\,\text{zeros})^T$ (see Appendix 3).

Once you have verified that you have constructed the classical problem Hamiltonian correctly, the next step is to simulate an adiabatic algorithm solving the problem. To simulate the adiabatic algorithm, you have to simulate evolution with a time dependent Hamiltonian. How the states evolve in time can be described as governed by an evolution operator $U(t,0)$ so that $|\psi(t)\rangle = U(t,0)\,|\psi(t=0)\rangle$. This evolution operator can be written as an infinite product of matrix exponentials: For an evolution from $t = 0$ to $t = t_{\max}$,

$$U(t_{\max},0) = \lim_{q\to\infty} \mathcal{T} \prod_{j=1}^{q} \exp\left\{-i\frac{t_{\max}}{q}\, H\left(\frac{j\,t_{\max}}{q}\right)\right\}, \tag{6.11}$$

where $\mathcal{T}$ indicates that the product is time ordered. If we are willing to accept some numerical error, we can set q to be large but not strictly infinite, thus obtaining a discrete version of the continuous-time evolution in a form convenient for numerical calculations. If we start in state $|\psi(t=0)\rangle$, then the state at time $k\,t_{\max}/q$ can be written as

$$\left|\psi\left(t = \frac{k\,t_{\max}}{q}\right)\right\rangle \approx \mathcal{T} \prod_{j=1}^{k} \exp\left\{-i\frac{t_{\max}}{q}\, H\left(\frac{j\,t_{\max}}{q}\right)\right\} |\psi(t=0)\rangle\,, \tag{6.12}$$

where we use the convention that the right-hand most term of the product is that with $j = 1$ and the left-hand most that with $j = k$ (thus term j acts on the left of term $j-1$, which itself acts on the left of term $j-2$, etc.). Mathematically, exponentiating a matrix is well defined (at least for square matrices) through the power series expansion of the exponential function. Fortunately, Python has a very efficient function to exponentiate matrices: `scipy.linalg.expm`.

6.5 Milestone project

For the milestone you will simulate an adiabatic algorithm which solves the maximum independent set problem on the graph defined by the adjacency matrix given in Eq. (6.10), using Eq. (6.12). For this exercise, set $A(t) = 1 - t/t_{\max}$ and $B(t) = t/t_{\max}$ where $t_{\max}$ is the total runtime of the algorithm. Plot the probability of 'success' (i.e., the probability that the measured result is the maximum independent set) versus $t/t_{\max}$ for several different values of $t_{\max}$. This plot should include both values of $t_{\max}$ for which the success probability is barely changed from random guessing, values for which the success probability

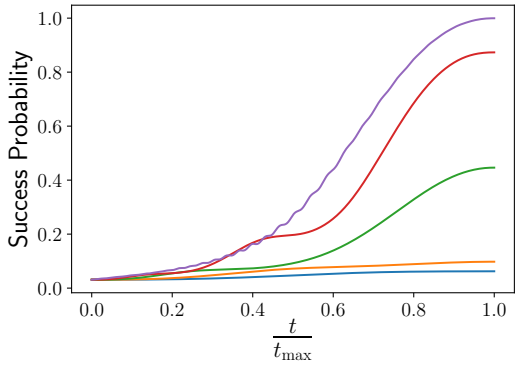

Figure 6.2: Success probability versus time for solving the maximum independent set problem on the graph given in Fig. 6.1 and Eq. (6.10) for different total runtimes: $t_{\max} = 1$ (blue), $t_{\max} = 2$ (orange), $t_{\max} = 5$ (green), $t_{\max} = 10$ (red), $t_{\max} = 100$ (purple). This plot was created using $\kappa = 1/2$.

is greater than 0.95, and at least one value where the probability is somewhere in between. Find an acceptable value of q by trial and error (q does not have to scale with $t_{\max}$). You may aim at reproducing Fig. 6.2, which shows the success probabilities versus time for different total runtimes.

6.6 Milestone extensions

There are a wide variety of possible extensions which would be appropriate, including many not listed here. All of the listed ones are are suitable for Level 3 computer projects, but those which are likely to be more challenging are marked with a †):

- Map other problems such as maximum 2 satisfiability using the mapping from this experimental paper [1] and † more complicated problems such as those described in [2].
- Apply sparse matrix techniques to simulate adiabatic quantum computing on larger systems using `scipy.sparse.linalg.expm_multiply`, possibly also looking at problems beyond maximum independent set.
- Investigate the effect of different annealing schedules, in other words different functional forms of $A(t)$ and $B(t)$ rather than just the linear example used for the milestone, for example:

$$A(\alpha, s', t) = \frac{1 - t/t_{\max}}{\alpha + (1-\alpha)\,(1 - t/t_{\max})\,(1-s')^{-1}}\,, \tag{6.13a}$$

$$B(\alpha, s', t) = \frac{t/t_{\max}}{\alpha + (1-\alpha)\,t/t_{\max}\,(s')^{-1}}\,, \tag{6.13b}$$

  where $0 < s' < 1$ controls where the algorithm slows down (it translates to a value of $t/t_{\max}$ in the original $A = t/t_{\max}$, $B = 1 - t/t_{\max}$ parametrization) and $0 < \alpha \leqslant 1$ controls how much it slows down. How does this affect the performance? Is it better to slow down when the energy gap between the ground and first excited state is large, or when it is small?
- Reproduce some of the results obtained in early proof-of-principle problems for adiabatic quantum computing [3]. Note that some of these problems are not formulated as Ising models.

- Explore QAOA (quantum approximate optimisation algorithm) rather than adiabatic protocols. In QAOA, $-\sum_j \sigma_j^x$ and H_{Ising} are applied in an alternating fashion rather than simultaneously — see Refs. [4, 5].
- Explore quantum walks on graphs [6](† possibly also including marked states and decoherence [7]). Quantum walks have recently been considered as a tool to solve optimization problems [8], but more results are known for walks on graphs.
- † Write code to perform path integral quantum annealing (PIQA) [9], which can be applied to much larger problems than the matrix simulation methods used here. Potentially reproduce a version of the PIQA plots in [10].

Warning: the use of the terms *adiabatic quantum computing* versus *quantum annealing* is not standardized in the literature, and different authors use these terms differently. Read carefully, to understand what is actually being done in a paper.

Appendix 1: A review of tensor products

Tensor products provide a powerful mathematical tool for a range of Physics models. The basic idea behind a tensor product is that it separates operations occurring on different degrees of freedom of a physical system. Tensor products are required to complete the milestone. Note that the Python function `numpy.kron` performs tensor products. Terminology in Python for matrix operations is not the same as usually used in Physics, and you should always check that it is actually doing what you want. In particular, matrix multiplication using * is usually elementwise; you need to use something like `numpy.dot` to do what we consider normal matrix multiplication.

Python has functions to perform all the operations you need (use them, they are very efficient implementations). However, it is necessary to understand how they work, in order to check that your code is correct. Effectively, the action of the tensor product $a \otimes b$ is to replace each element of a, a_{jk}, with the matrix $a_{jk} \times b$. Since each element is replaced by a matrix, the total size of the resulting matrix is length$(a) \times$ length(b). Unlike standard matrix multiplication, where the multiplication dimension needs to match, tensor products can be performed between pairs of matrices (or vectors) of any size. Like all types of multiplication, tensor products are *associative*, $a \otimes b \otimes c = a \otimes (b \otimes c) = (a \otimes b) \otimes c$.

As an example, consider the smallest non-trivial matrix tensor product, which is a tensor product of two 2×2 matrices, such as matrices representing quantum mechanical operators acting on qubits (blue ket vectors are added as a visual aid) :

$$a \otimes b = \begin{matrix} \\ \langle 0| \\ \langle 1| \end{matrix} \begin{matrix} \begin{matrix} |0\rangle & |1\rangle \end{matrix} \\ \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \end{matrix} \otimes \begin{matrix} \\ \langle 0| \\ \langle 1| \end{matrix} \begin{matrix} \begin{matrix} |0\rangle & |1\rangle \end{matrix} \\ \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} \end{matrix} =$$

$$\begin{matrix} \\ \langle 00| \\ \langle 01| \\ \langle 10| \\ \langle 11| \end{matrix} \begin{matrix} \begin{matrix} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{matrix} \\ \begin{pmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{01}b_{00} & a_{01}b_{01} \\ a_{00}b_{10} & a_{00}b_{11} & a_{01}b_{10} & a_{01}b_{11} \\ a_{10}b_{00} & a_{10}b_{01} & a_{11}b_{00} & a_{11}b_{01} \\ a_{10}b_{10} & a_{10}b_{11} & a_{11}b_{10} & a_{11}b_{11} \end{pmatrix} \end{matrix}. \tag{6.14}$$

The action of the tensor product, denoted by $\otimes$, is to combine the spaces in which these two operators act. In this way, a composite representation can be constructed where states of the two-qubit system can be written as single state vectors: $|\alpha\beta\rangle = |\alpha\rangle \otimes |\beta\rangle$.

Each degree of freedom can be addressed independently using the tensor product, for instance, if we replace matrix a in Eq. (6.14) with a 2×2 identity matrix, the resulting matrix,

$$\begin{array}{c} \\ \langle 00| \\ \langle 01| \\ \langle 10| \\ \langle 11| \end{array} \begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{pmatrix} b_{00} & b_{01} & 0 & 0 \\ b_{10} & b_{11} & 0 & 0 \\ 0 & 0 & b_{00} & b_{01} \\ 0 & 0 & b_{10} & b_{11} \end{pmatrix} \end{array}, \tag{6.15}$$

can only flip or apply phase differences to the second qubit. Similarly, if instead b were replaced by the 2×2 identity matrix the resulting matrix,

$$\begin{array}{c} \\ \langle 00| \\ \langle 01| \\ \langle 10| \\ \langle 11| \end{array} \begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{pmatrix} a_{00} & 0 & a_{01} & 0 \\ 0 & a_{00} & 0 & a_{01} \\ a_{10} & 0 & a_{11} & 0 \\ 0 & a_{10} & 0 & a_{11} \end{pmatrix} \end{array}, \tag{6.16}$$

can only flip or apply phase differences to the first qubit.

This generalises for tensor products of more degrees of freedom. The matrices quickly become unwieldy to write out explicitly, but computers can store and manipulate very large matrices. An operation a on the jth qubit of an n qubit system can be written $a_j = (\bigotimes_{k=1}^{j-1} I_2) \otimes a \otimes (\bigotimes_{k=j+1}^{n} I_2)$ where I_2 is a 2×2 identity matrix and $\bigotimes_{k=1}^{j-1}$ indicates repeated tensor products in the same way $\prod_{k=1}^{j-1}$ would represent repeated multiplication. In particular $\bigotimes_{k=n-q+1}^{n} I_2 = I_2 \otimes I_2 ... (\text{total of q times}) ... \otimes I_2 = I_2^{\otimes q}$.

Any Hermitian operator of size 2^n can be constructed from sums and products of Pauli operations on different sites $\sigma_j^{\{x,y,z\}}$ plus the identity operation. The operational meaning of $\sigma_j^{\{x,y,z\}}$ is to perform a Pauli $\{x, y, z\}$ operation on the jth qubit while doing nothing (the identity) to the others. Simultaneous operations on qubits j and k where $j \neq k$ can be represented as $\sigma_j^{\{x,y,z\}} \sigma_k^{\{x,y,z\}}$. Pauli operators acting on different qubits commute with each other, i.e., $[\sigma_j^{\{x,y,z\}}, \sigma_{k \neq j}^{\{x,y,z\}}] = 0$.

Coding tip: write a function (or three) to create $\sigma_j^{\{x,y,z\}}$ once, and call it in later functions, rather than try to 'hard code' the creation of each term from tensor products. Such functions can be written using less than 10 lines of code, if written well.

Appendix 2: Graphs and adjacency matrices

Graphs are formal mathematical objects consisting of vertices (usually represented visually by circles) connected by edges (usually represented by line segments). Graphs can be drawn on a two dimensional plane by choosing (x, y) coordinates for the vertices and drawing appropriate edges between them. This positioning of the vertices is important for human understanding but is not part of the definition of the graph itself. Rearranging the vertices keeps the graph the same. The power of graphs is that they allow

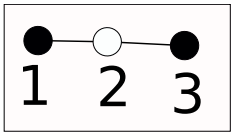

Figure 6.3: A simple graph with three vertices, coloured to show maximum independent set and numbered to match Eq. (6.18).

an abstract representation of the relationships between different entities. A family tree, for instance, is one type of graph which provides information about ancestry and lineage. In general graphs can have different types of edges (relationships between vertices) or weights assigned to each edge, and can either be directed (the relationship has a direction, e.g., parent to child in a family tree) or undirected. The maximum independent set problem is defined on a very simple class of graphs where edges are undirected and for which there is only one type of (unweighted) edge, any two vertices can only be unconnected to each other or connected by a single edge, and vertices are not allowed to connect to themselves (no 'self loops').

While graphs provide a powerful tool for humans to visualize real problems and systems, they are not a way of representing information which computers can easily operate on directly. Fortunately, since a graph is defined by the connections between vertices rather than the positioning of those vertices, a graph can efficiently be represented by a matrix, and matrices can be efficiently manipulated by computers. Consider the simple graphs which are used to define maximum independent sets. Each pair of vertices either share an edge or do not share an edge. The information contained in a graph of this kind with n vertices can be expressed as $n\,(n-1)$ binary variables (numbers which can only be 0 or 1). In practice however, it is more convenient to organize these variables into an $n \times n$ *adjacency matrix*, an array of numbers which has ones in the $(j, k > j)$ position if there is an edge between the vertices j and k and has zeros everywhere else. (Depending on the application, adjacency matrices may be defined with entries below the diagonal, too, for traversing edges from k to j.) The upper triangular definition is convenient for application to maximum independent set Hamiltonians, but we could have used either convention.

To construct an adjacency matrix, one must assign labels in some order to the vertices, as was done in Fig. 6.1(a). This ordering is arbitrary, but once chosen must be used consistently. The matrix in Eq. (6.10) is reproduced below, with the rows and columns labelled as a reference

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}. \tag{6.17}$$

As an example, if we look in the first row and second column, there is a 1 entry in this matrix, and indeed, there is an edge between vertex 1 and vertex 2 in Fig. 6.1. On the other hand, there is a zero in the first row and fourth column, and there is indeed no edge shared between vertex 1 and vertex 4.

Appendix 3: Examples of Hamiltonians

Given a graph, the vector h and matrix J which encode the problem in the Ising Hamiltonian can be defined by a repeated use of Eq. (6.9). Mathematically, this Hamiltonian can be expressed as a $2^n \times 2^n$ matrix for n vertices, although for large problems this matrix will not be practical to construct simply because of its size. Already for the maximum independent set problem defined in the main text, the matrix will be $2^5 \times 2^5 = 32 \times 32$, rather large to write out explicitly on the page.

Instead, consider a three qubit problem defined by the graph depicted in Fig. 6.3, which has the adjacency matrix

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{array}. \tag{6.18}$$

The Hamiltonian for the maximum independent set is therefore represented by the following arrays:

$$J = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{array}, \qquad h = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \\ \begin{pmatrix} -1+\kappa \\ -2+\kappa \\ -1+\kappa \end{pmatrix} \end{array}. \tag{6.19}$$

Straightforward calculations show that the *p*roblem Hamiltonian for this problem (the 8×8 Ising Hamiltonian) takes the following form,

$$H_{\text{problem}} = \begin{array}{cc} & \begin{array}{cccccccc} |0\rangle & |1\rangle & |2\rangle & |3\rangle & |4\rangle & |5\rangle & |6\rangle & |7\rangle \\ |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \end{array} \\ \begin{array}{cc} \langle 0| & \langle 000| \\ \langle 1| & \langle 001| \\ \langle 2| & \langle 010| \\ \langle 3| & \langle 011| \\ \langle 4| & \langle 100| \\ \langle 5| & \langle 101| \\ \langle 6| & \langle 110| \\ \langle 7| & \langle 111| \end{array} & \begin{pmatrix} -2+3\kappa & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2+\kappa & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2+\kappa & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2-\kappa & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2+\kappa & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2-\kappa & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2-\kappa & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6-3\kappa \end{pmatrix} \end{array}. \tag{6.20}$$

Note that this matrix is diagonal, because it has been constructed entirely from σ_z operators plus identity terms. The σ_x terms in the transverse field in Eq. (6.4) will appear as off-diagonal entries (see below). The bras and kets labelling the rows and columns depict two different ways of expressing the states, either by listing the states of each of the three qubits (blue), or by converting this state to a decimal number (green). For visual clarity, the lowest energy eigenvalue has been coloured orange (recall that $0 < \kappa < 1$). By examining the problem Hamiltonian, we can observe that the maximum independent set is the state $|101\rangle \equiv |5\rangle \equiv (0,0,0,0,1,0,0)^T$, where the last expression is the state expressed as a vector.

Finally, the adiabatic evolution Hamiltonian can be constructed. Setting $\kappa = 1/2$ and plugging into

Eq. (6.4). This yields

$$H(t) = -A(t)\sum_j \sigma_j^x + B(t)H_{\text{problem}} =$$

$$\begin{array}{c} \\ \langle 000| \\ \langle 001| \\ \langle 010| \\ \langle 011| \\ \langle 100| \\ \langle 101| \\ \langle 110| \\ \langle 111| \end{array} \begin{array}{c} \begin{array}{cccccccc} |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \end{array} \\ \begin{pmatrix} -\frac{1}{2}B(t) & -A(t) & -A(t) & 0 & -A(t) & 0 & 0 & 0 \\ -A(t) & -\frac{3}{2}B(t) & 0 & -A(t) & 0 & -A(t) & 0 & 0 \\ -A(t) & 0 & -\frac{3}{2}B(t) & -A(t) & 0 & 0 & -A(t) & 0 \\ 0 & -A(t) & -A(t) & \frac{3}{2}B(t) & 0 & 0 & 0 & -A(t) \\ -A(t) & 0 & 0 & 0 & -\frac{3}{2}B(t) & -A(t) & -A(t) & 0 \\ 0 & -A(t) & 0 & 0 & -A(t) & -\frac{5}{2}B(t) & 0 & -A(t) \\ 0 & 0 & -A(t) & 0 & -A(t) & 0 & \frac{3}{2}B(t) & -A(t) \\ 0 & 0 & 0 & -A(t) & 0 & -A(t) & -A(t) & \frac{9}{2}B(t) \end{pmatrix} \end{array}. \tag{6.21}$$

Bibliography

[1] S. Santra, G. Quiroz, G. Ver Steeg, and D. A. Lidar, *Max 2-SAT with up to 108 qubits*. New Journal of Physics **16**, 045006 (2014).

[2] V. Choi, *Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems*. arXiv:1010.1221 (2010).

[3] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser *Quantum computation by adiabatic evolution*. arXiv:quant-ph/0001106 (2000).

[4] E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm*. arXiv:1411.4028 (2014).

[5] S. Hadfield, *Quantum algorithms for scientific computing and approximate optimization* (PhD. thesis). arXiv:1805.03265 (2018).

[6] B. Tregenna, W. Flanagan, R. Maile, and V. Kendon, *Controlling discrete quantum walks: coins and initial states*. New Journal of Physics **5**, 83 (2003).

[7] V. Kendon, *Decoherence in quantum walks – a review*. Mathematical Structures in Computer Science **17**, 1169 (2007).

[8] A. Callison, N. Chancellor, F. Mintert, and V. Kendon, *Finding spin-glass ground states using quantum walks*. arXiv:1903.05003 (2019).

[9] R. Martonak, G. E. Santoro, and E. Tosatti *Quantum annealing by the path-integral Monte Carlo method: the two-dimensional random Ising model*. Physical Review B **66**, 094203 (2002).

[10] N. Chancellor *Modernizing quantum annealing using local searches*. New Journal of Physics **19**, 023024 (2017).