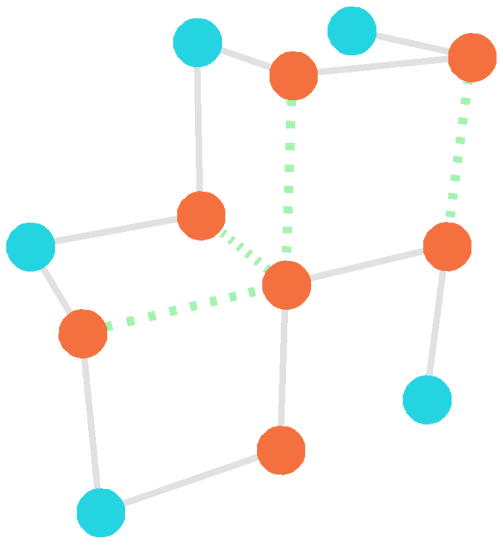

The University of Western Australia

# Lattice Protein Folding with Quantum Computing

*Ja-Jet Loh*
Department of Physics
School of Physics, Mathematics and Computing

Supervised by
Prof. Jingbo Wang
Department of Physics
School of Physics, Mathematics and Computing

28th October 2019



# Abstract

We present a detailed quantum circuit to solve lattice protein folding in the Hydrophobic-Hydrophilic model based on the Quantum Approximate Optimisation Algorithm (QAOA) with $O(pm^2)$ gates, $O(pm^2(\log m)^2)$ circuit depth and $3n(m+2)+3m+k+3$ qubits for a protein consisting of $m+1$ residues and for QAOA depth $p$. The circuit is specified down to the individual gate level, and achieves an exponential speedup in comparison to an exact classical search, which scales exponentially in time for longer residues. We simulate the quantum algorithm on a classical computer and obtain results for the performance of the algorithm for different values of $p$, in both 3D cubic and 3D BCC lattice models with hypercube and complete graph mixers. The success of these experiments indicate the potential for efficient lattice protein folding on near-term quantum computers.

# Acknowledgements

Many people have helped me on my journey to finishing this thesis. Thank you to all my friends, family and colleagues for supporting me in one way or another.

I'd like to say a special thanks and congratulations to Edric, Leigh and Matthew in my cohort, and to Sam and Mitchell for your advice and technical help before and during my Master's degree.

A special thanks my supervisor, Jingbo Wang, for taking me under her wing and for her immense support through my degree, without whom I would know nothing about protein folding nor quantum computing, nor have the skills I have developed throughout.

Finally, another special thanks to Jacquie, for your endless support and words of encouragement through thick and thin.

—

This work was supported by resources provided by the Pawsey Supercomputing Centre with funding from the Australian Government and the Government of Western Australia.

# Summary of Student Work

Upon deciding on Protein Folding as the topic of my research after consultation with my supervisor Jingbo Wang, I performed thorough exploration of the literature surrounding Protein Folding to improve my understanding of the field and to identify opportunities for exploration for my project. I explored the various models of Protein Folding and proposed a scheme which utilised the phase of qubits to store positional information.

After learning of the Quantum Approximate Optimisation Algorithm (QAOA) from my supervisor, I began to formulate my approach to using Quantum Computing to perform Protein Folding. Having identified a disadvantage of previous attempt at this from the papers [1, 2, 3], I decided to (1) use quantum gates in the Quantum Gate model of Quantum Computing to perform protein folding, as the Quantum Gate model lis the most promising of the current Quantum Computing technologies with known error correction and fault-tolerance protocols, and (2) to construct the circuit for protein folding as explicitly as possible.

After studying the 2D square lattice encoding used in [1], I independently developed the 'Classical Algorithm' for lattice Protein Folding described in Section 4.2 for a 2D square lattice. Upon studying the papers [2, 3] in more detail, I developed my encoding for the 3D cubic lattice which could take advantage of symmetries to reduce computation time, then also developed my encoding for the 3D BCC lattice.

I then completed the design of Circuit 7 for the calculation of conformation energies, and its sub-circuits and proceeded to develop the details of the simulations to be run.

All simulations and computations detailed in this thesis were programmed and performed solely by me, with the exception of the use of software packages listed at the beginning of Section 5. A selection of my work includes:

- Programming a vector-matrix exponential multiplication based on the scaling and squaring method, utilising Multiprocessing [4, 5] to improve performance
- Programming a Nelder-Mead optimisation function
- Programming and running MPI-parallel and multi-threaded QAOA simulation functions capable of running efficiently on Magnus in the Pawsey Supercomputing Centre, integrated with the matrix exponential and Nelder-Mead optimisation functions
- Verifying the results of my simulation functions using Mathematica [6]
- Adapting the QAOA simulation functions for, and performing the stochastic minima search in Section 5.3.

With advice from my supervisor and Samuel Marsh, I then developed Circuit 6 based on a circuit in [7], and Circuit 15 based on a circuit in [8], and thus completed the overall circuit design.

# Contents


**1 Introduction** **8**

**2 The Protein Folding Problem** **9**

2.1 Background information for the Bioscience Layman . . . . . 10

2.2 What is the Protein Folding Problem? . . . . . 12

2.3 Protein Folding models . . . . . 13

2.3.1 Lattice models . . . . . 13

2.3.2 Potential-based models . . . . . 14

2.3.3 Statistical models . . . . . 15

**3 Quantum Computing** **16**

3.1 Classical Computing and its Limitations . . . . . 16

3.2 The Potential of Quantum Computing . . . . . 16

3.3 Quantum Gate Model . . . . . 18

3.4 The Quantum Approximate Optimisation Algorithm (QAOA) . . . . . 21

3.5 Previous Quantum Computational attempts at Protein Folding . . . . . 22

3.6 Our Proposed Approach . . . . . 24

**4 Mathematical Framework for Classical and Quantum Protein Folding** **26**

4.1 Problem Definition . . . . . 26

4.2 Classical Algorithm . . . . . 26

4.2.1 Conformation encoding . . . . . 26

4.2.2 Conformation energies . . . . . 30

4.2.3 Scaling of the computation . . . . . 33

4.3 Quantum Algorithm . . . . . 34

4.3.1 Circuit complexity calculation . . . . . 37

4.3.2 State initialisation and the encoding of conformations . . . . . 38

4.3.3 Unitary phase operator $U_{\mathrm{C}}(\gamma_i)$ . . . . . 40

4.3.4 Unitary mixing operator $U_{\mathrm{B}}(\beta_i)$ . . . . . 50

**5 Simulating the Quantum Algorithm** **52**

5.1 Overview of the Classical Simulation . . . . . 52

5.1.1 Obtaining the energy vector . . . . . 53

5.1.2 Performing the QAOA . . . . . 54

5.1.3 Optimisation and post-processing . . . . . 55

5.2 Parameter Space Results and Visualisation . . . . . 55

5.3 QAOA Depth . . . . . 60

**6 Conclusion and Future Work** **67**

**References** **68**

**Appendix 1: Quantum Gate Reference Guide** **76**

**Appendix 2: Project Proposal and Deviations** **80**

# 1 Introduction

It is an exciting time to be a part of Quantum Computing. The race to achieve quantum supremacy is hotly contested and discussed, and more research than ever is being done on quantum computation and information. In particular, it is thrilling to work on a problem such as protein folding, which has been a long-standing problem in biochemistry with important medical applications. With this, we can imagine what the world may be like if researchers had all the computational power they need to understand the causes of protein misfolding, to design drugs for medicines, or to predict patient outcomes from large amounts of data. We hope that this work will help pave the way for these opportunities in the future.

Section 2 begins by introducing the Protein Folding Problem and explaining its relevance. We then look at a number of models that are used for Protein Folding.

Section 3 begins with an overview of Quantum Computing and our motivations for it, before explaining the Quantum Gate model. The Quantum Approximate Optimisation Algorithm (QAOA) is also discussed, as are some particular approaches to Protein Folding using Quantum Computing by other researchers previously.

Section 4 provides a step-by-step guide of our proposed algorithm for Lattice Protein Folding on a Quantum Computer, by first explaining a corresponding classical algorithm before translating the algorithm into a quantum circuit. The QAOA is the heuristic approach used here, with the proposed quantum circuits forming the main part of the QAOA.

Finally, Section 5 describes the process used to simulate the quantum algorithm to verify the method used, and presents the results of simulations using the supercomputer Magnus from the Pawsey Supercoming Centre. Some opportunities for extending the work presented in this thesis are provided at the end of this section.

Appendix 1 contains a detailed list of quantum gates and sub-circuits used in this thesis, which is included for reference.

Appendix 2 contains the proposal presented at the beginning of this project, and briefly explains the main changes between it and the research performed and presented in this thesis.

# 2 The Protein Folding Problem

A single adult human has approximately $3 \times 10^{13}$ cells, each of which can consist of around $4 \times 10^7$ protein macromolecules - totalling around $10^{21}$ proteins in a person[1]. Each protein has a biological role to play, based almost entirely on its native state - that is, the natural folded structure of the protein.

However, proteins (such as in Figure 1) themselves are extremely complex - they are typically chains of hundreds of individual residues, with 20 possible amino acids for each residue. Attempting to predict the native structure of a protein from the sequence of amino acids is thus a hugely difficult task. Some estimates of the time required for a protein to find its native state via a naive, random search would range up to $10^{27}$ years, though in reality proteins fold in seconds [11], or microseconds at their fastest [12].

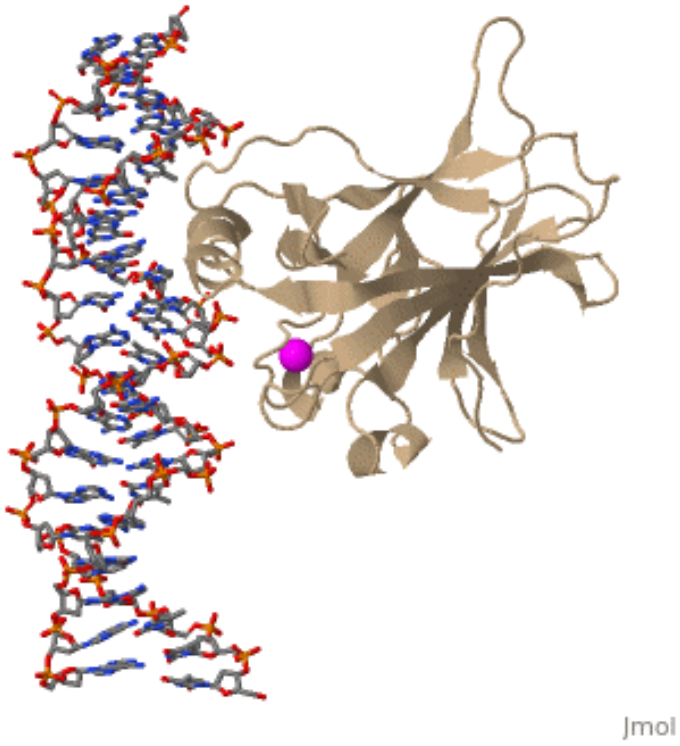



Figure 1: A ribbon diagram of the 'p53' protein (right) complexed with a $Zn^{2+}$ ion (centre) and a section of DNA (left). The thin, brown lines are the various residues in sequence and their positions in space, and the arrows represent $\beta$-sheets, a common structure in proteins [13].

With such an enormous space of conformations[2], and so many proteins in a single person, protein misfolding is inevitable - that is, proteins folding into a structure other than their native state. As protein functions are often critical, some diseases characterised by misfolding include:

[1] Using estimates of $3 \times 10^{13}$ cells in a human body and $4 \times 10^7$ proteins per cell [9, 10]

[2] Conformations are unique spatial configurations of amino acid residues

- Alzheimer's disease [14, 15, 16, 17]: a neurodegenerative disease associated with deterioration of parts of the brain which control thought, memory and language [18, 19, 20]. The disease occurs in 8% of Australians over the age of 65[3].
- Parkinson's disease [15, 16, 17]: a neurodegenerative disease which predominantly affects the part of the brain which produces dopamine, and is associated with muscle tremors, stiffness and difficulty with movement, balance and coordination [24, 25]. The disease occurs in 1.7% of Australians over the age of 65 and 3.0% of those over 85 [26].
- Huntington's disease [15, 16]: a hereditary disease caused by a defective gene which interferes with the production of a protein (huntingtin) required for proper brain development [27, 28].
- Creutzfeldt-Jakob disease [16]: a rare, neurodegenerative disease caused by prions (a type of protein which causes other specific proteins in the brain to fold abnormally [29, 30]). The most common form of the disease (sporadic CJD) occurs in 85% of cases and 90% of such cases do not live a year past initial diagnosis [31].
- Many types of cancer [32, 33, 34, 17], with more than 50% of human cancers being linked with loss of function of the 'p53 tumour suppressor' protein [33, 34].
- Type 2 diabetes [14, 17, 35], where the misfolding of the islet amyloid polypeptide has been linked with the disease [35].

A detailed understanding of the process of protein folding would not only help to understand the aetiology of the above diseases, but could also be used to improve drug design to assist in treatment and recovery from not only the aforementioned diseases, but also other diseases not caused by protein misfolding [36, 37]. This is a particularly useful application of protein folding with a lot of recent activity, such as with Google DeepMind's AlphaFold [36], and even with startups such as ProteinQure which use quantum computing techniques to predict folded structures [2, 3]. This is explored more in Section 3.5.

## 2.1 Background information for the Bioscience Layman

Proteins play many vital roles in the human body, from comprising muscles which allow for our movement, to breaking down food that we eat as enzymes, to transporting smaller atoms and molecules throughout our bodies, to almost every other function that is vital to life [38, 39]. However, before we go into what it means for a protein to 'fold', we must first have an understanding of what a protein is.

[3]Estimated with 447000 Australians with dementia take 27000 with early onset dementia (under 65) in 2019 [21], and with 70% of dementia cases from Alzheimer's disease [22], we can estimate 294000 cases of Alzheimer's in over 65s. With $3.67 \times 10^6$ persons over 65 in 2016 [23], we can estimate prevalence of approximately 8% in over 65s in 2019.

Proteins themselves are very large and complex molecules (an example is shown in Figure 1). They are chains of proteinogenic[4] amino acids residues[5], which themselves are comprised of between 10 and 39 atoms of C, H O and N each (an example is shown in Figure 2) [40].

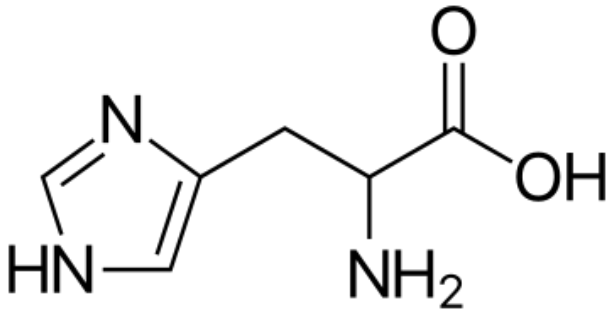



Figure 2: Molecular structure of the amino acid Histidine [41]

The example given in Figure 1 is the native state of the 'p53 tumour suppressor' protein, named as it is important in the body for the prevention of cancerous tumours [42, 43]. One can imagine untangling the protein as if untangling a length of string, until it forms a completely straight chain. At each point in this untangling process, there is an energy associated with the conformation of the protein, based on attractive and repulsive forces between all the atoms within the protein. The protein has high energy when it is 'untangled' and unstable; it has the lowest energy and is stable in its native state. Thus, an untangled protein will spontaneously fold itself back into its native structure. This spontaneous change from an 'untangled' conformation to its native state is the phenomenon of protein folding.

However, due to the enormous space of conformations for even a single protein, there are many ways in which proteins can misfold, illustrated by Figure 3.

[4]Proteinogenic amino acids are special amino acids which occur naturally in biological systems, which are in turn used to form proteins.

[5]These amino acids are referred to as amino acid residues when bonded in a chain, like in a protein.

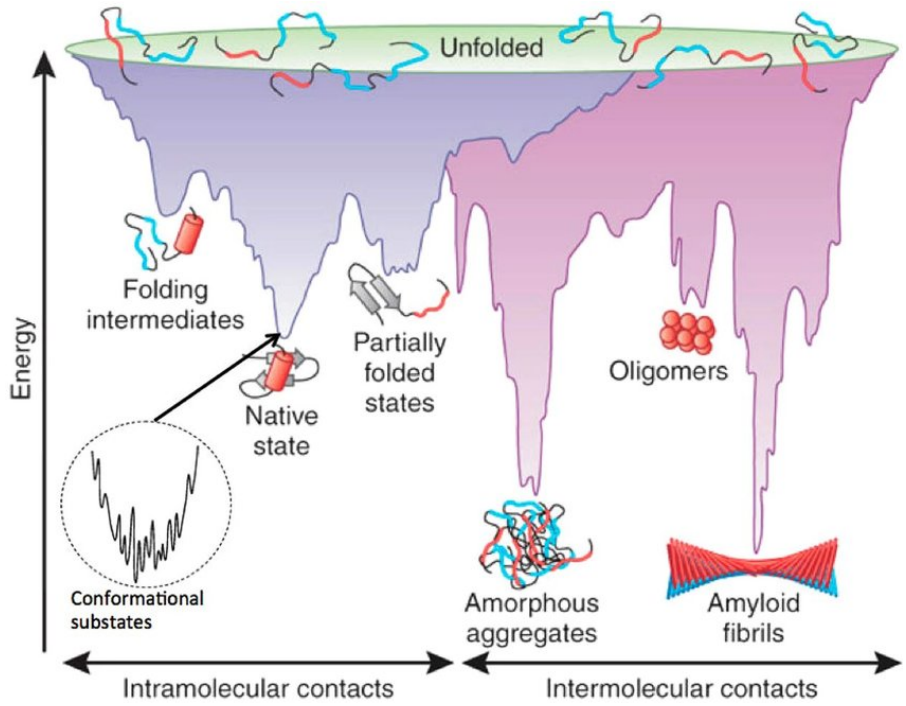



Figure 3: Illustration of a protein's energy landscape from [44]. A single native state exists where the protein is functional, but there are many local minima, and when considering interactions with other proteins, aggregates and other structures can form as well.

## 2.2 What is the Protein Folding Problem?

The Protein Folding Problem can be considered in three parts [45, 46, 47].

1. The folding code - the theoretical problem of what the folded structure of a protein is, based on its sequence, and what causes the state to be the native state.
2. Levinthal's paradox [48]- what are the physical mechanisms which allow proteins to fold into their native configurations so quickly and consistently?
3. Computation - the practical problem of designing and executing an algorithm which can predict the native/folded state of a protein based only on its sequence.

All three problems are considered well understood currently nowadays [45], with folding due to hydrogen bonding causing $\alpha$-helix and $\beta$-sheet structures, Van de Waals interactions, bond angle preferences and electrostatic interactions. Levinthal's paradox is resolved, as protein energy landscapes act as funnels producing main 'pathways' through the conformation space proteins fold on. Figure 3 alludes to this, and this funnelling is aided by the random motion of other molecules such as water continually 'jostling' proteins down the funnel.

However despite the wealth of knowledge on the processes behind protein folding, it is still very difficult to simulate the folding of proteins computationally. In 2008-2009, research groups

simulated *ab initio*[6] protein folding at the atomic level for durations of around 1 millisecond [49, 50], and in 2014 some groups were able to simulate milliseconds of folding time within just a day [51]. More recent work has seen the use of machine learning methods including deep learning [52, 53] and genetic algorithms [54]. These methods are based on models of protein folding, which we will now explore.

## 2.3 Protein Folding models

With proteins being such inherently complex structures, a wide variety of models exist with varying degrees of abstraction. A few key types of protein models are explained here.

### 2.3.1 Lattice models

The simplest models for Protein Folding are lattice-based models such as the Hydrophobic-Hydrophilic (HP) model [55]. In the HP model, proteins are modelled as self-avoiding walks on a 2D or 3D lattice, such as in Figures 4 and 5.

In the HP model, amino residues are classified as Hydrophoblic (H) or Hydrophilic/Polar (P), and the energy of a conformation is determined by counting the number of adjacent H-H pairs of residues. Adjacency here refers to lattice points which are directly connected to each other.

This is clearly a very coarse approximation of a real protein, but despite this, research utilising the model is still active, with groups now resorting to machine learning methods [56, 57, 58] and even quantum computing techniques [2, 3]. In addition to this coarse approximation, finding the native state in this model does not predict the trajectory through conformation space taken to achieve the native state.

Other lattice models exist, such as the Miyazawa-Jernigan (MJ) model [59]. In the MJ model, instead of just using two classifications, all 20 amino acid classifications are used, and interaction energies between all pairs of amino acid types are empirically provided. This accounts for interactions between all different types of adjacent amino acid pairs, as opposed to the HP model where interactions only occur between adjacent residues.

[6] *ab initio* means 'from the beginning', referring to methods based on basic mathematical rules, such as the CHARMM potential (Section 2.3.2). This is as opposed to methods such as pure homology-based modelling (Section 2.3.3).

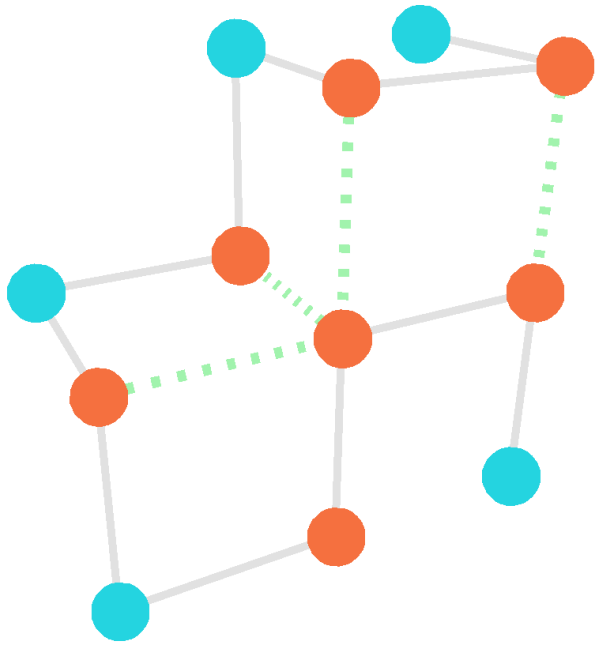


Figure 4: A chain of residues in the 3D cubic HP model. Orange and blue points denote H (hydrophobic) and P (polar) residues respectively. Grey lines denote bonded residues, and green dotted lines denote H-H interactions. This and other lattice diagrams were produced using Plotly [60] for Python.

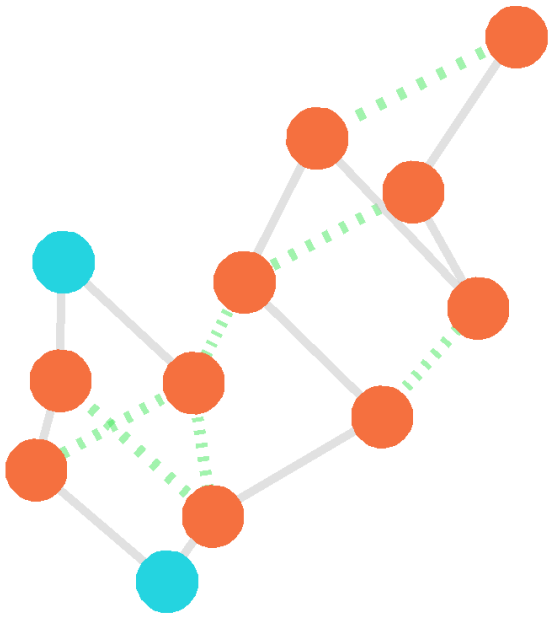


Figure 5: A chain of residues in the 3D BCC HP model. Each lattice point has 8 adjacent points (as opposed to 6 in the cubic case), corresponding to relative positions of $(\pm 1, \pm 1, \pm 1)$.

### 2.3.2 Potential-based models

More sophisticated models predict the energy of protein conformations as functions of the precise locations of individual atoms relative to each other, accounting explicitly for interatomic forces. One such potential is the CHARMM potential [61], which is of the form

$$\begin{aligned} U(b,\theta,\tau,\omega,r_{i,j}) = &\sum_{\text{bonds}} c_l(b-b_0)^2 \quad + \sum_{\text{bond angles}} c_\alpha(\theta-\theta_0)^2 \quad + \sum_{\text{improper torsion angles}} c_i(\tau-\tau_0)^2 \\ &+ \sum_{\text{dihedral angles}} \text{trig}(\omega) \quad + \sum_{\text{charged pairs}} \frac{Q_iQ_j}{Dr_{ij}} \quad + \sum_{\text{unbonded pairs}} c_w\varphi\left(\frac{R_i+R_j}{r_{ij}}\right) \end{aligned} \tag{1}$$

and can be considered as the summation of covalent bonding (through bond, bond angle, improper torsion angle and dihedral angle terms), electrostatic forces and Van de Waals interactions. A more detailed explanation of each term and variable can be found in [62]. Other potential-based models of similar forms exist, such as AMBER [63] and AAWSEM [64].

The force of each particle is given by partial derivatives of the potential

$$\boldsymbol{F}_i(x) = -\nabla_i U \tag{2}$$

and by Newton's Second Law, these forces are related to the particle's acceleration. This results in a large set of coupled differential equations which can be solved numerically to determine the time-evolution of the particles comprising the system.

There are many schemes which can be used to perform this numerical integration, such as the Euler method, or the Verlet method. The Verlet method is one of the integration schemes included with CHARMM and propagates a particle's position based on its previous locations and the force acting on the particle as follows: [65]

$$x_i(t + \Delta t) = 2x_i(t) - x_i(t - \Delta t) + \frac{F_i(t)}{m_i}(\Delta t)^2 \tag{3}$$

Potential-based models offer arbitrarily small precision, but are also very slow due to:

- The sheer number of parameters required, and the associated difficulty in finding a global minimum in the parameter space, and
- Having to be evolved in time from some initial state to obtain their final state. This is advantageous in that it is possible to observe the dynamics of the folding, but again this evolution takes an intractable amount of computing power.

### 2.3.3 Statistical models

With such difficulty in performing protein folding from physical principles, it is very common to use proteins with known structures and sequences to assist in the structure prediction of slightly different proteins. The two main categories of these methods are homology modelling and protein threading/fold recognition. These methods are possible due to the large number of proteins with known structures through databases such as the Protein Data Bank [66] and other databases.

Currently the best models for protein folding use a combination of *ab initio*, homology and fold recognition techniques [67, 68, 69], as measured by the biennial Critical Assessment of Techniques for Protein Structure Prediction (CASP).

Current protein folding models are increasingly adopting machine learning methods, with a recent prominent example being DeepMind's AlphaFold [36], which caused a sudden upset in the 2018 CASP13 by beating existing teams by a large margin [70, 71].

# 3 Quantum Computing

## 3.1 Classical Computing and its Limitations

'Calculating machines' have existed for centuries with the purpose of performing arithmetic and other calculations [72], with early computers such as Leibniz's Step Reckoner [73] being entirely mechanical. Later computers such as Colossus [74] were electronic and utilised vacuum tubes to perform calculations, before the advent of transistors leading to the electronic computers we are familiar with today.

Over time, the combination of advancing technologies and more efficient production meant better processors could be manufactured faster, leading to Moore's famous law [75] predicting that the number of transistors per area in integrated circuits to double every two years. This prediction has proved accurate up until now, though many see the end of this trend soon as the size of transistors on today's best integrated circuits approaches the scale of significant quantum effects.

As computing power grows, there is greater opportunity to solve more difficult problems. We can measure how 'difficult' problems are by looking at the computational complexity of algorithms to solve them - that is, we can see how algorithms perform as the size of the problem increases. We will illustrate this with the case of the Discrete Fourier Transform (also known as the Fast Fourier Transform, or FFT).

One of the best known algorithms for computing the FFT is the Cooley-Tukey algorithm, which requires $O(n2^n)$ operations for a vector of $2^n$ complex numbers.

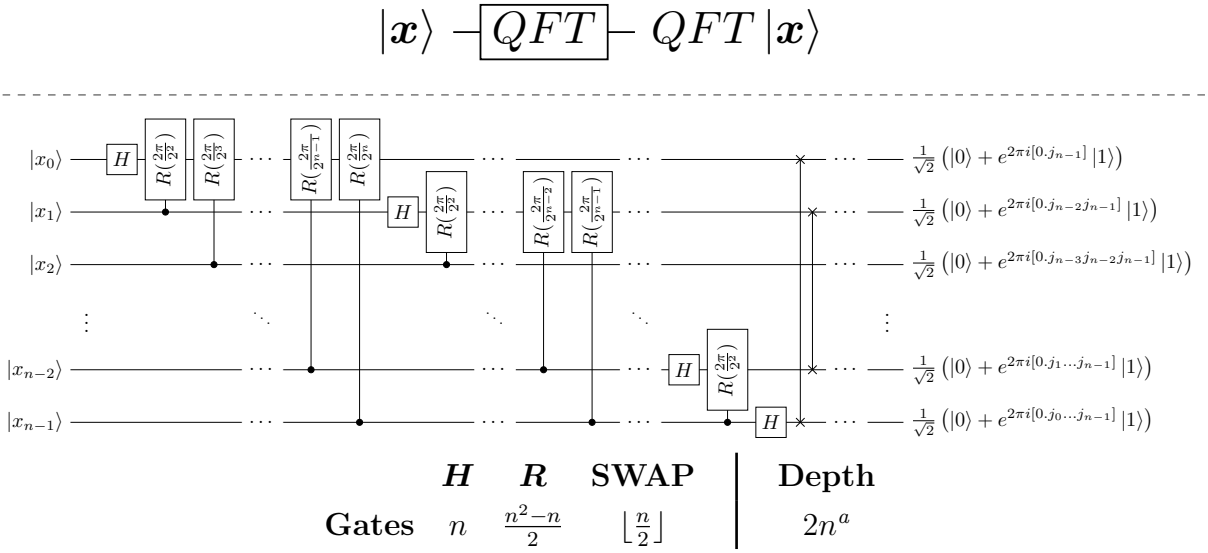
We can contrast this with the circuit for the Quantum Fourier Transform (Circuit 1), which uses only $O(n^2)$ gates/operations [76, 77]. This is an exponential improvement in the number of operations required to compute the FFT, meaning that algorithms requiring intensive FFT calculation can potentially see enormous increases in performance if performed using quantum computing.

## 3.2 The Potential of Quantum Computing

The FFT is one example of quantum computing improving on current computational performance, and although quantum computing has only gained a lot of traction in recent years, the idea of quantum computing has existed for decades.

Richard Feynman is widely attributed as the first person to formally propose the use of quantum mechanics to perform calculations, particularly simulations of quantum mechanics [78] in the early 1980s. In the years following Feynman's proposal, the groundwork for quantum computing was built, eventually becoming the field of research it is today.

Circuit 1: Quantum Fourier Transform




[a]Sets of $H$ and $R$ gates can be performed in parallel resulting in depth $2n-1$. All SWAP gates except the outermost can be performed in parallel with the final $H$ and $R$ gates.

Circuit 1: Circuit for the Quantum Fourier Transform. This and all other quantum circuit diagrams were produced using the $\langle q|pic\rangle$ package [79].

The two aspects of quantum mechanics which give quantum computing its power are superposition and entanglement. With entanglement, it is possible to process more information in one step with 2 qubits than possible with two unentangled qubits[7]. With superposition, we perform operations on multiple states at a single time - theoretically as many states as desired.

One challenge with quantum computing is having to adapt to the use of quantum operators instead of using the logic that wenare accustomed to with classical computing. One manifestation of this is that all quantum operations are unitary, thus no information is lost through quantum operations, and all operations (except measurement) are reversible. This may seem problematic at first, as even basic gates such as the AND gate[8] violate this reversibility. However, a simple workaround to this is to perform operations 'controlled' on input qubits, and have an additional qubit (an ancillary qubit, or ancilla) which stores the desired result. With workarounds such as these, it is possible to perform many, and indeed all classical computations with at least the

[7]To illustrate this, we can write the state of two unentangled qubits as $e^{i\gamma}(\cos\frac{\theta_0}{2}\,|0\rangle + e^{i\phi_0}\sin\frac{\theta_0}{2}\,|1\rangle)(\cos\frac{\theta_1}{2}\,|0\rangle + e^{i\phi_1}\sin\frac{\theta_1}{2}\,|1\rangle)$. Neglecting $\gamma$ as a global phase factor cannot be detected, our system has 4 degrees of freedom. We can write the state of two qubits more generally as $\alpha_0\,|00\rangle + \alpha_1\,|01\rangle + \alpha_2\,|02\rangle + \alpha_3\,|03\rangle$, where each $\alpha_i \in \mathbb{C}$. One degree of freedom is lost to global phase and another from the constraint $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$, thus now we have 6 degrees of freedom - two more than compared to when unentangled.

[8]The classical AND gate takes two inputs, and returns an output of 1 if both inputs are also 1. The gate returns 0 otherwise. This is clearly not reversible as a single 0 output cannot uniquely determine the inputs that created it.

same efficiency using quantum computing [80].

## 3.3 Quantum Gate Model

In this thesis we will utilise the Quantum Gate Model of Quantum Computing. In this model, quantum states are formed by qubits and represented by kets. Individual or multiple qubits can be acted on by different quantum gates, which perform unitary transformations on states. This is analogous to the use of logic gates in low-level electronics to perform operations on bits.

In classical computing, the most basic operations are performed by basic logic gates such as AND, OR, XOR and NOT, which form a universal set of logic gates, meaning these gates can be combined in ways to form any desired computational operation. These gates act on bits, which have a value either 0 or 1 at any given time.

In quantum computing, the most basic operations are performed by the Hadamard (H), Pauli gates (X, Y, Z) and Controlled-Not (CNOT) gates, which form a universal set. These quantum operators act on quantum bits, called qubits. The physical implementation of the qubits can vary, and we briefly discuss some of these implementations at the end of this subsection (3.3). Below is a table of some common quantum gates (Table 1) and an example circuit (Circuit 2) with calculations of its result on an initial state. An explanation of the circuit follows:

- Each horizontal line in Circuit 2 represents a qubit (or in other cases, multiple qubits). We read each line from left to right, with each symbol along the line representing a quantum gate.
- The kets to the left of each horizontal line ( $\begin{smallmatrix}|0\rangle\\|0\rangle\end{smallmatrix}$ ) denote the initial state of each qubit.
- Similarly, kets to the right of each horizontal line denote the final state of the qubits. As the qubits can be entangled at the end of this circuit, we need to write the state of the resulting entangled qubits together, denoted by the brace }.
- The gate elements perform unitary operations on the qubits. A selection of common quantum gates can be found in Table 1, and a more detailed list can be found in Appendix 1: Quantum Gate Reference Guide.

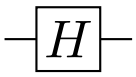
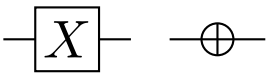
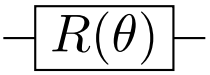
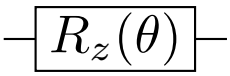
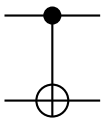
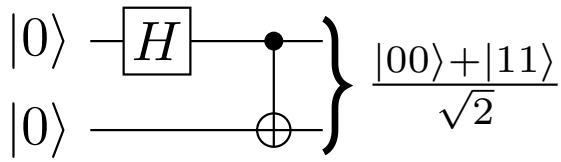
| **Gate name(s)**<br>**Description** | **Symbol(s)** | **Matrix** |
|---|---|---|
| Hadamard (H)<br>Maps $\|0\rangle$ to $\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$<br>and $\|1\rangle$ to $\frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$ |  | $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$ |
| Pauli-X ($\sigma_x$) / NOT (X)<br>'Flips' the qubit, like a<br>traditional NOT gate |  | $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$ |
| Phase |  | $\begin{pmatrix}1 & 0\\ 0 & e^{i\theta}\end{pmatrix}$ |
| Z Rotation |  | $e^{-i\sigma_z\theta/2} = \begin{pmatrix}e^{-\frac{i\theta}{2}} & 0\\ 0 & e^{\frac{i\theta}{2}}\end{pmatrix}$ |
| Controlled NOT (CNOT)<br>Applies 'NOT' to the<br>second/target qubit if the<br>first/control qubit is $\|1\rangle$ |  | $\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$ |

Table 1: A selection of common gates in the Quantum Gate model of Quantum Computing. A full list of gates referenced in this thesis can be found in Appendix 1: Quantum Gate Reference Guide.

Circuit 2: Generating a Bell state




Calculation 1

$$|00\rangle \xrightarrow{H} \frac{|00\rangle + |10\rangle}{\sqrt{2}} \xrightarrow{CNOT} \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Calculation 2

$$\begin{aligned}
|00\rangle &= CNOT\,(H \otimes I) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}
\end{aligned}$$

Circuit 2: Circuit for generating the Bell state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Calculation 1 shows the effect of gates on the qubit level, and Calculation 2 shows the effect of the gates as matrix multiplication. Note that the tensor product $H \otimes I$ can be represented as the Kronecker product of the two matrices $H \otimes I = \frac{1}{\sqrt{2}} \begin{pmatrix} I & I \\ I & -I \end{pmatrix}$.

When denoting a ket as a vector, it is conventional to use the 'computational basis'. For example in Circuit 2, where 2 qubits were used, the vector representation of our quantum state has 4 components, which correspond to $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$ respectively. We will presume the use of this convention though the rest of this thesis unless explicitly mentioned otherwise.

We can also encode larger integers in the computational basis by taking their binary expansion. For example, to represent $11 = 1011_2$, we use $|1011\rangle$. Reading this state on a circuit, by convention the top-most qubit represents the most significant bit in our integer. We will assume use of this convention throughout this thesis.

In order to have a working quantum computer, there must be physical qubits to perform computations, akin to the use of electrical signals as bits in classical computing. The spin of an atomic nucleus, or of an electron are both examples of quantities which can function as a qubit. The first manipulation of qubits in order to implement a 'quantum gate' was in 1995, where a CNOT gate was implemented with two atomic hyperfine states and two harmonic oscillator states [81]. Other particles have been proposed as qubits since and are more commonly used now, such as Cooper pairs in superconductors [82, 83], or photons [84].

The Quantum Gate model is not the only scheme for performing Quantum Computing. Other schemes include quantum annealing, quantum walks, and measurement-based cluster states are also used, but we will largely neglect these except for quantum annealing, as this is used in later discussion.

In quantum annealing, optimisation problems are framed in terms of a Hamiltonian, whose ground state corresponds to the desired solution. Qubits are initialised in the ground state of a simpler initial Hamiltonian with a known ground state, and the qubits evolve via the Schrodinger equation. A transverse magnetic field can be used to slowly (adiabatically) change the Hamiltonian such that the qubits remain in the ground state of the instantaneous Hamiltonian. Once the Hamiltonian has been evolved to the desired Hamiltonian, the qubits are in the ground state of the desired Hamiltonian, and can be measured. This can be difficult to physically implement, as it is difficult to maintain qubit coherence, and quantum annealing performs best with adiabatic changes, which are 'slow' by definition.

Currently there are many small quantum computers being developed in industry such as by IBM [85], Google [86] and D-Wave Systems [87], using different qubit implementations and different schemes of Quantum Computing. Development of practical quantum computers is a rapidly growing field, with claims of Quantum Supremacy[9] by Google [88] and argued refutation by IBM [89] just days before the completion of this thesis.

Whichever implementation is used, especially in the near-term, a significant issue is noise and decoherence in the quantum system. As quantum effects are only observed on the atomic scale, and effects such as entanglement are difficult to maintain even for more than a millisecond, the first quantum computers will be very noisy and will only be capable of a small number computations before qubits decohere beyond practical use [90]. Error correction and fault tolerance are schemes which aim to counter these effects, but this requires many thousands of ancillary qubits [91].

## 3.4 The Quantum Approximate Optimisation Algorithm (QAOA)

Whilst it is difficult to perform adiabatic changes to field strength and maintain coherence as is required for quantum annealing, it is possible to approximate such a process.

The QAOA (Quantum Approximate Optimisation Algorithm) is a hybrid quantum-classical algorithm proposed by Farhi et. al. [92] and then generalised by Hadfield et. al. [93] and Marsh & Wang [94, 8]. The algorithm minimises some objective function of bits through amplifying the probability of measuring the optimal state/s via the application of parameterised phase and mixing operators. Optimal values for these parameters can be determined, and this increases

[9]Though varying definitions and interpretations of 'Quantum Supremacy' exist, it can be considered as the proof that quantum computers are able to perform computations which are practically impossible for classical computers.

the probability of measuring the optimal state/s.

For completeness, we detail the steps from [93] here. Consider an objective function $C(z)$ to be minimised with respect to possible strings of bits (bitstrings e.g. 1001011) $z$.

1. Prepare an initial state $|s\rangle$
2. For some parameters $\beta = (\beta_0, \beta_1, \cdots, \beta_{p-1})$ and $\gamma = (\gamma_0, \gamma_1, \cdots, \gamma_{p-1})$ where $\beta_i \in [0, \pi)$ and $\gamma_i \in [0, 2\pi)$, and some choice of cost Hamiltonian $C$ and mixing Hamiltonian $B$, produce the state

$$|\beta, \gamma\rangle = e^{-iB\beta_{p-1}} e^{-iC\gamma_{p-1}} \cdots e^{-iB\beta_1} e^{-iC\gamma_1} e^{-iB\beta_0} e^{-iC\gamma_0} |s\rangle \tag{4}$$

   $C$ is the objective function i.e. $C|z\rangle = C(z)|z\rangle$, in which case both $C$ and $e^{-i\gamma_j C}$ are both diagonal matrices.

   $B$ is a mixing matrix can be chosen specifically to the problem at hand, as long as

   - The space of bitstrings $F$ corresponding to feasible solutions is mapped to itself (as depending on how a problem is encoded into bitstrings, not all bitstrings may correspond to physically real situations), and
   - $B$ allows for transitions between any two elements in $F$. That is, $\forall\, x, y \in F$, $\exists\, \beta^* \in [0, \pi)$ and $r \in \mathbb{Z}$ such that $\left|\langle x \mid e^{-i\beta^* B} \mid y\rangle\right| > 0$

3. Apply the cost Hamiltonian to the $|\beta, \gamma\rangle$ state and measure such states repeatedly as to obtain its expectation value

$$\langle C\rangle = \langle \beta, \gamma \mid C \mid \beta, \gamma\rangle \tag{5}$$

4. Repeat steps 2 and 3, minimising $\langle C\rangle$ with respect to the parameters $\beta$ and $\gamma$.
5. Once optimal values for $\beta$ and $\gamma$ have been determined, prepare $|\beta, \gamma\rangle$ with these parameters and measure repeatedly until a satisfactory result is obtained.

## 3.5 Previous Quantum Computational attempts at Protein Folding

With the high degree of complexity in protein folding and the potential computational power of Quantum Computing, a natural question is whether Quantum Computing can be used to perform Protein Folding. Indeed, there have been prior attempts at this, and we will briefly summarise some key attempts.

In 2012, Perdomo-Ortiz et. al. [1] used quantum annealing to demonstrate the folding of a 6-residue chain in the 2D MJ model, performed on a quantum annealer. Using the quantum annealer, they obtained the native state of the chain in 0.13% of trials. In this paper, conformations were encoded in a bitstring (e.g. 01 01 10), where pairs of bits described bond directions between residues.

In 2018, Fingerhuth, Babej & Ing [3] proposed the use of the Quantum Alternating Operator Ansatz (QAOA) [93] as a method for protein folding, where the 'one-hot' encoding shown in Figure 6 is used to encode protein conformations. [3] also provides a number of mixing operators, the most effective of which yielded a ground state probability of 0.477 for a four-residue chain (PSVK) in the 2D MJ model when simulated without accounting for error.

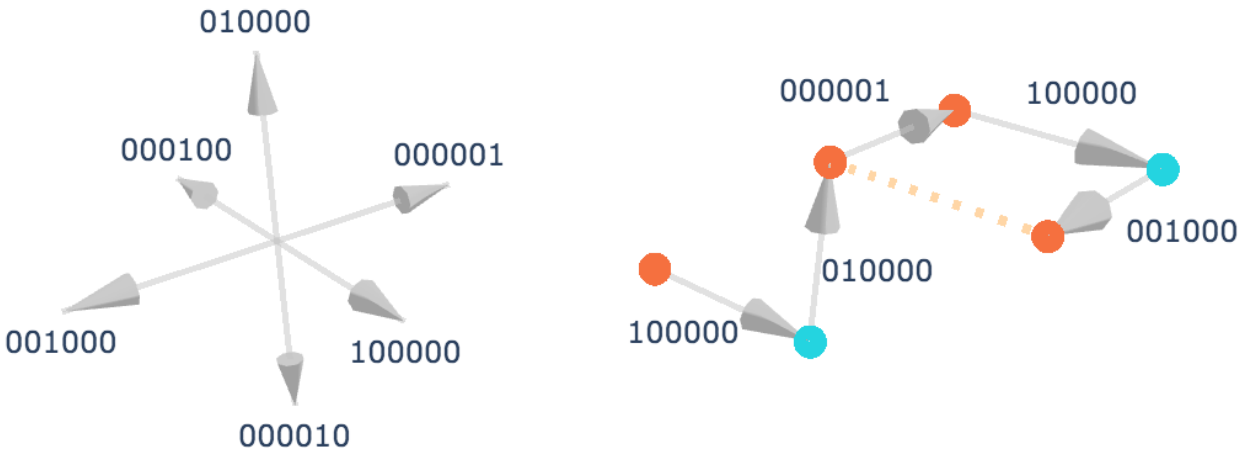



Figure 6: One-hot encoding in a 3D cubic lattice from [3] with an example chain

The 'one-hot' encoding uses 6 qubits to encode a single bond in the 3D case, with each qubit corresponding to a possible bond direction or 4 qubits in the 2D case. We will describe the 3D case from here on, as the 2D case is very similar. Only sextuples with a Hamming weight of 1 (that is in this case, only those with exactly one '1' qubit) correspond to physically real bonds, and the computation is set up as to only work in the space of such sextuples. This is a relatively sparse encoding of the conformation, but lends itself to construction of custom mixing operators.

The cost Hamiltonian $H_c$ is a function of the protein conformation, which is uniquely specified by the one-hot encoding and is defined as

$$H_C = H_{overlap} + H_{pair}$$

where $H_{overlap}$ is an energy penalty which has some positive value when the conformation described by a bitstring self-intersects (i.e. more than one residue occupy the same coordinate), and $H_{pair}$ describes the energy of nearest-neighbour pairs of residues which are not bonded to each other. Specifically these are given by

$$H_{overlap}(i,j) = \prod_{k=1}^{D} \prod_{r=1}^{\lceil \log_2(j-1) \rceil} \text{XNOR}\left(s_k^r(i,j), s_{\bar{k}}^r(i,j)\right) \tag{6}$$

$$H_{pair} = \sum_{i=1}^{n-3} \sum_{j=1}^{(N-i-1)/2} P_{i,1+i+2j} \sum_{k=1}^{D} a_k(i, 1+i+2j) \tag{7}$$

As we have explicit formulae to determine the cost Hamiltonian, the elements of $H_C$ can be computed and the QAOA can then be simulated.

The same group in 2018 also made an attempt at using a quantum annealer to protein fold in the 3D MJ model [2]. Instead of a one-hot encoding, the group used 3 qubits to encode each bond and used a Hamiltonian of the form

$$H = H_{\text{back}} + H_{\text{redun}} + H_{\text{olap}} + H_{pair}$$

with each term respectively corresponding to a penalty to prevent bonds from reversing onto the previous bond, a penalty on 'redundant' bitstrings (000 and 011), a residue self-intersection penalty, and pair interaction energies. With this attempt, they report a $R_{99}$ value of $R_{99} =$ 190 262, meaning with the final state, 190 626 measurements are required for a 99% confidence of measuring the native state at least once (more details on page 56).

A number other papers exist which propose algorithms for protein folding on quantum computers, such as [95], and [96] which also claims to also simulate the dynamics of protein folding using a potential with polynomial complexity in the number of atoms to be simulated.

## 3.6 Our Proposed Approach

Universal quantum computers and quantum annealers are currently being developed to be the first generation practical quantum computers. As mentioned previously, there have been attempts to perform protein folding with current quantum annealers, but we chose to pursue a quantum algorithm which could run on a gate model quantum computer.

We also decided on the use of the QAOA, as its ability to search large parameter spaces was promising. The only significant hurdle was the construction of the objective function, in this case the energy function (as a function of the conformation, or its corresponding bitstring). Many such constructions of objective functions only provide mathematical expressions for the desired state but do not provide an explicit quantum circuit for constructing such a state (as this is often difficult in itself), so during the development of the energy function, possible circuit implementation was kept in mind.

Current universal quantum computers range around 10-50 qubits and have significant error due to current physical limits, so we will rely on simulating the algorithm on a classical computer

to provide insights and analysis of the proposed approach, instead of attempting to run the algorithm on a currently available quantum computer.

This is the approach taken by Fingerhuth, Babej and Ing in [3], however we propose a number of differences.

1. We will design our algorithm to work on a gate model quantum computer which utilises quantum gates to manipulate qubits, as opposed to constructing a Hamiltonian for encoding into an adiabatic quantum computer.
2. We will focus on an algorithm for which we can construct an explicit quantum circuit, to ensure that the algorithm can be performed efficiently on an actual quantum device as well as theoretically.

# 4 Mathematical Framework for Classical and Quantum Protein Folding

## 4.1 Problem Definition

We begin by first defining the problem we wish to solve.

A protein can be modelled as a self-avoiding walk on a 3D lattice beginning at the origin, where the $i$th node/vertex in the walk represents the $i$th amino acid residue from a given sequence of residues. Examples of such walks are shown in Figures 4 and 5.

Each particular walk (which we refer to a conformation) can be assigned an energy, depending on the energy model used. For simplicity we consider the HP (Hydrophilic-Hydrophobic) model, in which the energy is directly proportional to the number of adjacent H-H pairs (pairs of H residues which are on adjacent lattice points) which are not adjacent in sequence.

We aim to determine which conformation/s have minimal energy, according to the chosen energy model and a given sequence of residues. More specifically, we aim to construct an efficient quantum algorithm and circuit where we measurements can tell us which conformations have minimal energy and thus are the native state for that sequence. However, as we are limited to simulating our algorithms on classical computers, we will limit ourselves to relatively small chains of amino acids such that the required computations can be performed on a classical supercomputer.

## 4.2 Classical Algorithm

### 4.2.1 Conformation encoding

To perform the computation on a classical computer, we first require a way of representing conformations in an efficient manner. The method used is that used by [1] and [97], which is explained here for completeness.

In the HP model, bonded residues occupy adjacent lattice points, so instead of storing the individual coordinates of each residue, we can completely specify the conformation by specifying the absolute direction of each bond (i.e. specifying the displacement vectors between successive residues). Consider the conformation in Figure 7 in the HP model with a 2D square lattice. We can use a pair of bits to specify the direction of each bond. Specifically, 00, 01, 10 and 11 represent downwards, right, left and upwards bonds respectively.

The conformation in Figure 7 contains 6 amino acids, and thus 5 bonds. The direction of each bond is contained in the bitstring 01 00 10 10 11, with each successive pair of bits specifying

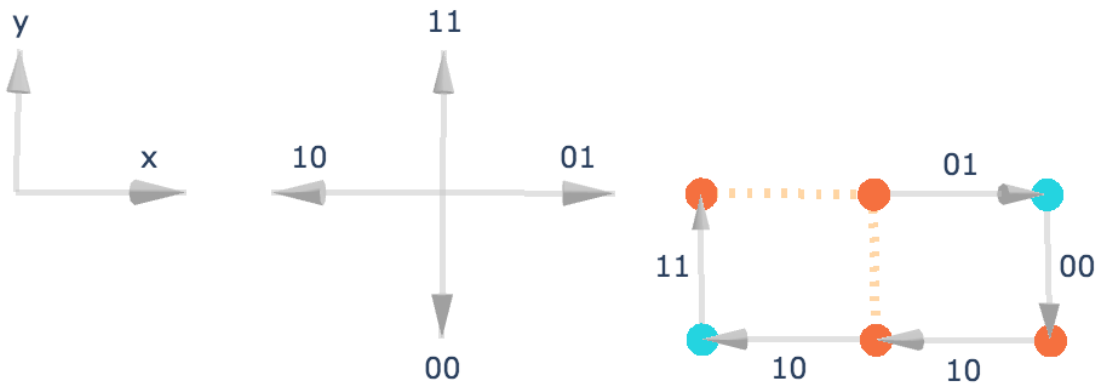



Figure 7: Direction encoding for the 2D lattice HP model as used in [1] and an example chain with bitstring 01 00 10 10 11. Orange points denote hydrophobic (H) residues, and blue points denote hydrophilic (P) residues. Grey arrows between points denote peptide bonds between sequential residues. Orange dotted lines denote H-H interactions which contribute towards the total energy of the conformation.

the direction of each successive bond. Note that the bitstring can be interpreted as a binary number, which in this case has the decimal value $299 = 2^8 + 2^5 + 2^3 + 2^1 + 2^0 = 0100101011_2$

This allows a natural way of encoding all possible conformations - by considering all possible values the bitstring can take.

For sequence with 6 residues and therefore 5 bonds, there are $2^{2\times5}$ possible bitstrings, corresponding to integers from 0 to $1023 = 2^{2\times5} - 1$. However, many of these conformations are identical up to rotational and/or reflective symmetry, so we can fix the first three bits of the bitstring to be 010, now reducing the number of possible bitstrings to $128 = 2^{2\times5-3}$.

Proteins exist and fold in 3 dimensions, so we can also extend this model to a 3D cubic lattice. Representing bond directions as in Figure 8 we can encode conformations similarly, but we require 3 qubits to fully specify each bond direction. A 3D cubic lattice is shown in Figure 9) for illustration purposes.

In the 2D case, there are 4 directions (positive and negative, in both x and y directions), which can have a one-to-one correspondence to the 4 possible bitstrings from 2 bits. However in the 3D case, there are 6 such directions, and 3 bits (corresponding to 8 directions) are needed to encode each bond. If a one-to-one correspondence is used, this leaves two bitstrings without corresponding directions. For a concrete example, consider the encoding in Figure 8. Note that in this encoding, the bitstrings '000' and '111' have no corresponding direction.

We can resolve this issue two ways:

1. We can interpret states with conformation bitstrings containing '000' and '111' bitstrings to be 'invalid', and we can assign an energy penalty (which we refer to as the conformation's 'invalidity') to such states so they cannot have minimal energy. However for the

sake of our later quantum implementation, we will assign directions to these bitstrings anyway - specifically, '000' will be the same direction as '001' and '111' will be the same direction as '100'. We refer to this method as using an 'invalidity penalty'.

2. We can interpret states with conformation bitstrings containing '000' and '111' bitstrings as 'valid', and assign these bitstrings a direction as above but without an energy penalty. We refer to this method as having 'duplicate bonds'.

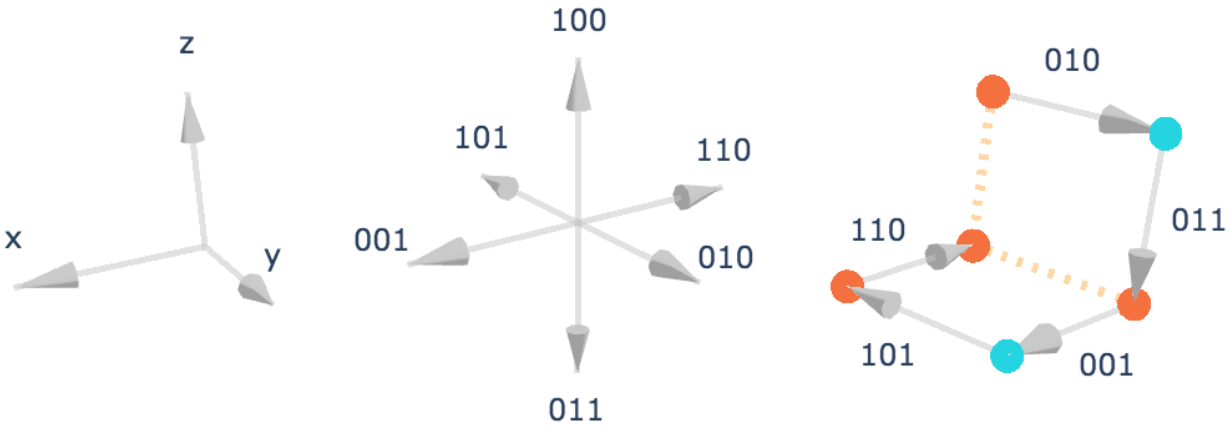



Figure 8: Direction encoding for the HP model on a 3D cubic lattice used for this thesis and the conformation with bitstring (010 011 001 101 110). Orange and blue points denote H and P residues respectively, and dotted orange lines denote H-H interactions.

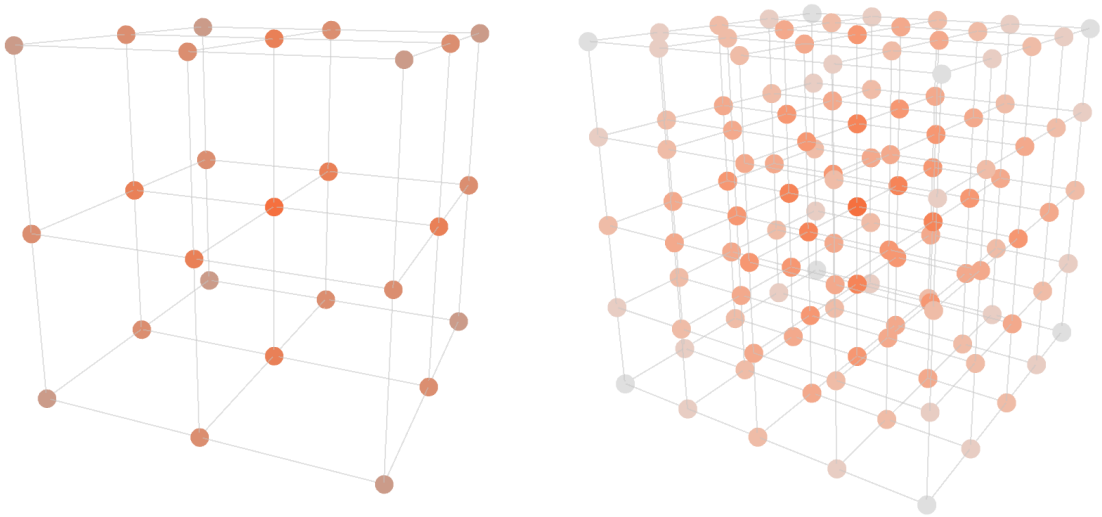


Figure 9: A small and large 3D cubic lattice. Colour indicates distance away from the origin/centre.

A third solution to this is to change the lattice type so that there are indeed 8 possible directions for each bond, so that all bond conformations are again 'valid' without having to double-assign.

This is possible with an encoding as shown in Figure 10 with a Body-Centred Cubic (BCC) lattice, as shown in Figure 11.

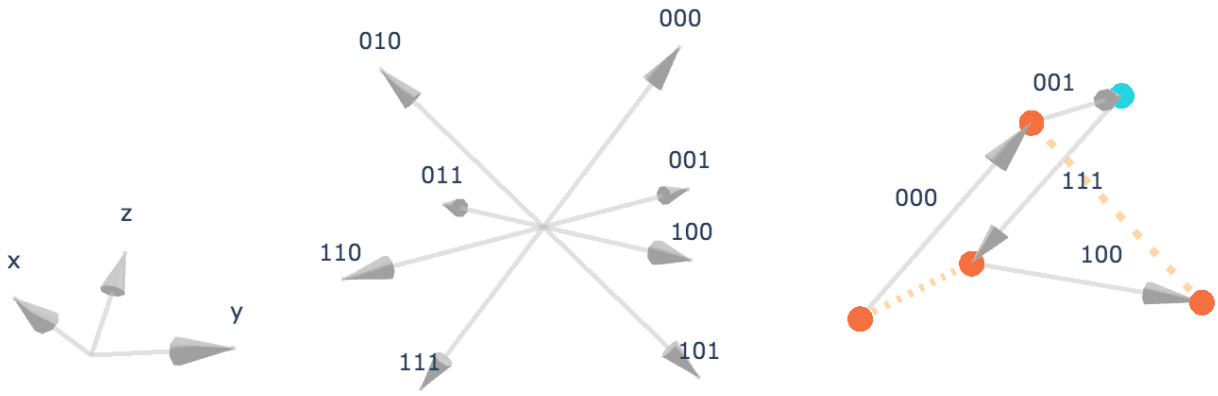



Figure 10: Direction encoding for the 3D body-centred cubic (BCC) model used for this thesis and the conformation with bitstring (000 001 111 100). Orange and blue points denote H and P residues respectively, and dotted orange lines denote H-H interactions.

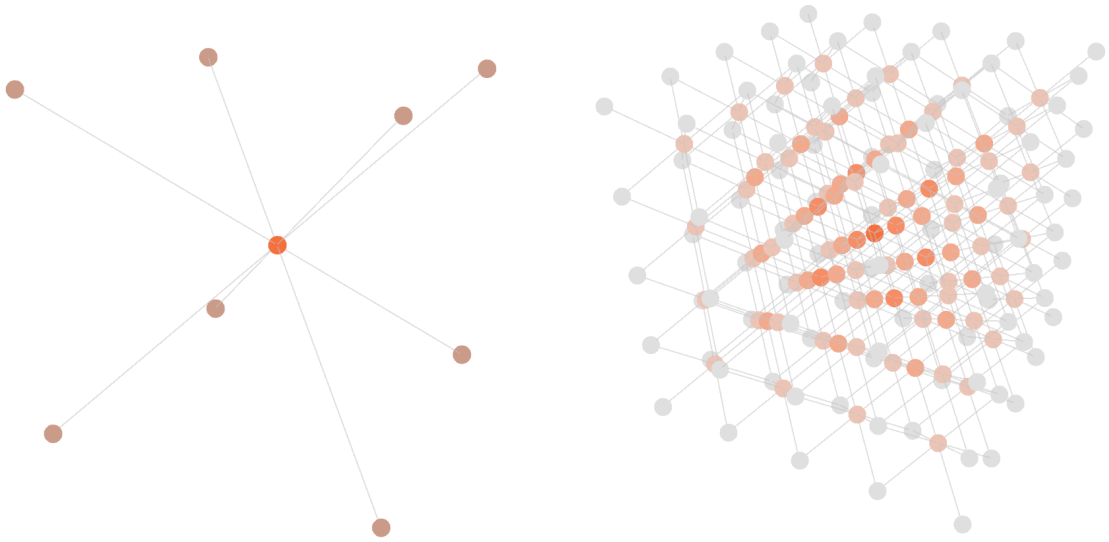


Figure 11: A small and large 3D BCC lattice. Colour indicates distance away from the origin/centre.

Now considering these 3D lattices, we require $3m$ bits to encode a conformation of $m$ bonds, so there are $2^{3m}$ possible bitstrings. As with the 2D case, many of these conformations are identical up to symmetries. In the 3D cubic case, we can fix the first 5 bits to '010 01', thus fixing the first bond in the +y direction, and the second bond to either be parallel, or perpendicular (in the -z direction) to the first bond. In the 3D BCC case, we can fix the first 4 bits to '000 0', thus fixing the first bond. The second bond can either be parallel with the first (fixed to 000), at an angle of $\cos^{-1}\left(-\frac{1}{3}\right) \approx 109°$ (fixed to 010 or 001), or at an angle of $\cos^{-1}\left(\frac{1}{3}\right) \approx 71°$ (fixed to 011).

Encoding conformations as bitstrings which can then be represented as a range of integers is useful, as all physically possible conformations can be found exhaustively. However, in addition to any conformations we deem 'invalid', some bitstrings correspond to conformations which are not physically possible because of self-intersections, such as in Figure 12. This and the calculation of conformation energies are the subject of the next section.

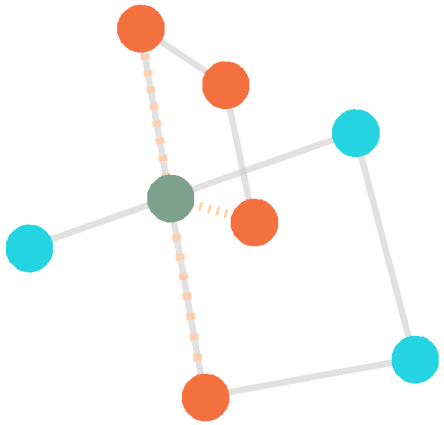


Figure 12: A physically infeasible conformation, as such a combination of directional bonds would result in different residues occupying the same lattice point in space. In this case, a H and P residue both occupy the same coordinates, and the algorithm counts its interactions with the three neighbouring H residues.

### 4.2.2 Conformation energies

We can now begin to determine the energies of our conformations by considering their bitstrings. To do this, we use the bitstring to generate the list of residue coordinates in the conformation, then calculate the energy from these coordinates.

Generating the list of coordinates is straightforward. The 0th coordinate is set to the origin, then each following coordinate is the previous coordinate plus the next bond specified by the bitstring. The positional changes corresponding to each bitstring used in this thesis are summarised in Table 2.

In the HP model, only adjacent H-H pairs of amino acids lower the energy of the conformation, and all other adjacent pairs do not change the energy, so determining the minimal energy conformation is a question of counting the number of these adjacent H-H pairs in each conformation. However, we will only count H-H pairs where the pairs are not adjacent in the residue sequence (e.g. if residues 2 and 3 in the sequence are type H, we do not count them towards the energy calculation, as they are adjacent in all possible conformations and we are only interested in relative differences in energy between conformations).

| Bits | Bond vector |
|---|---|
| | 2D cubic |
| 00 | (0,-1) |
| 01 | (1,0) |
| 10 | (-1,0) |
| 11 | (0,1) |

| Bits | Bond vector | |
|---|---|---|
| | 3D cubic | 3D BCC |
| 000 | (1,0,0)* | (1,1,1) |
| 001 | (1,0,0) | (-1,1,1) |
| 010 | (0,1,0) | (1,-1,1) |
| 011 | (0,0,-1) | (-1,-1,1) |
| 100 | (0,0,1) | (1,1,-1) |
| 101 | (0,-1,0) | (-1,1,-1) |
| 110 | (-1,0,0) | (1,-1,-1) |
| 111 | (0,0,1)* | (-1,-1,-1) |

Table 2: Correspondence between bit tuples and bonds for the 2D cubic, 3D cubic and 3D BCC models. Bond vectors with a * are 'duplicated' bonds, and so may be penalised depending on the model used. The correspondence used for the 2D cubic case is used in [1].

Recall that some bitstrings do not correspond to physically real conformations, as mentioned previously. There are two reasons this may be:

1. The directions specified by the bitstring would result in self-intersection/s of the protein - that is, two or more amino acids would occupy the same coordinates, or
2. A bitstring contains 'invalid' bitstrings, and we interpret the corresponding conformations as not physically real.

We can easily penalise instances of the second case by checking the bitstring of each bond to ensure none are invalid, but in general it is difficult to identify whether a bitstring's conformation is self-intersecting. Recall that we are only interested in determining minimal energy conformations, so instead of attempting to remove instances of these cases from the calculation entirely, we can instead assign energy penalties to conformations with these, so no such conformations can be of minimal energy. In other words, we are applying a soft constraint onto the energy of the conformation to 'discourage' these self-intersecting conformations.

Considering all the above contributions to the energy for a particular bitstring $z$ corresponding to a sequence with $m+1$ residues (and thus $m$ bonds), we can write this energy as

$$\begin{aligned} E^z &= \sum_{i=0}^{m-1} \sum_{j=i+1}^{m-1} \left( E_- \left[ \text{Dist}(\mathbf{r}_i^z, \mathbf{r}_j^z) = 1 \right] + E_+ \left[ \text{Dist}(\mathbf{r}_i^z, \mathbf{r}_j^z) = 0 \right] \right) \\ &\quad + \sum_{i=0} E_{++} \left[ z_i = 000 \vee z_i = 111 \right] \qquad (8) \\ &= \sum_{\substack{0<i,j<m \\ i<j \\ \text{Dist}(\mathbf{r}_i^z, \mathbf{r}_j^z)=1}} E_- \quad + \sum_{\substack{0<i,j<m \\ i<j \\ \text{Dist}(\mathbf{r}_i^z, \mathbf{r}_j^z)=0}} E_+ \quad + \sum_{\substack{0<i<m \\ z_i \text{ invalid}}} E_{++} \qquad (9) \end{aligned}$$

where $E_-$, $E_+$ and $E_{++}$ are the energies associated with each H-H interaction, self-intersection

and invalid bond respectively, $\mathbf{r}_i^z$ is the position vector of residue number $i$ corresponding to bitstring $z$, square brackets denote a boolean value that can be taken as 0 for False and 1 for True, and $\vee$ denotes the logical OR operator. $\text{Dist}(\mathbf{r}_i^z, \mathbf{r}_j^z)$ is the distance between lattice points $\mathbf{r}_i^z$ and $\mathbf{r}_j^z$ in the particular lattice model used. For a 3D cubic lattice and 3D BCC lattice respectively, this is given by

$$\text{Dist}_{cubic}(\mathbf{r}_i^z, \mathbf{r}_j^z) = \left|x_j^z - x_i^z\right| + \left|y_j^z - y_i^z\right| + \left|z_j^z - z_i^z\right| \tag{10}$$

$$\text{Dist}_{BCC}(\mathbf{r}_i^z, \mathbf{r}_j^z) = \max\left(\left|x_j^z - x_i^z\right|, \left|y_j^z - y_i^z\right|, \left|z_j^z - z_i^z\right|\right) \tag{11}$$

We now have a way of generating all possible conformations and calculating their corresponding energies. All that is left is to keep track of the lowest energy so far in the calculation and its corresponding bitstring. To show the algorithm more concretely, pseudocode for it is provided below.

Classical lattice folding algorithm pseudocode

```
min_energy = Infinity
min_energy_bitstring = 0
for bitstring in [bitstring_min, ..., bitstring_max]:
    coord_list = generate_coordinates(bitstring)
    energy = 0
    for j in [0, 1, ..., m − 1]:
        if penalise_invalids == True:
            if bitstring[j] == 000 or bitstring[j] == 111:
                energy = energy + 10
        for k in [j + 1, j + 2, ..., m − 1]:
            if dist(coord_list[j], coord_list[k]) == 0:
                energy = energy + 5
            elif dist(coord_list[j], coord_list[k]) == 1:
                energy = energy - 1
    if energy <min_energy:
        min_energy = energy
        min_energy_bitstring = bitstring
return min_energy, min_energy_bitstring
```

Pseudocode 1: Pseudocode for the classical lattice folding algorithm. coord_list[$i$] gives the coordinates of the $i$th residue for that particular bitstring. bitstring[$i$] gives the substring within bitstring corresponding to the $i$th bond.

### 4.2.3 Scaling of the computation

Arguably the most important aspect to the algorithm is how it scales with more residues considered, so we will examine this now.

The generation of coordinates and counting of the energy has to be repeated as many times as there are possible bitstrings, and the number of possible bitstrings increases exponentially with the increasing number of residues. Specifically, for the 3D cubic case, every additional residue introduces another bond which requires an additional 3 bits. Thus, every additional residue increases the number of computations required by 8 times.

However, looking at the other aspects of the algorithm, we find that their scaling is much better. Generating the list of coordinates scales linearly with more residues, and the comparison of pairs of residue coordinates scales quadratically.

If it is possible to reduce the exponential scaling over the number of possible bitstrings to a polynomial scaling, then the algorithm becomes only polynomial in the number of residues. One question is whether it is possible to cleverly filter out self-intersecting conformations, then only calculate conformation energies over the non-intersecting conformations. Figure 13 shows that in both 3D cubic and BCC models, both the number of non-intersecting and self-intersecting conformations increase exponentially with more residues. Figure 14 shows that as the number of residues increase, a larger proportion of bitstrings form self-intersecting conformations. Thus, a subroutine which cleverly filters out self-intersecting conformations so would be useful for increasing numbers of residues, but the reduction in search space would not enough to offset the search space's exponential growth.

It is with this exponential scaling in mind that we turn our attention towards the potential of quantum computing.

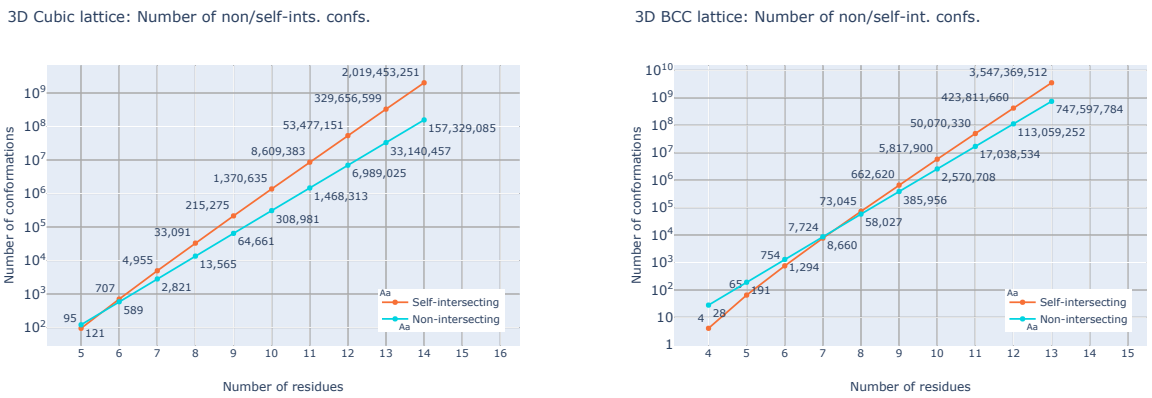



Figure 13: Number of self-intersecting (orange) and non-intersecting (blue) conformations over number of residues with 3D cubic (left) and 3D BCC (right) lattices. Directions of some bonds are restricted due to symmetries. Numbers for self-intersections and non-intersections are above-left and below-right of their corresponding points respectively.

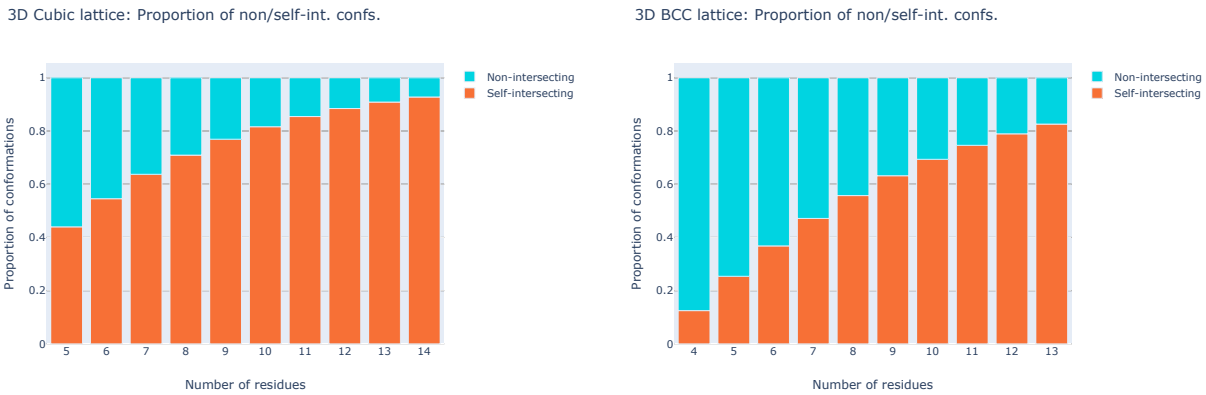



Figure 14: Proportion of self-intersecting (orange) and non-intersecting (blue) conformations over number of residues with 3D cubic (left) and BCC (right) lattices. Directions of some bonds are restricted due to symmetries.

## 4.3 Quantum Algorithm

Once it is possible to compute the conformation energy as in the classical case, an optimisation scheme such as QAOA can then be used to determine these minimal energy conformations. We propose Circuit 3 as a quantum circuit for finding minimal energy conformations as described in the Classical Algorithm section.

The details of the sub-circuits comprising Circuit 3 will be explained in Sections 4.3.2 to 4.3.4, but to summarise the logic of the circuit, we:

1. Select values for the parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p-1}) \in [0, \pi)^{\otimes p}$ and $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, \ldots, \gamma_{p-1}) \in [0, 2\pi)^{\otimes p}$.
2. Begin with the $|0\rangle^{\otimes 3m}$ state, then initialise our superposition $|\Psi\rangle$ which encodes all possible conformations as bitstrings. Some of the first qubits are fixed to $|0\rangle$ or $|1\rangle$ due to symmetry arguments explained in Section 4.2.1
3. Apply the $U_C(\gamma_0)$ operator to obtain

$$U_C(\gamma_0) |\Psi\rangle = e^{-iC\gamma_0} |\Psi\rangle \tag{12}$$

   where $C$ is the diagonal matrix of conformation energies, which we are trying to find the minimum of. $C$ is implemented here as a circuit (Circuit 7) which calculates these energies in three steps:

   (a) Initialise some ancillary qubits in the $|0\rangle$ state. For the conformation encoded by bitstring $s$, which itself is encoded in the superposition $|\Psi\rangle$, generate the list of residue coordinates for the conformation using the ancillary qubits. The result is

the state

$$|\Psi, \mathbf{x}, \mathbf{y}, \mathbf{z}\rangle = \sum_s \alpha^s |s\rangle |\mathbf{x}^s\rangle |\mathbf{y}^s\rangle |\mathbf{z}^s\rangle \tag{13}$$

where $\mathbf{x}^s$ contains the x coordinates of each residue, and similarly for $\mathbf{y}^s$ and $\mathbf{z}^s$. Each bitstring $|s\rangle$ is now entangled with the coordinates of its conformation in the ancillary register.

(b) Initialise the energy register as $|0\rangle$.

(c) If using an invalidity penalty, check the bitstrings of each bond in the conformation bitstring $s$. Increment the energy register by $E_{++}$ for each invalid bond.

(d) From the list of coordinates, check each pair of residues (except those adjacent in sequence). If any pairs occupy the same coordinates, increment the energy register by $E_+$. If any pairs occupy adjacent coordinates and both residues are of type H, increment the energy register by the negative quantity $E_-$.

(e) Reset the ancillary qubits containing the list of coordinates back to the zero state.

Now we have an implementation of $C$ as a circuit, we can use Circuit 6 to perform the $U_\mathrm{C}$ operator.

4. Apply the $U_\mathrm{B}(\beta_0)$ operator to obtain

$$U_\mathrm{B}(\beta_0)e^{-iC\gamma_0} |\Psi\rangle = e^{-iB\beta_0}e^{-iC\gamma_0} |\Psi\rangle \tag{14}$$

where $B$ is the 'mixer' matrix used. We will consider two types of mixer, the hypercube adjacency matrix implemented by Circuit 14, and the complete graph adjacency matrix implemented by Circuit 15.

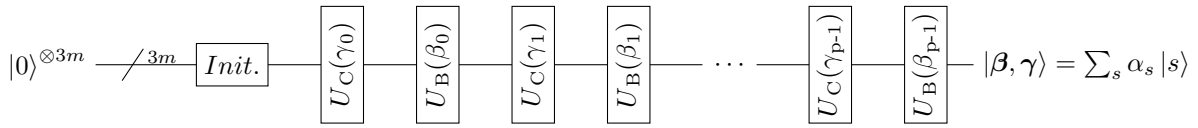
5. Repeat steps 3 and 4 a total of $p$ times with the remaining $\beta_i$ and $\gamma_i$ parameters. We then obtain the state

$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = e^{-iB\beta_{p-1}}e^{-iC\gamma_{p-1}} \dots e^{-iB\beta_0}e^{-iC\gamma_0} |\Psi\rangle \tag{15}$$

6. Measure the $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ state to obtain a bitstring $s$. We can then classically calculate the energy of the conformation corresponding to $s$. We reproduce and measure the $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ state multiple times to obtain an estimate for $\langle C\rangle$ for the current $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters.

7. External to the circuit we have a classical optimisation scheme which is used to minimise $\langle C\rangle$ with respect to the parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. With this we can find a local minima of $\langle C\rangle$ with its corresponding $\boldsymbol{\beta}_{min}$ and $\boldsymbol{\gamma}_{min}$ parameters. The state $|\boldsymbol{\beta}_{min}, \boldsymbol{\gamma}_{min}\rangle$ should have significant probabilities of measuring bitstrings with low-energy conformations.

Circuit 3: $[QAOA]$




| | $3m$-bit Toffoli | $3m$-bit Fourier | 3-bit Toffoli | $+M$ |
|---|---|---|---|---|
| **Gates** | $2p$ | $2p$ | $4pm(mn+5)$ | $6p(m^2-2m)$ |

| | $3n$-Control $+M$ | $X$ | $R_z$ | Depth |
|---|---|---|---|---|
| **Gates** | $9p(m^2-3m)$ | $O(pm^2)$ | $pk$ | $O(m^2n^2p)$ |

Circuit 3: Circuit for finding minimal energy conformations using the QAOA, parameterised by $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p-1}) \in [0, \pi)^{\otimes p}$ and $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, \ldots, \beta_{p-1}) \in [0, 2\pi)^{\otimes p}$. $m+1$ is the number of residues, thus $m$ is the number of bonds. $n = \lceil \log_2 m \rceil$ is the maximum number of qubits required to store any single $x$, $y$ or $z$ coordinate. $p$ is the QAOA depth of the circuit, and $k$ is the maximum number of qubits required to store a single conformation's energy. $Init.$ prepares the initial superposition of all considered states using only Hadamard ($H$) and NOT ($X$) gates depending on the model used. The following $U_C$ and $U_B$ gates perform the QAOA evolution to obtain the $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ state. The $U_C$ and $U_B$ gates also use ancillary qubits, however these are reset after each gate, thus they are not shown here. The table of gate and depth complexities is explained in Section 4.3.1. A total of $3n(m+2) + 3m + k + 3$ qubits are required for this circuit, with $3m$ qubits in the main register, and $3n(m+2) + k + 3$ ancilla from $U_C$ gates.

The following sections explain the subcircuits of Circuit 3 in more detail. However, for concreteness at each stage we will consider an example model with the following attributes to illustrate the logic of the circuit:

- A 'HHHHH' chain.
  - The chain consists of five hydrophobic residues, meaning that H-H interactions can occur between any two non-bonded residues.
- Using a 3D cubic lattice model.
- With energy parameters $(E_-, E_+, E_{++}) = (-1, 5, 10)$.
  - An energy of $-1$ is assigned to each pair of non-bonded, adjacent H-H residues, an energy of $+5$ is assigned to each pair of residues which occupy the same coordinates, and an energy of $+10$ is assigned to each 'invalid' triple of bits in the bitstring. We will consider '000' and '111' as invalid, as defined previously.
- With a complete mixer.

### 4.3.1 Circuit complexity calculation

With classical computing, we can measuring the 'difficulty' or computational power required for a computation in terms of the number of operations it requires, and how this number scales as the problem size increases.

In quantum computing, we can similarly measure the computational resources we require by analysing the proposed circuits. In particular, we can count the number of quantum gates required, the depth of the circuit, and the number of qubits required.

Each circuit diagram includes a table summarising the number of gates required for that circuit in terms of some number of 'base' gates. For concreteness, consider the table provided in Circuit 4. This indicates that this circuit requires a total of $3m - 5$ Hadamard and 2 $X$ (NOT) gates, where $m$ is the number of bonds between the $m + 1$ residues. The depth of the circuit can intuitively be seen as the 'width' of the circuit across the page, where as many gates are performed at the same time as others. As all gates in Circuit 4 can be performed simultaneously, the depth is 1.

In calculating the complexity of these circuits, we use number of rules and definitions:

- At each time step in a circuit, any given qubit can either not interact, be acted upon by a gate, or be a control qubit for one other gate. In particular, we assume that a qubit cannot simultaneously control multiple gates, nor can it be acted on at the same time it is used as a control. In circuits such as Circuit 8 where it is drawn that multiple gates are controlled from the same qubit, it should be understood that each individual gate is performed sequentially.
- The 'worst case scenario' is used for each calculation, meaning when given an choice such as using a 3D cubic lattice or a 3D BCC lattice, the calculation uses that which has the highest complexity. An example is with Circuit 8, whose given complexity assumes use of the cubic $W$ gate.
- Connectivity between all qubits. In reality, there may be restrictions on which qubits can interact based on the physical implementation of the quantum computer, but we neglect this as connectivity can vary widely, and the problem of connectivity can be fixed using a number of SWAP gates linear in $m$ [91].
- All single qubit gates and singly controlled single qubits gates (including CNOT) have a depth of 1. That is, we assume that these gates can be performed in a single operation on the quantum computer. In reality, more complex gates can also be performed using a single operation, depending on the implementation of the quantum computer, such as in [98]. Such operations can further reduce the number of gates required and the depth of the circuit.
- Singly controlled gates can be implemented with the same depth as without control, with

a constant overhead [91].

- Multiple controls can be implemented with controlled phase operations, with the number of gates quadratic in the number of controls, and depth linear in the number of controls [99].

### 4.3.2 State initialisation and the encoding of conformations

As in the classical computation, we need a way of encoding conformation information in the quantum circuit, specifically one which uses qubits.

A benefit of using bitstrings to encode conformations is that they can be directly mapped to the computational basis states of qubits. Consider $m+1$ residues in a 3D cubic lattice. We have $m$ bonds to encode and this we require $3m$ qubits. After setting the first five bits to '01001' due to symmetries, we have $3m-5$ bits remaining, and thus $2^{3m-5}$ bitstrings to consider. We can then encode all $2^{3m-5}$ bitstrings in the superposition $|\Psi\rangle$

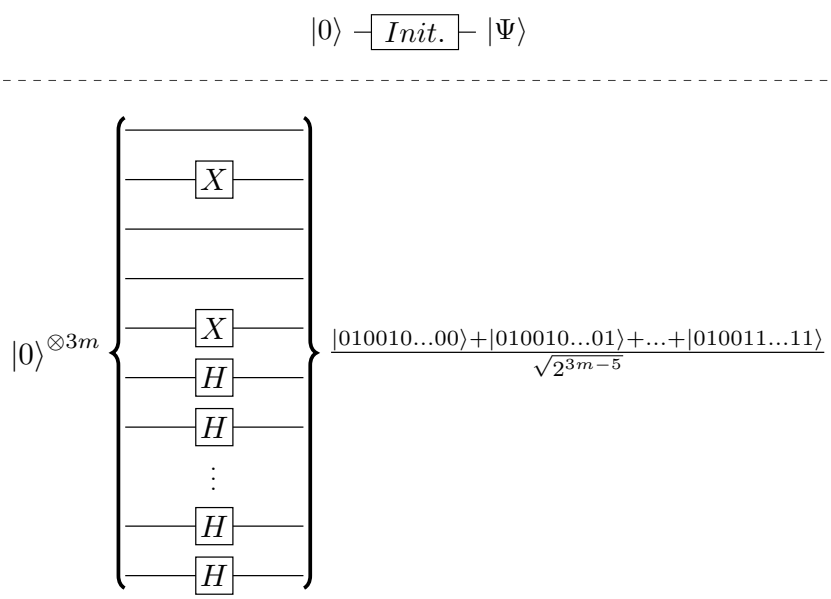
$$|0\rangle^{\otimes 3m} \rightarrow |\Psi\rangle \tag{16}$$

$$= |01001\rangle\, H^{\otimes 3m-5}\, |0\rangle^{\otimes 3m-5} \tag{17}$$

$$= \frac{1}{\sqrt{2^{3m-5}}}\, |01001\rangle \otimes (|00\ldots00\rangle + |0\ldots01\rangle + \ldots + |11\ldots11\rangle) \tag{18}$$

The initialisation of this superposition $|\Psi\rangle$ is performed by Circuit 4. We can similarly prepare the initial superposition for $m+1$ residues in a 3D BCC lattice, with the difference of setting the first four qubits to $|0000\rangle$, performed by Circuit 5.

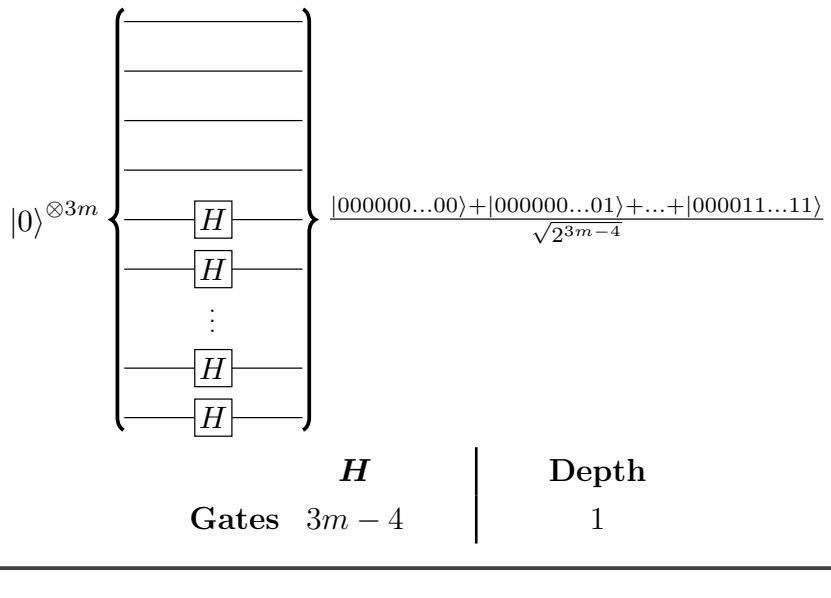
Circuit 4: [*Init.*] for a 3D cubic lattice




| | ***H*** | ***X*** | **Depth** |
|---|---|---|---|
| **Gates** | $3m-5$ | 2 | 1 |

Circuit 4: Bitstring superposition initialisation for the 3D cubic model. The first five qubits of the output are set to $|01001\rangle$ and all other qubits are in the superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

Circuit 5: [*Init.*] for a 3D BCC lattice




| | ***H*** | **Depth** |
|---|---|---|
| **Gates** | $3m-4$ | 1 |

Circuit 5: Bitstring superposition initialisation for the 3D BCC model. The first four qubits of the output are set to $|0000\rangle$ and all other qubits as in the superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

Example

With our 'HHHHH' chain, we have $m = 5 - 1 = 4$, and so we want to consider all bitstrings from 010 010 000 000 to 010 011 111 111 (inclusive). The first five bits are set as '010 01' due to symmetry arguments. To obtain an equal superposition of these bitstrings, we apply the Hadamard gate to 6th to 12th qubits. We want bits 1, 3 and 4 to remain as 0, so these qubits are untouched. We also want bits 2 and 5 to remain as 1, so we use NOT gates to flip the corresponding qubits to the $|1\rangle$ state. The result is our superposition of states $|\Psi\rangle$

$$|\Psi\rangle = \frac{1}{\sqrt{2^7}} \left(|010, 010, 000, 000\rangle + \ldots + |010, 011, 111, 111\rangle\right) \tag{19}$$

We can also write the state as a sum over bitstring states $s$ (i.e. computational basis states)

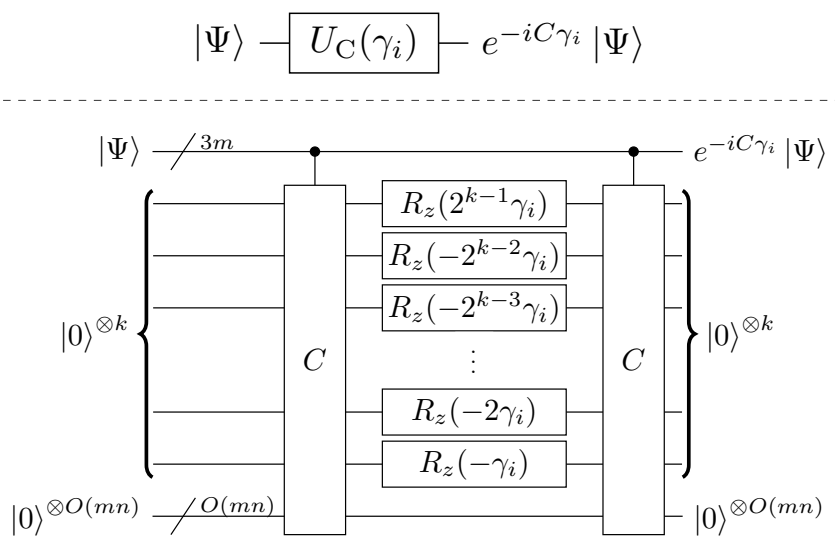
$$|\Psi\rangle = \sum_s \alpha^s |s\rangle \tag{20}$$

We will assume that the sum over $s$ is only over bitstrings we are considering (those between 010 010 000 000 and 010 011 111 111), so each $\alpha^s$ coefficient is $\frac{1}{\sqrt{2^7}}$.

### 4.3.3 Unitary phase operator $U_{\mathbf{C}}(\gamma_i)$

Circuit 6 (adapted from the circuit for the exponentiation of diagonal Hamiltonians in [7]) performs the $U_{\mathrm{C}}(\gamma_i)$ operator for a parameter $\gamma_i$ with a given value.

Circuit 6: $[U_C(\gamma_i)]$




| | **3-bit Toffoli** | **+M** | **3n-Control +M** | **X** | **$R_z$** | **Depth** |
|---|---|---|---|---|---|---|
| **Gates** | $4m(3n+5)$ | $6(m^2-2m)$ | $9(m^2-3m)$ | $O(m^2)$ | $k$ | $O(m^2n^2)$ |

Circuit 6: Unitary phase shift operator parameterised by $\gamma_i \in [0, 2\pi)$, adapted from [7]. This gate is applied alternately with the unitary mixing operator to perform the QAOA evolution. $k = O(n)$ is the maximum number of qubits required to store a single conformation's energy.

Example

With our superposition $|\Psi\rangle = \sum_s \alpha^s |s\rangle$, $U_C$ applies a phase shift to each $\alpha^s$ according to the conformation energy of each $s$. We consider a bitstring in the superposition to illustrate.

For $s$ = 010 011 101 101, we have residue coordinates $\mathbf{r}_0^s = (0,0,0)$, $\mathbf{r}_1^s = (0,1,0)$, $\mathbf{r}_2^s = (-1,1,0)$, $\mathbf{r}_3^s = (-1,0,0)$, $\mathbf{r}_4^s = (-1,-1,0)$. There is an interaction between residues 0 and 3, and no self-intersection or invalid bitstrings, so $E^s = -1$. Thus $U_C(\gamma_i)$ maps $|010\ 011\ 101\ 101\rangle \to e^{i\gamma_i}|010\ 011\ 101\ 101\rangle$. This is done similarly for each other bitstring $s$ in $|\Psi\rangle$.

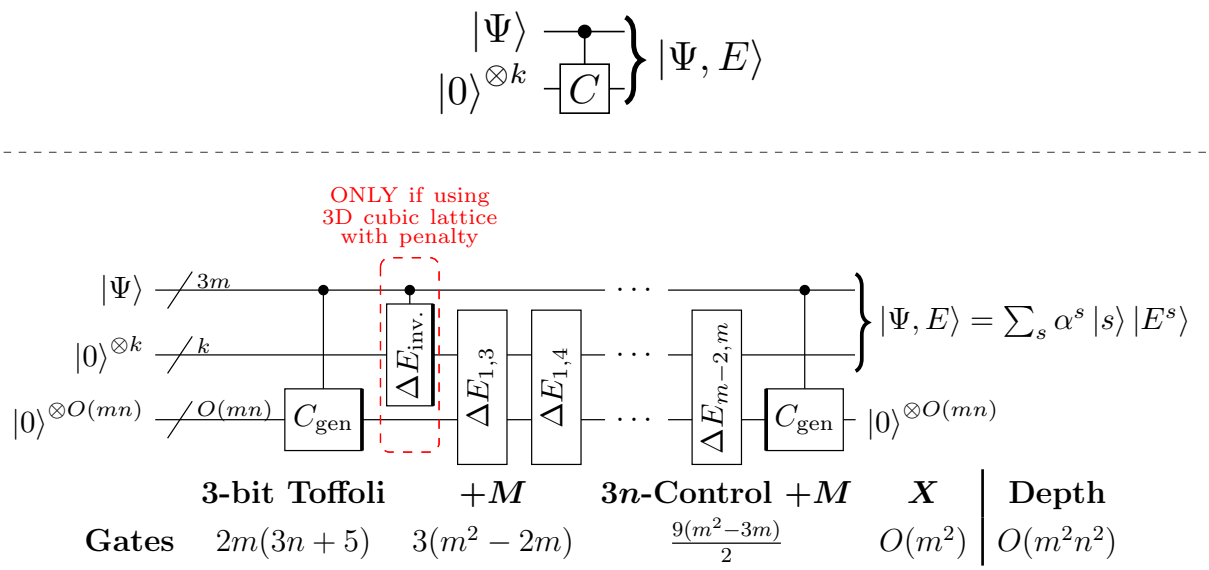
The gate $C$ performs the operation

$$|\Psi\rangle|0\rangle \to |\Psi, E\rangle \tag{21}$$

$$= \sum_s \alpha^s |s\rangle |E^s\rangle \tag{22}$$

where the bitstrings $s$ in the output state are entangled with their energies in the second (energy) register. This is implemented using Circuit 7.

Circuit 7: $[C]$




Circuit 7: Objective function operator $C$, composed of sub-circuits $C_{\text{gen}}$ and $\Delta E_{i,j}$. The first register contains bitstring conformations and the second register encodes conformation energies. The third register contains ancillary qubits required for the subcircuits. Note the bolded edges of both $C_{\text{gen}}$ operators, indicating that the first circuit is 'forwards', whilst the second circuit is 'backwards', as to reset the ancilla qubits to $|0\rangle$. The $\Delta E_{i,j}$ circuit is iterated over all pairs of residues i.e. $(1,3)$ to $(1,m)$, then $(2,4)$ to $(2,m)$, and so on until $(m-2,m)$, a total of $\frac{m^2-3m}{2}$ times. Circuit depth is $O(m^2n^2)$ from $O(m^2)$ $\Delta E_{i,j}$ gates with $O(n^2) = O(\log_2 m^2)$ depth each. $3n(m+2)+3$ ancillary qubits are required by $C_{\text{gen}}$ simultaneous to $k$ ancilla required for the energy, totalling $3n(m+2)+3+k$ ancillary qubits.

Circuit 7 is comprised of three parts. Firstly the gate $C_{\text{gen}}$ (Circuit 8) is used to generate the list of residue $x$, $y$ and $z$ coordinates corresponding to each bitstring $s$. These coordinates are stored in the ancillary qubit register in Circuit 7. The type of each residue (whether they are H or P residues) are also stored in the ancillary register and can be programmed before running the circuit based on the sequence to be simulated. After the coordinates have been generated, for each bitstring $s$ in $|\Psi\rangle$ we can check the coordinates and types of each pair of residues and add/take from the energy register if there is an interaction energy from adjacency or an energy penalty from self-intersection. A comparison between residue number $i$ and $j$ is carried out with the gate $\Delta E_{i,j}$ (Circuit 12 for the cubic model, 13 for the BCC model), and we can use this gate on each pair of residues, forming the second part of the $C$ operator. The resulting state is $\sum_s \alpha^s |s\rangle |E^s\rangle |x^s, y^s, z^s\rangle$, where the ancillary register contains coordinate information entangled with bitstrings and energies. To remove this entanglement we can run the $C_{\text{gen}}$ circuit in reverse, setting the ancillary register to $|0\rangle^{\otimes O(N)}$.

Example

Consider $s = 010\ 011\ 101\ 100$, with coordinates $\mathbf{r}_0^s = (0,0,0)$, $\mathbf{r}_1^s = (0,1,0)$, $\mathbf{r}_2^s = (-1,1,0)$, $\mathbf{r}_3^s = (-1,0,0)$, $\mathbf{r}_4^s = (0,0,0)$.

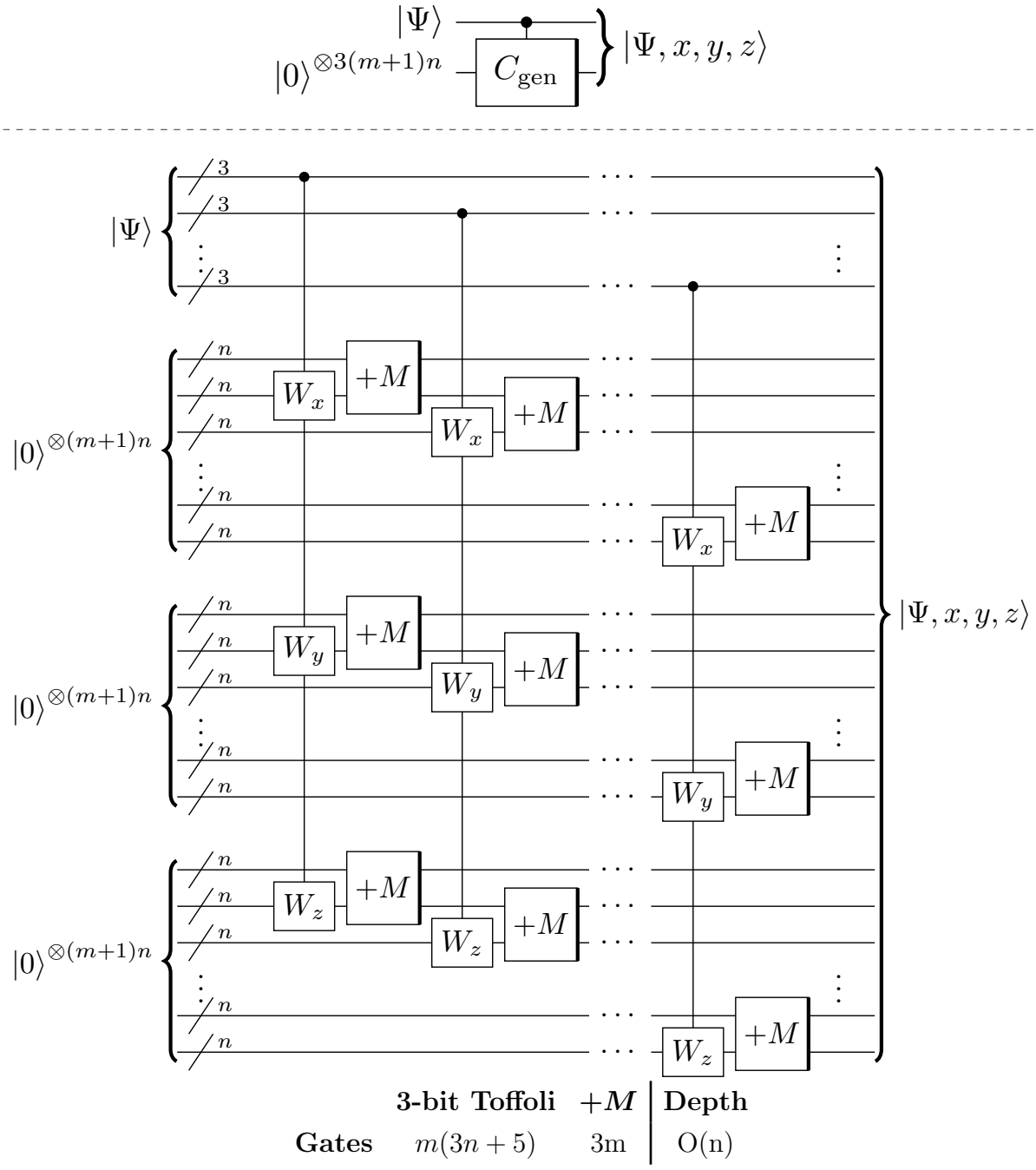
We start with $|s\rangle |0\rangle^{\otimes k)} |0\rangle^{\otimes O(mn)}$, where $s$ is a bitstring comprising the superposition state $|\Psi\rangle = \sum_s \alpha_s |s\rangle$. $C_{\text{gen}}$ calculates and stores the coordinates in the last register, resulting in $|s\rangle |0\rangle^k |x\rangle |y\rangle |z\rangle$. $|x\rangle$ itself is the state $|0\rangle |0\rangle |-1\rangle |-1\rangle |0\rangle$, corresponding to the $x$ component of each coordinate in order, and similarly for $|y\rangle$ and $|z\rangle$.

With the coordinates now encoded in ancillary qubits, the first $\Delta E_{0,2}$ gate checks if $\mathbf{r}_0^s$ and $\mathbf{r}_2^s$ intersect or are adjacent. They do not, so the energy register remains at $|0\rangle^{\otimes k)} = |0\rangle$. The next gate checks residues 0 and 3, and these are adjacent, so -1 is added to the energy register, which is now $|-1\rangle$. Checking residues 0 and 4, these are intersecting, so +5 is added to the energy register, which is now $|4\rangle$. The remaining checks are completed and the final energy of $|s\rangle$ is 4, so the resulting state is $|s\rangle |4\rangle |x\rangle |y\rangle |z\rangle$.

Finally we must reset $|x\rangle |y\rangle |z\rangle$ back to $|0\rangle^{\otimes O(m)}$. We can do this by simply applying the $C_{\text{gen}}$ gate from right to left to obtain $|s\rangle |4\rangle |0\rangle^{\otimes O(m)}$. This process is carried out simultaneously for each $|s\rangle$ in the superposition $|\Psi\rangle$.

The $C_{\text{gen}}$ circuit (Circuit 8) calculates the coordinates of each residue by first decoding the bond direction information from the bitstrings stored in $|\Psi\rangle$ using the $W$ gates for each respective direction. The 0th coordinates are also set at the origin, and subsequent residue coordinates are described by subsequent triples of qubits in $|\Psi\rangle$. The bond directions are then added to the previous coordinate values to obtain the current coordinate values, and the process is repeated for subsequent residues until all have been calculated.

Circuit 8: $[C_{\text{gen}}]$




| | 3-bit Toffoli | $+M$ | Depth |
|---|---|---|---|
| **Gates** | $m(3n+5)$ | 3m | O(n) |

Circuit 8: Coordinate list generator circuit $C_{\text{gen}}$, which generates the state $|\Psi, x, y, z\rangle = \sum_s \alpha_s |s\rangle |\mathbf{x}_s\rangle |\mathbf{y}_s\rangle |\mathbf{z}_s\rangle$. The first register contains bitstring conformations. The second register encodes the $x$ coordinates of each residue in the computational basis as $(x_0, x_1, \ldots, x_n)$. The third and fourth registers encode the $y$ and $z$ coordinates similarly. $3(m+1)n$ ancillary qubits are required for storing coordinate information, and a further $3(n+1)$ are required as working qubits for modular adders, totalling $3n(m+2)+3$ ancilla.

Example

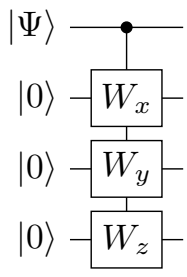
Circuit 8 works by decoding the difference in each coordinate between residues from the bitstring $s$, then adding this difference to previous residue's coordinates. We will only consider the $|\Psi\rangle$ and $y$ registers for now. For $s = 010; 011; 011; 100$, after the first three sets of $W_z$ and $+M$ gates we have $|s\rangle\,|0\rangle\,|0\rangle\,|1\rangle\,|2\rangle\,|0\rangle$. Only the last ancillary qubit still needs to be changed into its corresponding $x$ coordinate. The $W_x$ gate looks at the last triple of bits in the bitstring, '100'. As this corresponds to a change in $x$ of -1, the gate performs $|s\rangle\,|0\rangle\,|0\rangle\,|1\rangle\,|2\rangle\,|0\rangle \rightarrow |s\rangle\,|0\rangle\,|0\rangle\,|1\rangle\,|2\rangle\,|-1\rangle$. The $+M$ modular adder gate then changes the last qubit to be the sum of itself and the previous $x$ coordinate, thus adding the 'difference' to the previous $x$ coordinate to obtain the new $x$ coordinate $|s\rangle\,|0\rangle\,|0\rangle\,|1\rangle\,|2\rangle\,|-1\rangle \rightarrow |s\rangle\,|0\rangle\,|0\rangle\,|1\rangle\,|2\rangle\,|1\rangle$.

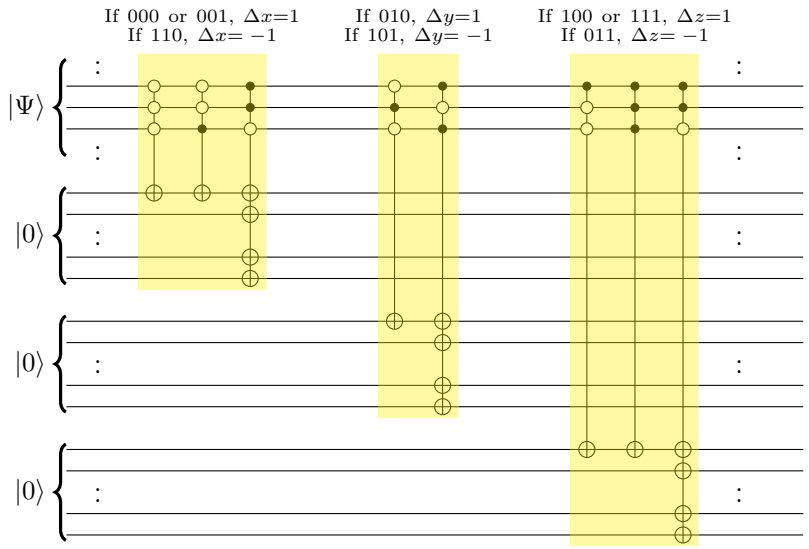
A similar procedure is done for the $y$ and $z$ registers with the $W_y$ and $W_z$ gates, and is performed over all $|s\rangle$ states simultaneously.

The following $W$ gates are used to decode triples of bits in each bitstring into coordinate changes in each direction. Circuits 9 and 10 are for decoding in the cubic and BCC models respectively.

For the cubic case, there are a total of 8 3-bit Toffoli gates, each checking for a unique bitstring from 000 to 111, thus for a particular bitstring $s$, only one of these gates performs its CNOT gates at a time. For the BCC case, there is a straightforward relationship between each bit and the difference in coordinates (0 means a change of +1 in the particular direction, 1 means a change of -1).

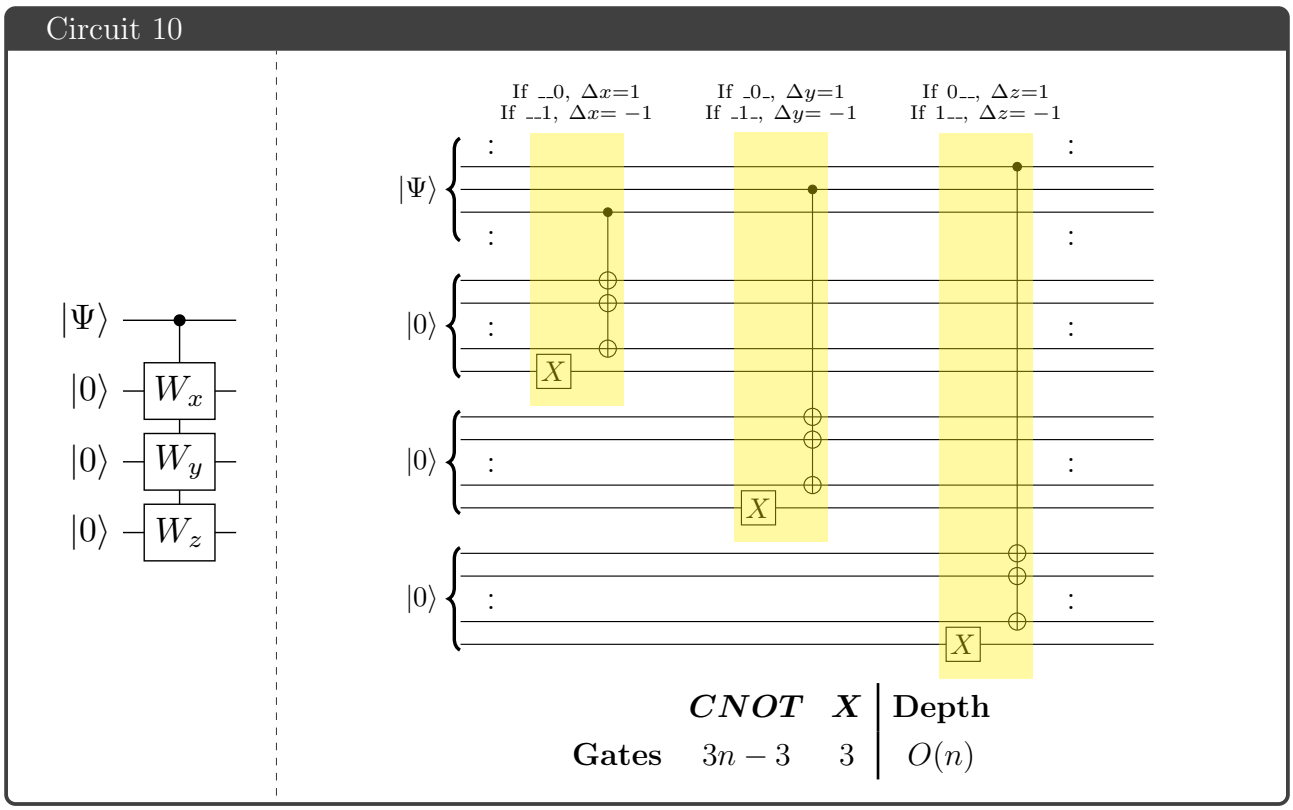
Circuit 9




| | 3-bit Toffoli | Depth |
|---|---|---|
| Gates | $3n + 5$ | $O(n^2)$ |

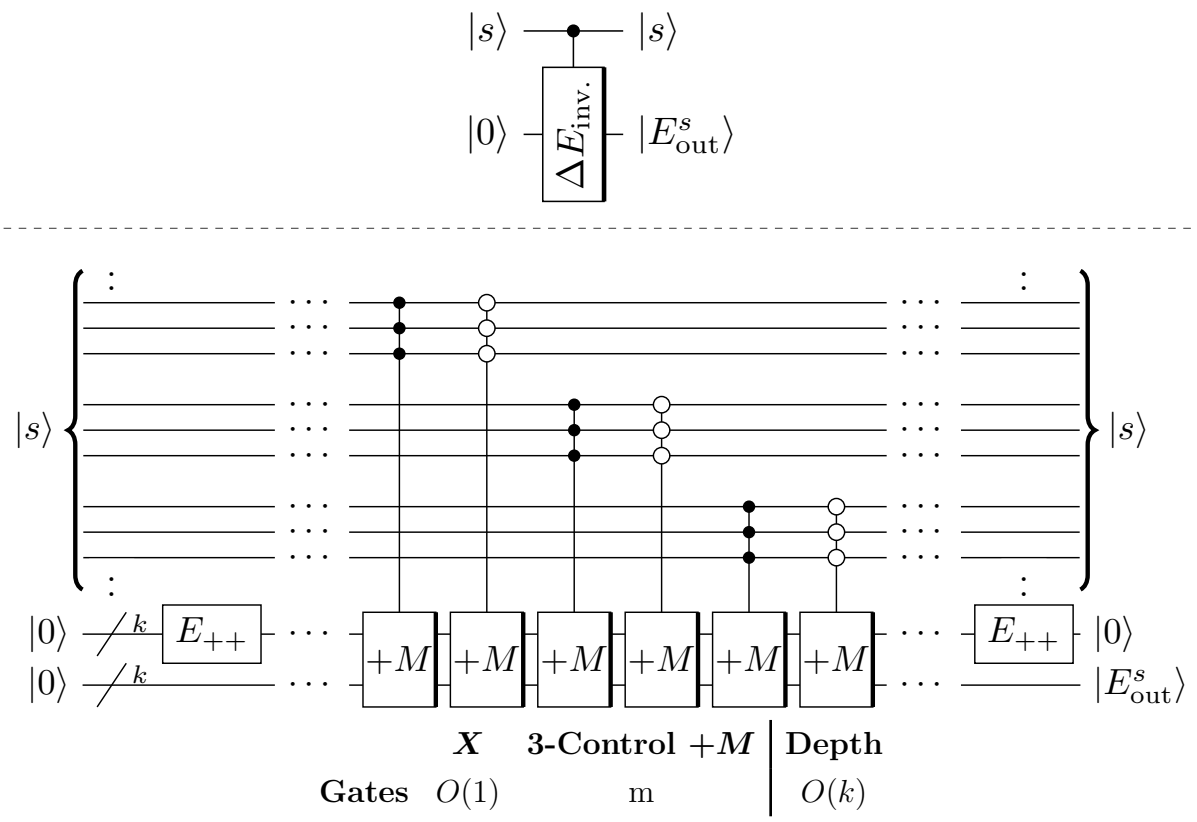
Circuit 9: $W$ gate for the 3D cubic model.




Circuit 10: $W$ gate for the 3D BCC model.

Returning to the other sub-circuits of $C$ (Circuit 7), we now look at the circuits for $\Delta E_{inv.}$ (Circuit 11) and $\Delta E_{i,j}$ (Circuits 12 and 13) gates.

Circuit 11: $\Delta E_{inv.}$ for the 3D cubic model

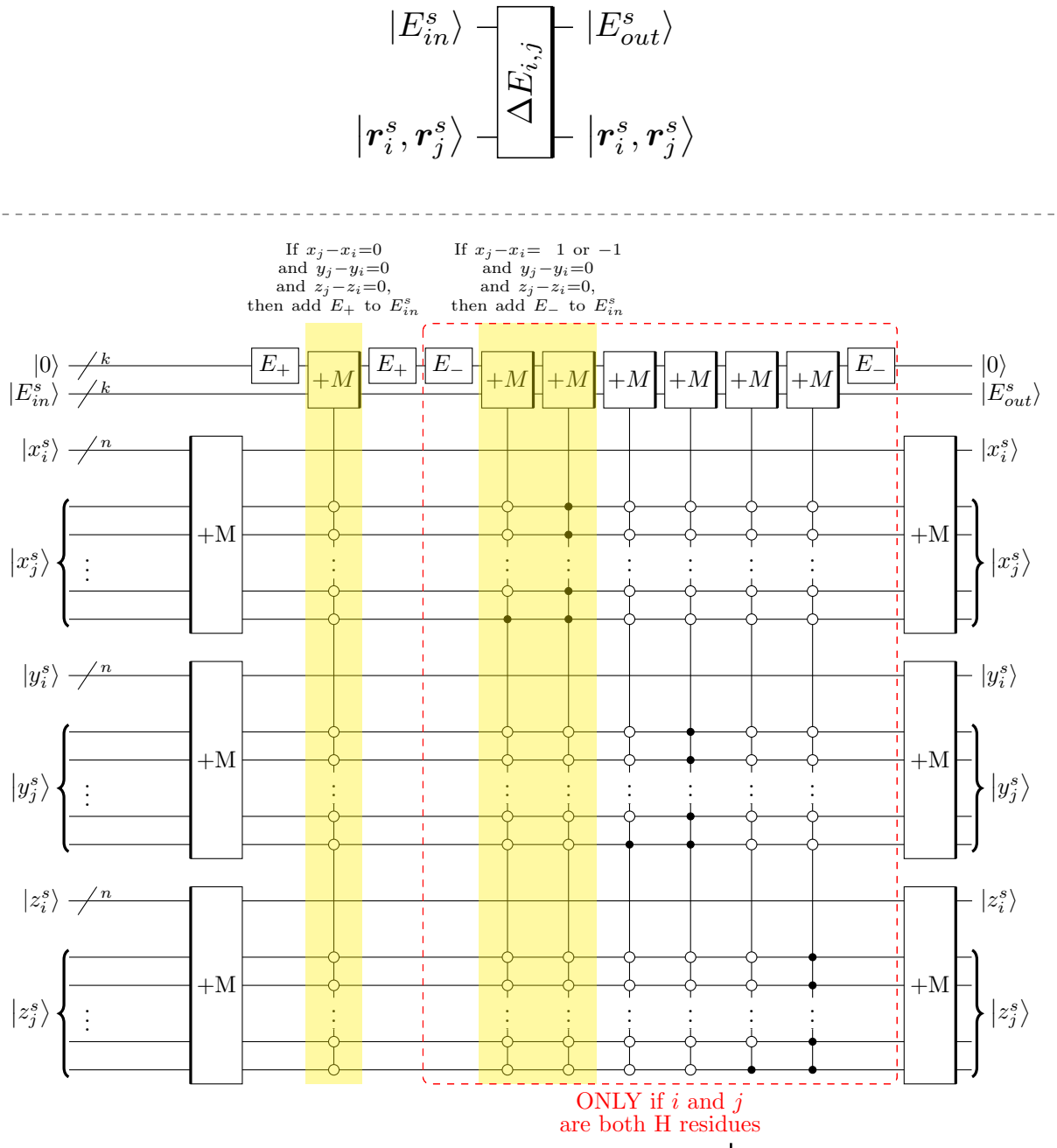



Circuit 11: $\Delta E_{inv}$ gate. This gate checks the bitstring $s$ for invalid bonds ('000' or '111'), and increments the energy register by an amount $E_{++}$ for each instance. $E_{++}$ performs the operation $E_{++}|0\rangle = |E_{++}\rangle$ and can be implemented with a constant number of gates, as per Circuit 12.

$\Delta E_{inv.}$ checks each bond (triple of bits) in each bitstring encoded, and increments the energy register by $E_{++}$ for any '000' or '111' bonds. Note that this gate is only required for the 3D cubic case when a penalty on these 'invalid' bitstrings are desired, and can be omitted from Circuit 7 otherwise.

For $\Delta E_{i,j}$, for the coordinates of residues $i$ and $j$ for a given bitstring $s$, we first use the reversed $+M$ gate to obtain the difference in coordinates. We can then check this difference to determine if the residues are intersecting, in which case the bitstring energy $|E^s_{in}\rangle$ is increased by $E^+$, or check if they are adjacent, in which case the negative quantity $E^-$ is added to the energy.

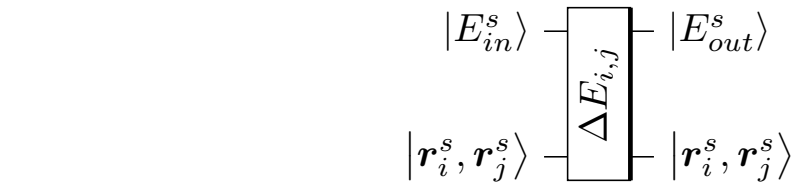
Circuit 12: $\Delta E_{i,j}$ for the 3D cubic model

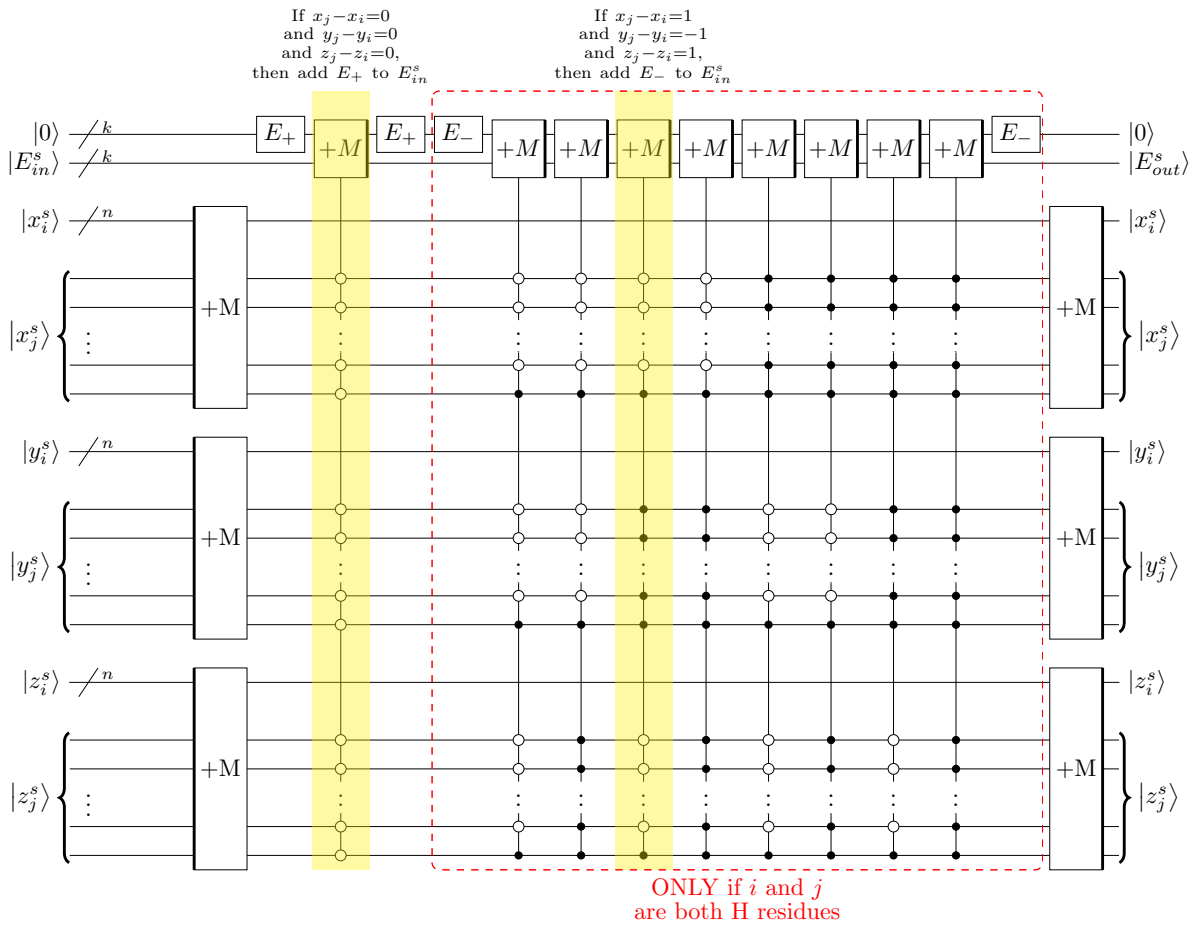



| | $+M$ | $3n$-Control $+M$ | $X$ | Depth |
|---|---|---|---|---|
| **Gates** | 6 | 7 | $O(1)$ | $O(n^2)$ |

Circuit 12: $\Delta E_{i,j}$ gate for the 3D cubic model. $x^s_i$ denotes the $i$th coordinate stored in $|x\rangle$ from Circuit 8 corresponding to a bitstring $s$, and similarly for $y^s_i$ and $z^s_i$. Gates in the red dashed line are only included if residues $i$ and $j$ are both of type H. Gates $E_+$ and $E_-$ are composed of parallel NOT gates with depth 1 and perform $E_+ |0\rangle = |E_+\rangle$ and $E_- |0\rangle = |E_-\rangle$ respectively. A maximum of $\lceil \log_2 E_\pm \rceil$ gates are required for each, but we can assume the values of $E_\pm$ to remain constant even with more residues, so the number of NOT gates per $M_\pm$ is constant ($O(1)$). The modular adders have linear depth, and the controlled modular adders have quadratic depth due to the multiple controls, hence the circuit depth is $O(n^2)$.

Circuit 13: $\Delta E_{i,j}$ for the 3D BCC model




| | $+M$ | $3n$-Control | $+M$ | $X$ | Depth |
|---|---|---|---|---|---|
| **Gates** | 6 | 9 | | $O(1)$ | $O(n^2)$ |

Circuit 13: $\Delta E_{i,j}$ gate for the 3D BCC model. $x^s_i$ denotes the $i$th coordinate stored in $|x\rangle$ from Circuit 8 corresponding to a bitstring $s$, and similarly for $y^s_i$ and $z^s_i$. Gates in the red dashed line are only included if residues $i$ and $j$ are both of type H. A constant number of NOT ($X$) gates are required for the $E_+$ and $E_-$ gates as per Circuit 12. The modular adders have linear depth, and the controlled modular adders have quadratic depth due to the multiple controls, hence the circuit depth is $O(n^2)$.

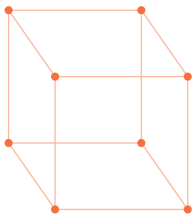
Example

Consider the bitstring $s = 010;011;101;010$, with residue coordinates $\mathbf{r}_0^s = (0,0,0)$, $\mathbf{r}_1^s = (0,1,0)$, $\mathbf{r}_2^s = (-1,1,0)$, $\mathbf{r}_3^s = (-1,0,0)$, $\mathbf{r}_4^s = (-1,1,0)$. After performing the gates $\Delta E_{0,2}$ to $\Delta E_{1,4}$ the energy register is now $|-1\rangle$, and we now perform $\Delta E_{2,4}$. We begin with the state $(|-1\rangle|-1\rangle)(|1\rangle|1\rangle)(|0\rangle|0\rangle)|0\rangle|-1\rangle$. Each set of parentheses denotes $x$, $y$ and $z$ directions respectively. The difference of each pair of $x$, $y$ and $z$ coordinates are taken, resulting in $(|-1\rangle|0\rangle)(|1\rangle|0\rangle)(|0\rangle|0\rangle)|0\rangle|-1\rangle$. The $E_+$ gate sets the second-last qubit to 5 (as we are assigning an energy of 5 to each self-intersection), and then checks if second, fourth and sixth registers are all $|0\rangle$. This is the case, so the 5 is added to the energy register. The second-last qubit is then reset back to $|0\rangle$ before $E_-$ is used to set it to $|-1\rangle$, and a similar check is carried out, but checking for adjacency. The second-last qubit is reset back to $|0\rangle$ and the initial $+M$ gates are performed forwards to reset the coordinate qubits to their initial state, resulting in the state $(|-1\rangle|-1\rangle)(|1\rangle|1\rangle)(|0\rangle|0\rangle)|0\rangle|4\rangle$.

This concludes our analysis of the sub-circuits of $U_{\mathrm{C}}(\gamma_i)$, and all that remains is the analysis of the mixing operator $U_{\mathrm{B}}(\beta_i)$. We consider two types of mixer: the hypercube adjacency matrix, and the complete graph adjacency matrix, which we will refer to as the hypercube and complete mixers respectively.

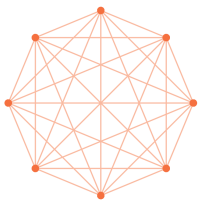
### 4.3.4 Unitary mixing operator $U_{\mathbf{B}}(\beta_i)$

To perform the mixing operation $U_{\mathrm{B}}(\beta_i)$, we have used a hypercube adjacency matrix and a complete adjacency matrix for $B$. For illustration, we show the graphs and corresponding adjacency matrices for the hypercube graph (15) and complete graph (16) below.



$$B = \begin{pmatrix} 0&1&1&0&1&0&0&0\\ 1&0&0&1&0&1&0&0\\ 1&0&0&1&0&0&1&0\\ 0&1&1&0&0&0&0&1\\ 1&0&0&0&0&1&1&0\\ 0&1&0&0&1&0&0&1\\ 0&0&1&0&1&0&0&1\\ 0&0&0&1&0&1&1&0 \end{pmatrix}$$

Figure 15: Hypercube graph and adjacency matrix

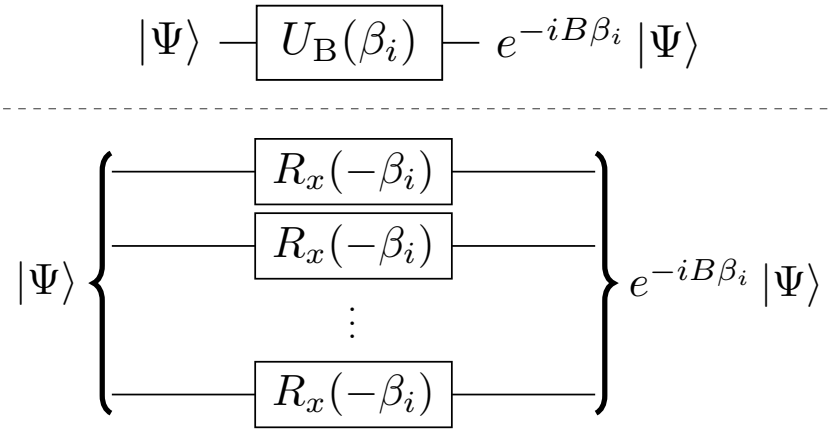


$$B = \begin{pmatrix} 0&1&1&1&1&1&1&1\\ 1&0&1&1&1&1&1&1\\ 1&1&0&1&1&1&1&1\\ 1&1&1&0&1&1&1&1\\ 1&1&1&1&0&1&1&1\\ 1&1&1&1&1&0&1&1\\ 1&1&1&1&1&1&0&1\\ 1&1&1&1&1&1&1&0 \end{pmatrix}$$

Figure 16: Complete graph and adjacency matrix

Circuit 14 shows the hypercube mixer.

Circuit 14: $[U_{\mathrm{B}}(\beta_i)]$ for $B$ a hypercube adjacency matrix

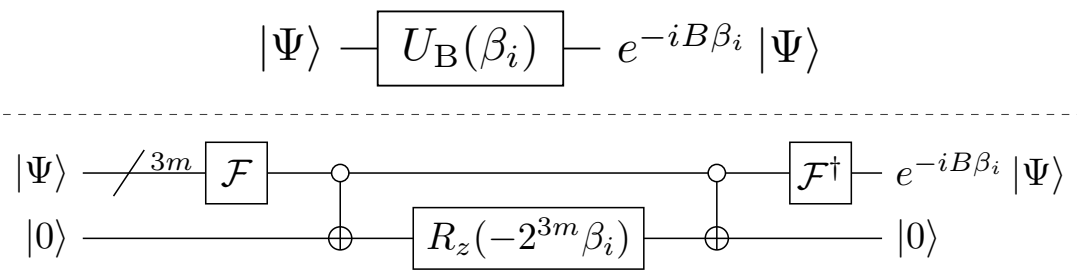



| | $R_x$ | Depth |
|---|---|---|
| **Gates** | $3m$ | 1 |

Circuit 14: Unitary mixing operator with a hypercube mixer, parameterised by $\beta \in [0, \pi)$. This gate is applied alternately with the unitary phase shift operator to perform the QAOA evolution.

Circuit 15 shows the complete mixer. The circuit is based on the mixing circuit proposed by [8].

Circuit 15: $[U_{\mathrm{B}}(\beta_i)]$ for $B$ a complete graph adjacency matrix




| | $3m$-bit Toffoli | $3m$-bit Fourier | $R_z$ | Depth |
|---|---|---|---|---|
| **Gates** | 2 | 2 | 1 | $O(m)$ |

Circuit 15: Unitary mixing operator with a complete mixer, parameterised by $\beta \in [0, \pi)$, based on the mixing circuit proposed by [8]. This gate is applied alternately with the unitary phase shift operator to perform the QAOA evolution. The CNOT gates are controlled on all $3m$ qubits being $|0\rangle$. $\mathcal{F}$ denotes the Quantum Fourier Transform, and has linear depth, hence the depth of $U_{\mathrm{B}}(\beta_i)$ is $O(m)$.

# 5 Simulating the Quantum Algorithm

Although practical working quantum computers are not available yet, it is still possible to verify that the proposed algorithm works in theory by simulating the algorithm on a classical computer.

Recall that the QAOA creates the state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ for some $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_{p-1})$ and $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, ..., \gamma_{p=1})$. The operator $C$ corresponding to the objective function can then be applied and the expectation value $\langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \, C \, |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ found by averaging over multiple measurements.

Classically, we can compute $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ to find the probabilities of measuring each state (contained in the modulus squared of components of $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$), and multiply by the energy corresponding to each state (conformation) and sum the result to obtain the theoretical expectation value. We can perform optimisation over the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters to obtain values of $\boldsymbol{\beta}, \boldsymbol{\gamma}$ that result in a local minimum value of $\langle C \rangle = \langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \, C \, |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$, and we can verify that measuring this state results in a 'good probability' of observing the minimal energy state.

In this section, we aim to:

- Simulate the algorithm implemented by the proposed circuit
- Verify the effectiveness of the QAOA in optimising our objective function $C$.
- Verify that classical optimisation can efficiently find 'good' minima and thus be used during the implementation of this circuit on a quantum computer

The simulation of the quantum algorithm was written in Python 3.7.3 [100], with the assistance of the following packages and libraries: NumPy [101, 102], Multiprocessing [4, 5], MPI for Python [103, 104, 105] and Pandas [106]. All plots in this were produced using Plotly [60].

The simulations were performed on the supercomputer Magnus in the Pawsey Supercomputing Centre, comprising of a 1488 nodes and 24 cores per node for a total of over 35 000 processor cores and over 1 petaFLOP of computing power [107]. A total of 80 000 core-hours were used for this project, with the largest single computation requiring almost 24 hours of compute time over 6 nodes (approximately 3000 core-hours).

The rest of this section details the implementation and results of the simulations used for this project.

## 5.1 Overview of the Classical Simulation

In this simulation, we are aiming to minimise the expectation value $\langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \, C \, |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ with respect to parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, thus increasing amplitude of the minimal energy state in the state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$.

It is important to note that we are only simulating the algorithm performed by the proposed circuit. Running the circuit on a quantum computer will result in the actual quantum state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$, but we instead will be representing these quantum states as vectors, and quantum gates as matrices. Thus to avoid confusion between kets and the vectors used as computational tools in the classical simulation, we will not use bras and kets $\langle \quad \rangle$ to denote the latter, but instead use standard vector conventions such as boldface ($\boldsymbol{C}$) and arrows ($\overrightarrow{\boldsymbol{\beta}, \boldsymbol{\gamma}}$). A table of corresponding kets and computation vectors are shown in Table 3.

| Quantum | Simulation | Quantum | Simulation |
|---|---|---|---|
| $\|\Psi\rangle$ | $\boldsymbol{\Psi}$ | $\|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ | $\overrightarrow{\boldsymbol{\beta}, \boldsymbol{\gamma}}$ |
| $C$ (diagonal matrix) | $\boldsymbol{C}$ (vector) | $\langle C \rangle$ | $\bar{C}$ |

Table 3: Corresponding quantum and classical simulation notation.

Firstly we generate a vector $\boldsymbol{\Psi}$ as an equal superposition of all considered states/bitstrings. Then according to the lattice model used, each bitstring is mapped to the set of coordinates of the conformation it encodes, and the vector of the energies of these conformations are stored in a vector $\boldsymbol{C}$, corresponding to the diagonal matrix $C$ of conformation energies. At this point, classically we can determine the minimal energy state by looking to find the lowest value stored in $\boldsymbol{C}$, however our goal in this simulation is to simulate the QAOA.

We consider the following three models, as well as the use of hypercube and complete mixers. For convenience, we will refer to these are models 1 to 3, and they are defined as follows:

- **Model 1:** 3D cubic lattice, conformations have -1 energy per H-H bond, +5 energy per self-intersection and +10 energy per invalid bond '000' and '111'.
- **Model 2:** 3D cubic lattice, conformations have -1 energy per H-H bond, +5 energy per self-intersection and no energy penalty for '000' and '111' bonds (we are allowing 'duplicates').
- **Model 3:** 3D body-centred cubic (BCC) lattice, conformations have -1 energy per H-H bond, +5 energy per self-intersection, and no bitstrings are invalid.

### 5.1.1 Obtaining the energy vector

In the quantum algorithm, conformation energies are stored as entries in a diagonal matrix/Hamiltonian $C$, but this can be more easily stored in computer memory as a vector. We denote this vector $\boldsymbol{C}$, of which each element corresponds to a considered bitstring. For example in the 3D cubic model, the 0th element of $\boldsymbol{C}$ corresponds to conformation bitstring 010 010 ... 000, the 1st element to 010 010 ... 001, and so on until the last element corresponds to 010 011 ... 111. For convenience we denote the number of components in $\boldsymbol{C}$ as $M$.

The first part of the simulation deals with obtaining the 'energy vector' $\boldsymbol{C}$ - that is, the vector containing the energies corresponding to each simulated conformation. The process for doing this is described in detail in Section 4.2.2.

### 5.1.2 Performing the QAOA

The aim of this next section is to obtain the value of $\bar{C}$ (corresponding to $\langle C \rangle = \langle \boldsymbol{\beta}, \boldsymbol{\gamma} | \, C \, | \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle$) for some given values of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ and some number of iterations $p$ indexed by $j \in \{0, 1, \ldots, p-1\}$. This can be computed as follows:

1. Begin with a uniform vector of size $M$, representing an equal superposition of all states:
$$\boldsymbol{\Psi} = \frac{1}{\sqrt{2^M}}(1, 1, \ldots, 1) \tag{23}$$
As with $\boldsymbol{C}$, each component of $\boldsymbol{\Psi}$ corresponds to a bitstring.

2. Operate on $\boldsymbol{\Psi}$ with the unitary operator $e^{iC\gamma_j}$ to obtain a new $\boldsymbol{\Psi}$
$$e^{iC\gamma_j}\boldsymbol{\Psi} = \begin{pmatrix} e^{i\boldsymbol{C}_0\gamma_j} & 0 & \cdots & 0 \\ 0 & e^{i\boldsymbol{C}_1\gamma_j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{i\boldsymbol{C}_{2^M-1}\gamma_j} \end{pmatrix} \boldsymbol{\Psi} \tag{24}$$
$$= \begin{pmatrix} e^{i\boldsymbol{C}_0\gamma_j} \\ e^{i\boldsymbol{C}_1\gamma_j} \\ \vdots \\ e^{i\boldsymbol{C}_{2^M-1}\gamma_j} \end{pmatrix} \cdot \boldsymbol{\Psi} \tag{25}$$
$$\longrightarrow \boldsymbol{\Psi} \tag{26}$$
Note that as $C$ is diagonal, the result of the matrix exponentiation is simply the exponential of each diagonal element.

3. Operate on $\boldsymbol{\Psi}$ with the mixing unitary $e^{iB\beta_j}$ to obtain a new $\boldsymbol{\Psi}$.
$$e^{iB\beta_j}\boldsymbol{\Psi} \longrightarrow \boldsymbol{\Psi} \tag{27}$$
$B$ is not diagonal, so obtaining a simplified form for the matrix exponential is not easy, and this part is done through direct computation of the matrix exponential, or more specifically, directly computing the effect of on each element of $\boldsymbol{\Psi}$.

4. Repeat steps 2. and 3. for each $j$ from 0 to $p-1$. We denote the result $\overrightarrow{\boldsymbol{\beta}, \boldsymbol{\gamma}}$, analogous to the state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$.

5. Compute the expectation value of the objective function $C$ using
$$\bar{C} = \left( (\overrightarrow{\boldsymbol{\beta}, \boldsymbol{\gamma}})^\dagger \odot (\overrightarrow{\boldsymbol{\beta}, \boldsymbol{\gamma}}) \right) \cdot \boldsymbol{C} \tag{28}$$
analogously to $\langle C \rangle = \langle \boldsymbol{\beta}, \boldsymbol{\gamma} | \, C \, | \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle$, where $\odot$ denotes element-wise multiplication of vectors.

### 5.1.3 Optimisation and post-processing

At this point, we have obtained a value of $\bar{C}$ for some choice of parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. We can considered $\bar{C}$ as a function of the $\boldsymbol{\beta}, \boldsymbol{\gamma}$ parameters, and so we can optimise $\bar{C}$ with respect to $\boldsymbol{\beta} \in [0, \pi)^{\otimes p}$ and $\boldsymbol{\gamma} \in [0, 2\pi)^{\otimes p}$. The resulting values of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ should yield a 'good' probability of measuring a minimal energy state.

We present the results of two simulations, both run with a number of models. The first simulation (Section 5.2) used $p = 2$ and performed a brute-force search on a grid over the parameter space, before refining the best minimum using the Nelder-Mead algorithm to obtain parameters for the global minimum of $\bar{C}$. The second simulation (Section 5.3) used $p = 1, 2, 4$ and 8 and only used the Nelder-Mead algorithm to find minima, but was run many times for randomly chosen initial parameters to obtain information on the distribution of local minima of $\bar{C}$ in the parameter space.

## 5.2 Parameter Space Results and Visualisation

We consider a HHHHH chain (5 residues, all hydrophobic) in the HP model with QAOA depth $p = 2$ (with 4 parameters, $\boldsymbol{\beta} = (\beta_0, \beta_1)$ and $\boldsymbol{\gamma} = (\gamma_0, \gamma_1)$), using models 1 to 3 (as defined below) and both hypercube and complete mixers. For each case, we divided the parameter space $\boldsymbol{\beta} \otimes \boldsymbol{\gamma} \in [0, \pi)^{\otimes p} \otimes [0, 2\pi)^{\otimes p}$ into a $h \times h \times 2h \times 2h$ grid and evaluated $\bar{C}$ at each point. For the Model 3 with a hypercube matrix, $h = 50$ was used, for a total of $2.5 \times 10^7$ evaluations of $\bar{C}$, and for all other cases $h = 60$ was used, for a total of $5.184 \times 10^7$ evaluations of $\bar{C}$.

For each plot in each of Figures 17 to 22, we consider the parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters at the global minimum of $\bar{C}$ and vary two parameters at a time whilst the other two parameters remain constant.

The optimal parameter values in each case are provided in Table 4 for comparison.

| Model no. | Mixer | $\bar{C}$ | Min. E prob. | $R_{99}$ | $\gamma_0$ | $\beta_0$ | $\gamma_1$ | $\beta_1$ |
|---|---|---|---|---|---|---|---|---|
| 1 | Hypercube | 0.1418 | 0.2769 | 15 | 6.1468 | 0.1919 | 0.3206 | 3.0118 |
| 1 | Complete | 0.5331 | 0.2163 | 19 | 0.1047 | 2.2515 | 6.0214 | 0.1047 |
| 2 | Hypercube | -0.2451 | 0.3067 | 13 | 6.0979 | 0.2173 | 5.9093 | 0.1189 |
| 2 | Complete | -0.3149 | 0.3697 | 10 | 0.5236 | 1.7802 | 6.0214 | 0.4189 |
| 3 | Hypercube | -0.6620 | 0.2046 | 21 | 1.5642 | 2.6172 | 6.0655 | 0.2596 |
| 3 | Complete | -0.8763 | 0.3529 | 11 | 1.6232 | 2.5656 | 1.7279 | 0.6807 |

Table 4: Global minima parameters for the HHHHH chain in models 1 to 3. $\bar{C}$ is the minimal value of the energy expectation, and 'Min. E prob.' is the probability of measuring a minimal energy conformation from state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$ with the given parameters. $R_{99}$ is the number of measurements required to have a 99% probability of measuring the minimal energy state at least once.

The $R_{99}$ metric describes the number of measurements of the state to obtain a 99% chance of measuring an optimal state, given an individual success probability $p$. We can derive an expression for $R_q$, as the probability of no success after $R_q$ measurements is $1 - q = (1 - p)^{R_q}$, from which $R_q = \frac{\ln(1-q)}{\ln(1-p)}$.

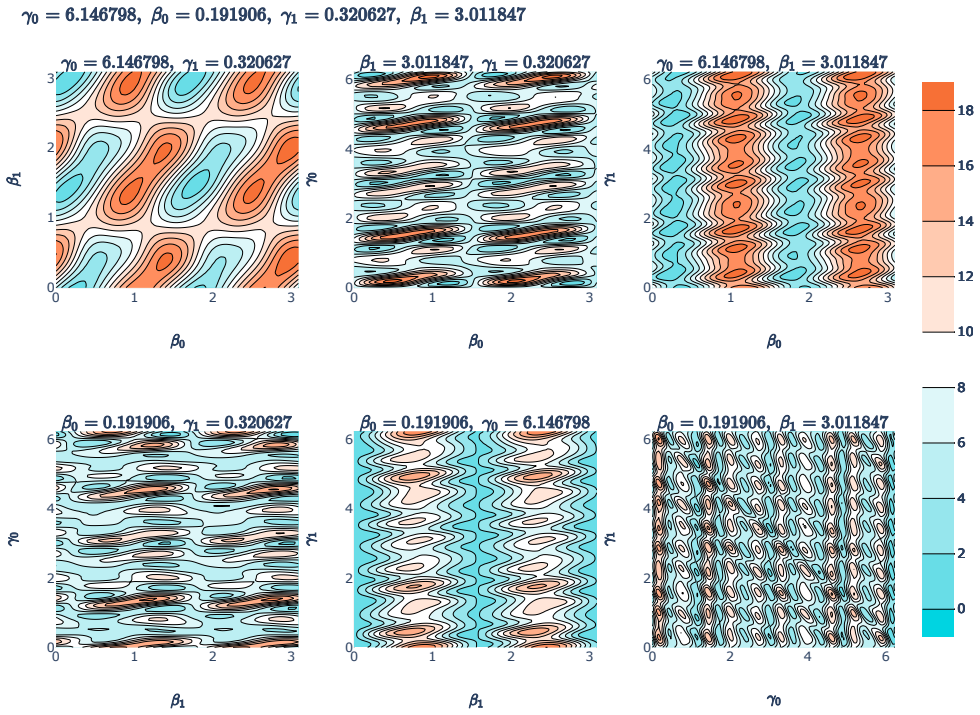



Figure 17: Model 1 with hypercube mixer; energy expectation parameter space. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,10). The $\beta_0, \beta_1, \gamma_0 and \gamma_1$

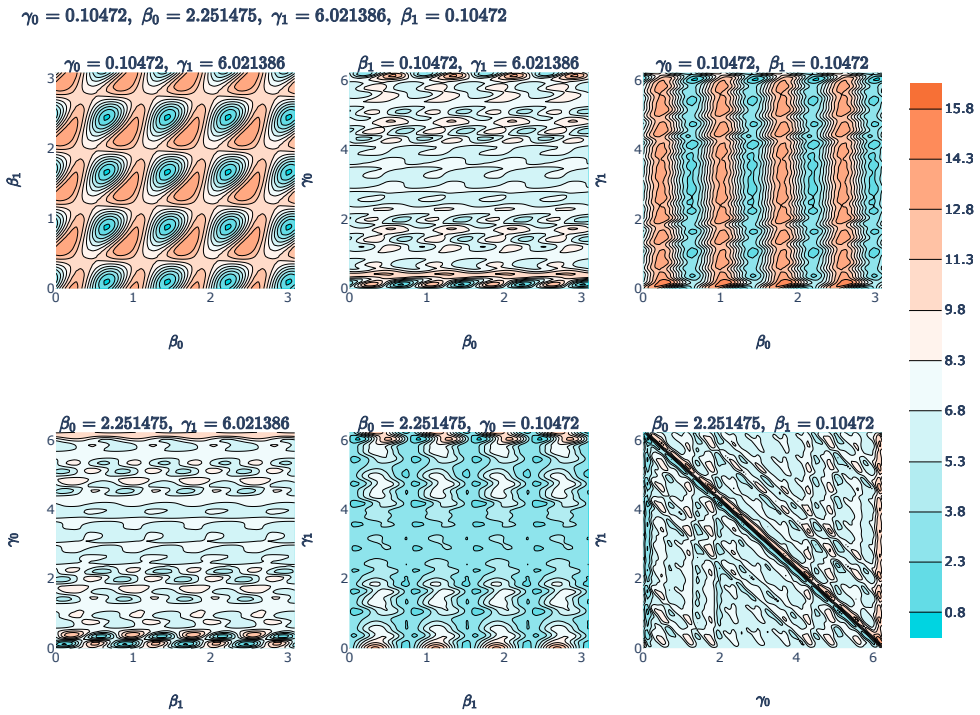



Figure 18: Model 1 with complete mixer; energy expectation parameter space. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,10).

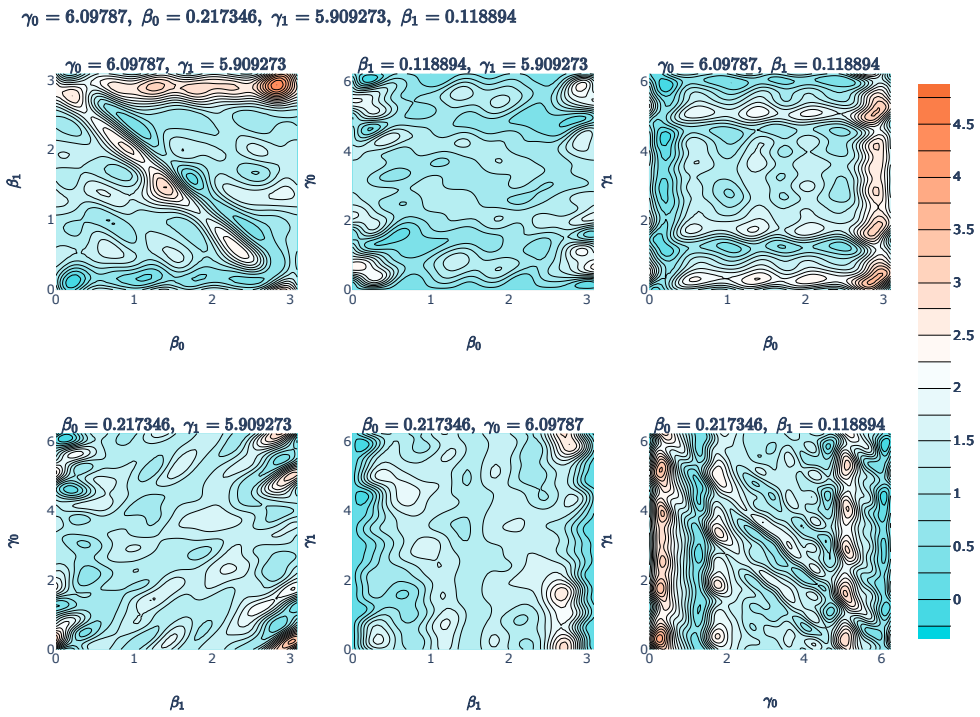



Figure 19: Model 2 with hypercube mixer; energy expectation parameter space. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,0).

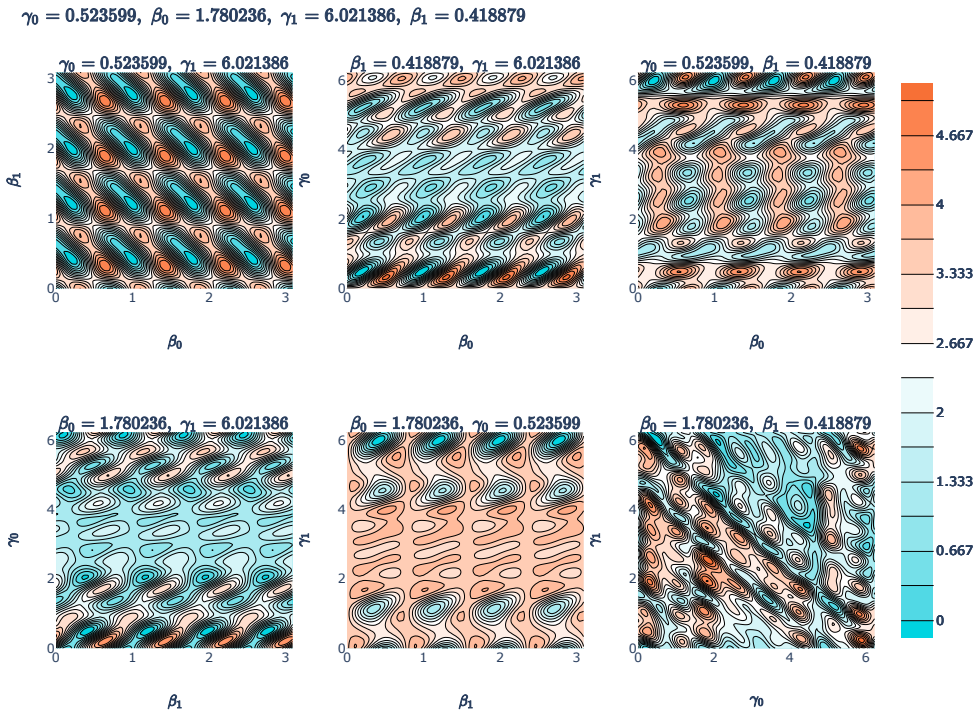



Figure 20: Model 2 with complete mixer; energy expectation parameter space. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,0).

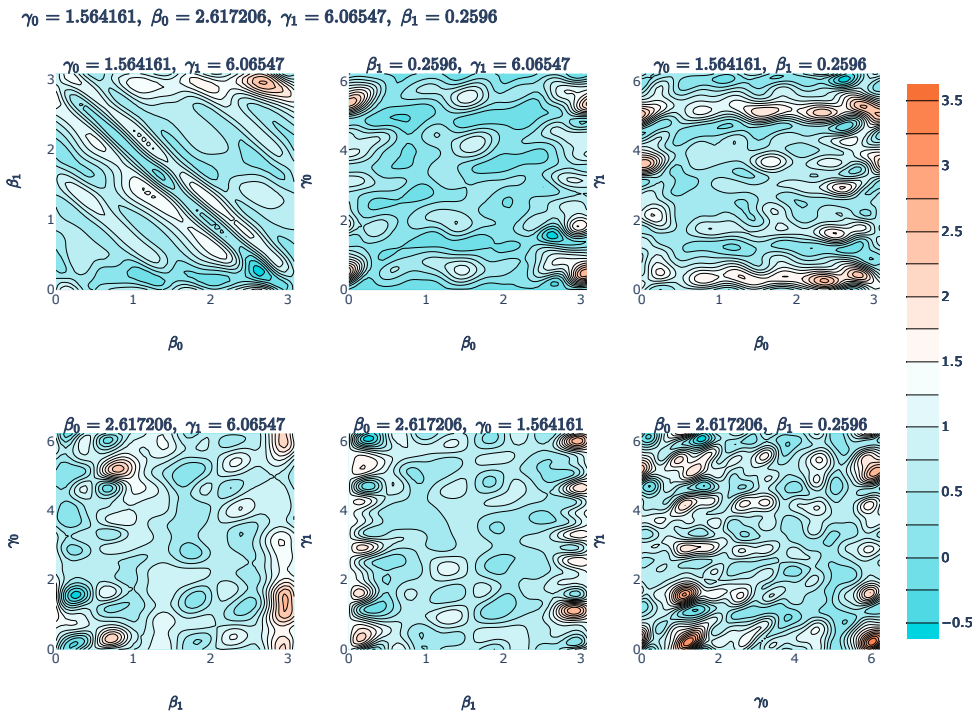



Figure 21: Model 3 with hypercube mixer; energy expectation parameter space. 3D BCC lattice, HHHHH chain, energy parameters (-1,5).

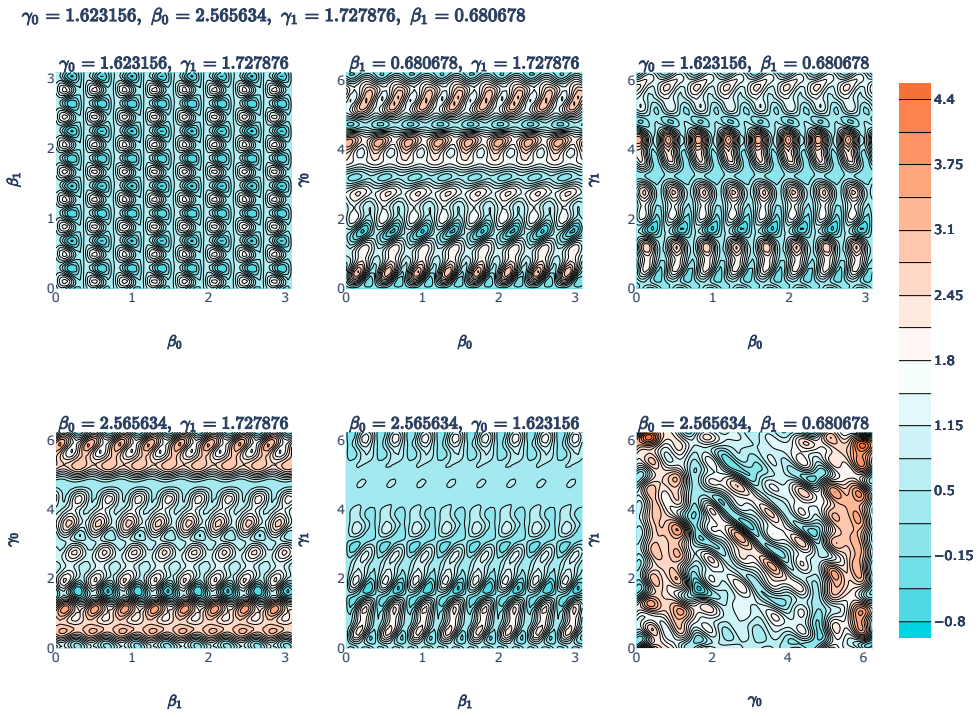



Figure 22: Model 3 with complete mixer; energy expectation parameter space. 3D BCC lattice, HHHHH chain, energy parameters (-1,5).

Some key observations and predictions from these results are:

- The parameter landscape has rough periodicity with many local minima, meaning that locating the global maximum will likely be difficult for higher values of $p$ (more QAOA iterations).
- Despite the difficulty in locating the global minima, many local minima have values close to that of the global minimum, so it may suffice to locate one of these minima instead. This is reminiscent of Machine Learning with Neural Networks, where some loss function ($\bar{C}$ in this case) is minimised with respect to the network weights (the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ in this case), and due to the very high dimensionality of the problem, it is extremely difficult to find the global minimum, and local minima are often sufficiently good.
- The choice of mixer has a significant impact on the landscape of $\bar{C}$, for example between Figures 21 and 22, where the latter is highly periodic with low frequency in $\beta_0$ and $\beta_1$. As a result, there is likely to be no correlation between 'good' values between different mixers.
- The hypercube mixer performed better in Model 1 by a wide margin in terms of $\bar{C}$ (and therefore also probability of measuring a minimal energy conformation), whereas the complete mixer performed better in Models 2 and 3 with smaller margins. It is difficult to say whether one mixer is better than the other for the cases tested here.

## 5.3 QAOA Depth

The previous section dealt with the case of four QAOA parameters ($p = 2$), optimising classically using brute force and the Nelder-Mead algorithm. Whilst this was good for visualising $\bar{C}$ over the parameter space, there are two issues with this approach.

1. Brute force optimisation is very slow, and so would likely not be used to optimise $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ when computing on an actual quantum computer.
2. We have only yet considered $p = 2$. One of the strengths of the QAOA is that as $p$ increases, the minimum value of $\bar{C}$ monotonically decreases [92], with the hope that the probability amplitude of the native state increases with this.

As such, we will now consider the cases of $p = 1, 2, 4, 8$, with the use of the Nelder-Mead algorithm to determine local minima.

The procedure used for calculating $\bar{C}$ is the same as for the brute-force case, but instead of calculating $\bar{C}$ at all points on a grid, we pick random initial values for our $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ parameters and minimise locally as follows:

1. Select a random point in the parameter space $[0, \pi) \times [0, 2\pi)$

2. Build a simplex about this point by using the initial random point as a vertex, and adding 0.005 to each parameter individually to obtain $2p$ other coordinates for vertices. 0.005 was chosen as initial experiments indicated this was small enough to not 'jump' between valleys, but also large enough to propagate quickly.

3. Use the Nelder-Mead algorithm to propagate the simplex to locally minimise $\bar{C}$. Terminate the propagation when 3000 steps have been taken or until the value of $\bar{C}$ at each vertex are within $10^{-8}$ of each other. Keep the lowest value of $\bar{C}$ and its parameters. 3000 steps and a tolerance of $10^{-8}$ were chosen as initial experiments indicated these parameters allowed nearly all runs to terminate near a minimum without compromising too much performance.

4. Record the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$, the value of $\bar{C}$ and the probability of measuring a minimal energy state

5. Repeat steps 1-4 for more randomly chosen starting points

This procedure was performed for QAOA depths $p = 1, 2, 4, 8$ with models 1, 2 and 3, with both hypercube and complete mixers. For each depth, model and mixer a total of 5760 runs were taken.

Figures 23, 25, 27, 29, 31 and 33 show the relative frequency of finding local minima of a particular energy expectation value, for each number of QAOA iterations.

Figures 24, 26, 28, 30, 32 and 34 show the energy expectation of each run versus the probability of measuring an optimal solution for that run, for each number of QAOA iterations.

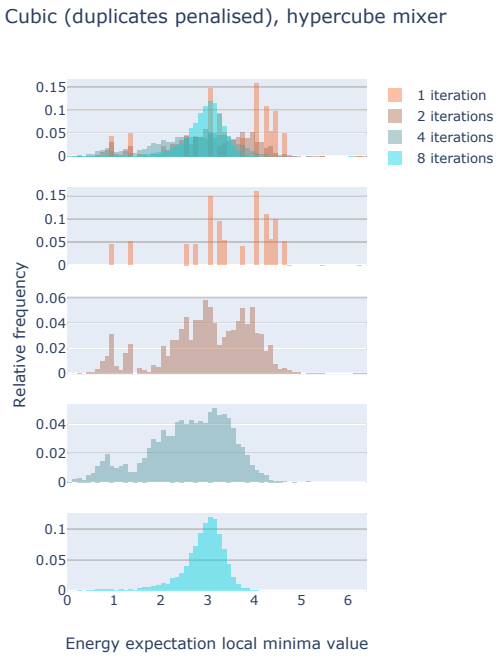



Figure 23: Model 1 with hypercube mixer; frequencies of local minima. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,10).

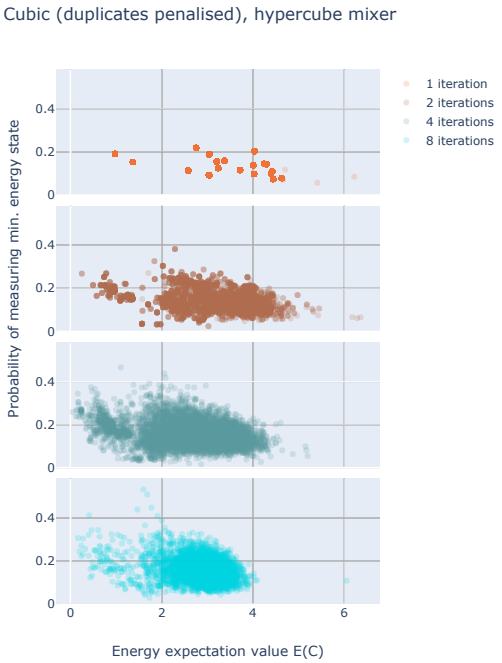



Figure 24: Model 1 with hypercube mixer; local minima energy expectations versus probability of measuring a minimal energy state.

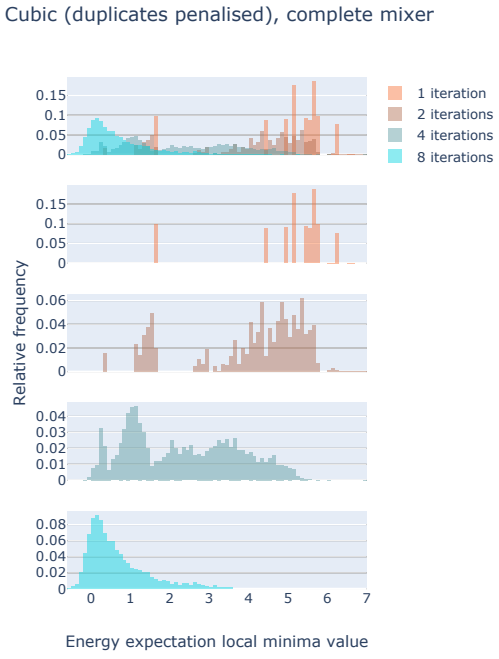



Figure 25: Model 1 with complete mixer; frequencies of local minima. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,10).

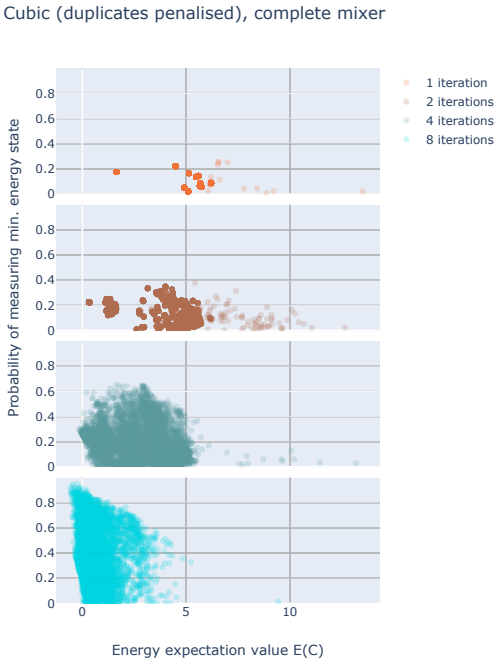



Figure 26: Model 1 with complete mixer; local minima energy expectations versus probability of measuring a minimal energy state.

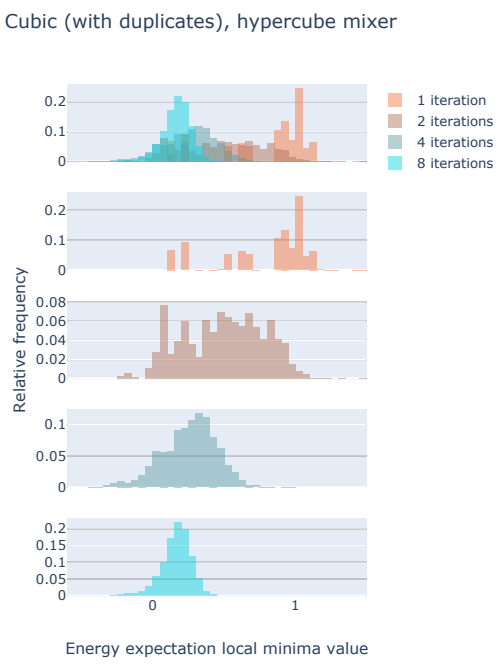



Figure 27: Model 2 with hypercube mixer; frequencies of local minima. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,0).

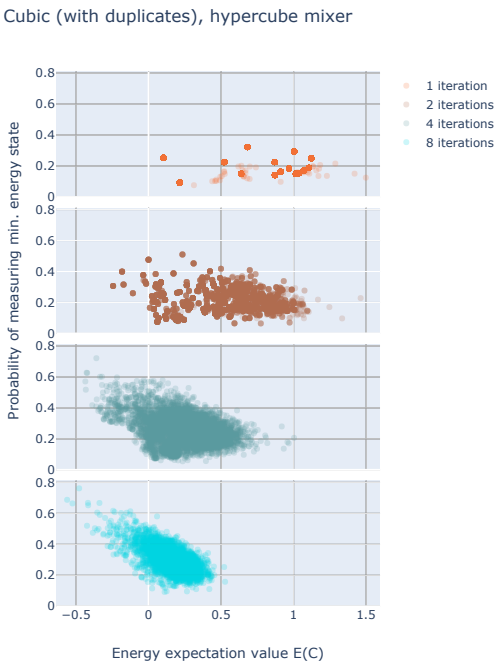



Figure 28: Model 2 with hypercube mixer; local minima energy expectations versus probability of measuring a minimal energy state.

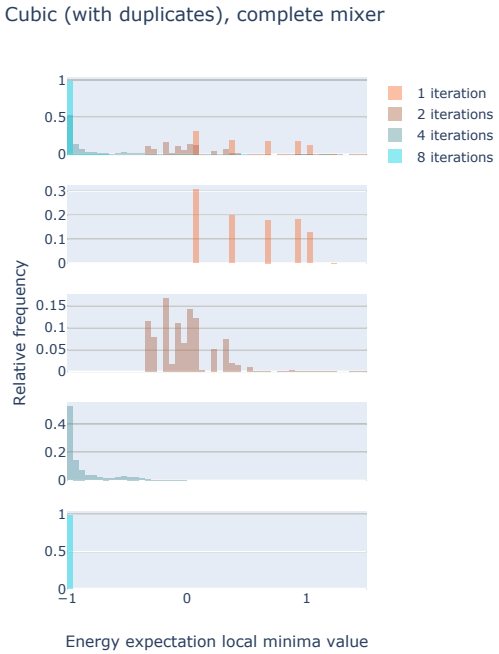



Figure 29: Model 2 with complete mixer; frequencies of local minima. 3D cubic lattice, HHHHH chain, energy parameters (-1,5,0).

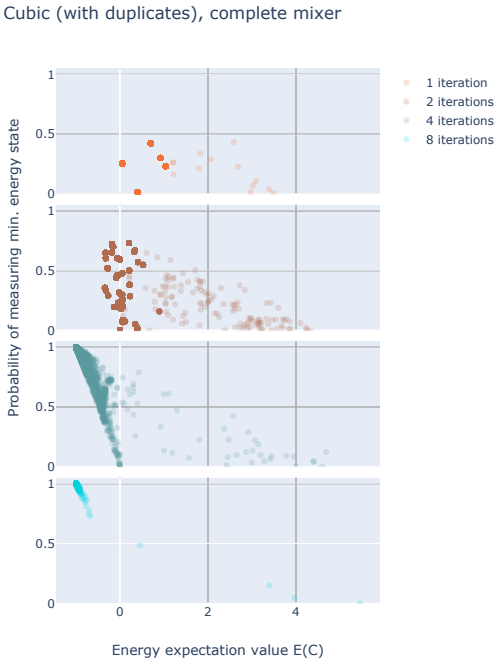



Figure 30: Model 2 with complete mixer; local minima energy expectations versus probability of measuring a minimal energy state.

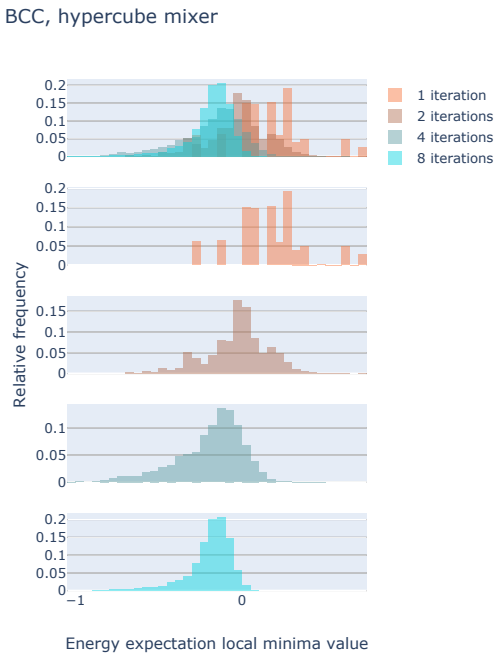



Figure 31: Model 3 with hypercube mixer; frequencies of local minima. 3D BCC lattice, HHHHH chain, energy parameters (-1,5).

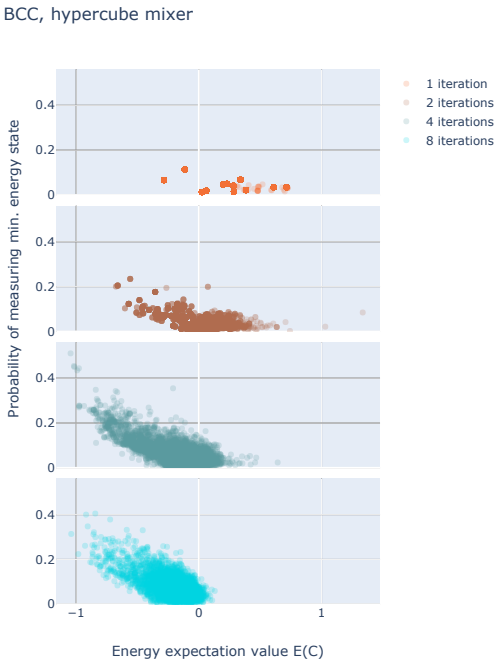



Figure 32: Model 3 with hypercube mixer; local minima energy expectations versus probability of measuring a minimal energy state.

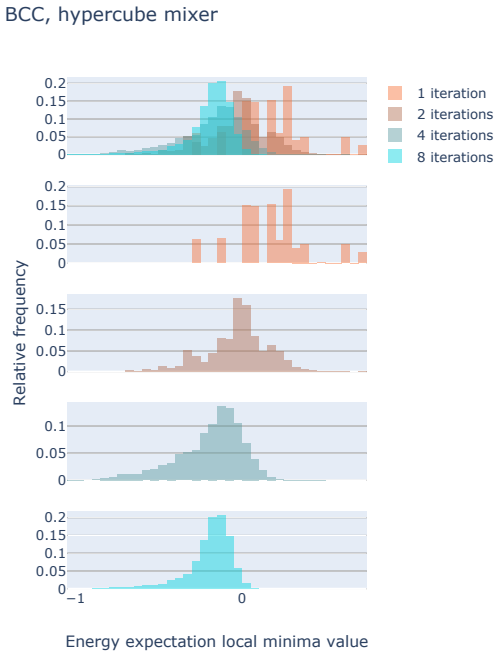



Figure 33: Model 3 with hypercube mixer; frequencies of local minima. 3D BCC lattice, HHHHH chain, energy parameters (-1,5).

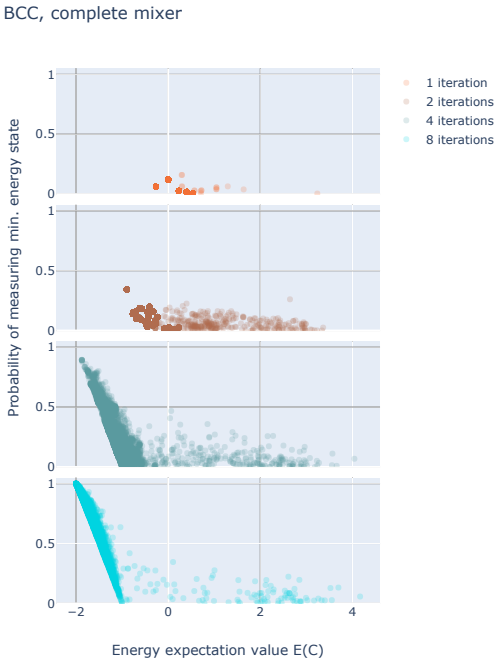



Figure 34: Model 3 with complete mixer; local minima energy expectations versus probability of measuring a minimal energy state.

Analysing these results, we note the following:

- The distribution of minima found improves (approaches $-1$ for the cubic cases and $-2$ for the BCC cases) with more QAOA iterations for nearly all cases. This is expected as the parameter space searched with $p$ iterations, $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{p-1}) \in [0, \pi)^{\otimes p}, \boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{p-1}) \in [0, 2\pi)^{\otimes p}$ is a subset of the space searched with $p + 1$ iterations. Specifically, it is the parameter space for the $p + 1$ case with $\beta_p = \gamma_p = 0$. Thus, we should expect that by only ever expanding the search space with increasing $p$, the optimal value of $\bar{C}$ (and thus $\langle C \rangle$ when performing the computation on an actual quantum computer) should only lower, or stay the same at worst.

- The complete mixer outperforms the hypercube mixer in minimising $\bar{C}$ in every case. This is also be to expected, as the connectivity of states in the hypercube mixer is much lower. Thinking of quantum walks on a hypercube, it takes $n$ steps for the some of the amplitude of the $|0\rangle$ state to reach the $|2^n - 1\rangle$ state. On the other hand, on a complete graph every state is connected to every other state in 1 step. Thus we can expect that computational basis amplitudes 'migrate' towards low-energy states more quickly with the complete mixer.

- Allowing duplicate bonds (instead of using an invalidity penalty) improves the expectation value. This is also somewhat expected; with an invalidity penalty more of the energy landscape will have a high $\bar{C}$, and $\bar{C}$ can also be easily lowered by decreasing the amplitude of high-energy states. Without such high values in the energy landscape, $\bar{C}$ is more likely to be reduced by increasing the amplitude of our desired minimal-energy states.

- A complete mixer with the cubic lattice with duplicated directions has a remarkably high probability of reaching close-to-optimal energies. For $p = 4$, over 50% of runs obtained an energy expectation of between -1 and -0.95, and for $p = 8$, over 80% of runs obtained an energy expectation of between -1 and -0.999. For $p = 8$, 99.7% of the minima found had a probability of measuring an optimal state of 0.99, thus having $R_{99} = 1$, and 88% of the minima found had a greater than $1 - 10^7$ chance of obtaining an optimal conformations on a single measurement. These extreme probabilities may be due to the allowance of duplicate bonds, so much more of the search space of bitstrings correspond to an optimal solution.

- Probability of a minimal energy measurement increases gradually with decreasing $\bar{C}$, but rises very quickly once $\bar{C}$ is below the second-lowest possible energy (0 in the cubic cases, -1 in the BCC cases). This is also expected, as below these values of $\bar{C}$, the only ways to decreases $\bar{C}$ further are to amplify the minimum energy solutions.

Whilst these results are promising, there are many avenues left to explore in improving both the algorithm and the results, such as:

- Performing a small case of the quantum circuit on an actual quantum computer to evaluate

sensitivity to noise, decoherence and other challenges. The simulations performed during this project have assumed no noise, and the ability to accurately obtain the expectation value $\bar{C}$ for parameter optimisation.

- Classically simulating the algorithm for other choices of parameters. Namely, we have only considered here a chain of 5 residues, all of which are hydrophobic (H), and with energy parameters $(-1, 5, 10)$ only. Changes in each of these parameters can be explored, but as we are ultimately interested in performing this algorithm for large numbers of residues, the performance of this algorithm with increasing numbers of residues should be explored further.

- Parameter optimisation with other optimisation methods. Stochastic gradient descent and Adam are examples of common optimisers in machine learning and these can be explored. Whilst this may be less important for cases of high $p$, a good optimisation method can allow near-term implementation with small $p$ to obtain good results.

- Promoting the algorithm to using the Miyazawa-Jernigan (MJ) model, which uses interaction energies between every pair of amino acids. The MJ model requires 210 programmable interaction energies, which can even be encoded naively in the computational basis with only a constant (but potentially large) number of ancillary qubits. If an efficient (polynomial in depth and gates) way of pairs of residue types and calling the corresponding interaction energies can be found, the same circuit structure can be used and the circuit gate and depth complexity remain polynomial in $m$, possibly still $O(m^2(\log_2 m)^2)$ if such a method is constant in depth and gates.

# 6 Conclusion and Future Work

We have demonstrated the potential of quantum computing for solving Protein Folding in the 3D HP model, specifically using the QAOA and the explicit quantum circuits we obtained. Our final circuit has gate complexity $O(pm^2 \log m)$, depth complexity $O(pm^2 \log^2 m)$ and uses $3n(m+2) + 3m + k + 3$ qubits for a chain of $m+1$ residues, with $p$ QAOA depth, where $n = \lceil \log_2 m \rceil$ and $k$-bit precision is required for storing energy values.

The performance was simulated without noise on Magnus in the Pawsey Supercomputing Centre, with a brute-force search with $p = 2$ and a random search assisted with the Nelder-Mead algorithm for $p = 1, 2, 4, 8$, using more than 80 000 core-hours of compute time. Our results are evidence for the improved measurement performance with increasing $p$, with the use of a complete mixer, and without the use of an 'invalidity' penalty. With $p = 2$, it was possible to achieve as low as $R_{99} = 10$, and with $p = 8$ it was possible to measure optimal solutions in a single measurement with 99.99999% confidence.

In terms of the proposed circuit, the next steps include promoting the quantum algorithm from the Hydrophobic-Hydrophilic (HP) model to the Miyazawa-Jernigan (MJ) model to more accurately describe the behaviour of real proteins, and to also simulate the current algorithm on a quantum computer.

There is still much work to be done with simulating the circuit classically too, by exploring the performance of the algorithm with different residue sequences and sequence lengths ($m$). The use of different mixers or even combinations of mixers, as well as different energy parameters can also be explored.

Whilst these are favourable results, research into Protein Folding with Quantum Computing is still very new, as well as the field of Quantum Computing itself. But with increasing work being done in the field every day, we can be optimistic that Quantum Computing will be used for great things in the future.

# References


[1] Alejandro Perdomo-Ortiz et al. "Finding low-energy conformations of lattice protein models by quantum annealing". In: *Scientific Reports* 2.571 (2012).

[2] Tomáš Babej, Christopher Ing, and Mark Fingerhuth. "Coarse-grained lattice protein folding on a quantum annealer". In: *arXiv e-prints*, arXiv:1811.00713 (Nov. 2018), arXiv:1811.00713. arXiv: `1811.00713 [quant-ph]`.

[3] Mark Fingerhuth, Tomáš Babej, and Christopher Ing. "A quantum alternating operator ansatz with hard and soft constraints for lattice protein folding". In: *arXiv e-prints*, arXiv:1810.13411 (Oct. 2018), arXiv:1810.13411. arXiv: `1810.13411 [quant-ph]`.

[4] Michael McKerns and Michael Aivazis. *pathos: a framework for heterogenous computing*. Available at `http://trac.mystic.cacr.caltech.edu/project/pathos` (25/10/2019). 2010.

[5] Michael M. McKerns et al. "Building a Framework for Predictive Science". In: *CoRR* abs/1202.1056 (2012). arXiv: `1202.1056`. URL: `http://arxiv.org/abs/1202.1056`.

[6] Wolfram Research, Inc. *Mathematica, Version 12.0*. URL: `https://www.wolfram.com/mathematica`.

[7] Andrew M Childs. "Quantum Information Processing in Continuous Time". PhD thesis. 2004, pp. 18–19.

[8] Samuel Marsh and Jingbo Wang. "Efficient quantum walks over feasible solutions to NP-hard combinatorial problems". Manuscript submitted for publication. 2019.

[9] Brandon Ho, Anastasia Baryshnikova, and Grant W. Brown. "Unification of Protein Abundance Datasets Yields a Quantitative Saccharomyces cerevisiae Proteome". In: *Cell Systems* 6.2 (2018). DOI: `https://doi.org/10.1016/j.cels.2017.12.004`.

[10] Jovana Drinjakovic. *A Cell Holds 42 Million Protein Molecules, Scientists Reveal*. Available at: `https://medicine.utoronto.ca/news/cell-holds-42-million-protein-molecules-scientists-reveal` (12/01/2019). 2018.

[11] Robert Zwanzig, Attila Szabo, and Biman Bagchi. "Levinthal's paradox". In: *Proceedings of the National Academy of Sciences* 89.1 (1992).

[12] Sophie E Jackson. "How do small single-domain proteins fold?" In: *Folding and Design* 3.4 (1998), R81–R91.

[13] Joel L. Sussman et al. *P53*. Available at: `http://proteopedia.org/wiki/index.php/P53` (12/01/2019).

[14] Ghulum Md Ashraf et al. "Protein misfolding and aggregation in Alzheimer?s disease and Type 2 Diabetes Mellitus". In: *CNS Neurol Disord Drug Targets* 13.7 (2016).

[15] Patrick Sweeney et al. "Protein misfolding in neurodegenerative diseases: implications and strategies". In: *Translational Neurodegeneration* 6.6 (2017).

[16] Tapan K Chaudhuri and Subhankar Paul. "Protein?misfolding diseases and chaperone?based therapeutic approaches". In: *The FEBS Journal* 273.7 (2006), pp. 1331–1349.

[17] F Ulrich Hartl. "Protein Misfolding Diseases". In: *Annual Review of Biochemistry* 86 (2017), pp. 21–26.

[18] Centers for Disease Control and Prevention. *Alzheimer's Disease and Related Dementias*. Available at https://www.cdc.gov/aging/aginginfo/alzheimers.htm (19/9/2019).

[19] Alzheimer's Association. *Alzheimer's and Dementia in Australia*. Available at: https://www.alz.org/au/dementia-alzheimers-australia.asp (19/9/2019).

[20] National Institute on Aging. *Basics of Alzheimer's Disease and Dementia: What is Alzheimer's Disease?* Available at: https://www.nia.nih.gov/health/what-alzheimers-disease (19/9/2019).

[21] Dementia Australia. *Dementia Prevalence Data 2018-2058*. Available at https://www.dementia.org.au/information/statistics/prevalence-data (19/9/2019).

[22] Dementia Australia. *Alzheimer's disease*. Available at https://www.dementia.org.au/about-dementia/types-of-dementia/alzheimers-disease (19/9/2019).

[23] Australian Bureau of Statistics. *3105.0.65.001 - Australian Historical Population Statistics, 2019: 2. Population Age and Sex Structure*. Available at https://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/3105.0.65.001Main+Features12016?OpenDocument (19/9/2019).

[24] Parkinson's Foundation. *What Is Parkinson's?* Available at: https://www.parkinson.org/understanding-parkinsons/what-is-parkinsons.

[25] National Institute on Aging. *Parkinson's Disease*. Available at: https://www.nia.nih.gov/health/parkinsons-disease (19/9/2019).

[26] Parkinson's Australia Inc. and Deloitte Touche Tohmatsu Limited. *Living with Parkinson's Disease: An updated economic analysis 2014*. Tech. rep.

[27] Genetics Home Reference. *Huntington disease*. Available at: https://ghr.nlm.nih.gov/condition/huntington-disease.

[28] Brain Foundation. *Huntington's Disease*. Available at: https://brainfoundation.org.au/disorders/huntingtons-disease/.

[29] Johns Hopkins Medicine. *Prion Diseases*. Available at: https://www.hopkinsmedicine.org/health/conditions-and-diseases/prion-diseases.

[30] Centers for Disease Control and Prevention. *Prion Diseases*. Available at: https://www.cdc.gov/prions/index.html.

[31] Alzheimer's Association. *Creutzfeldt-Jakob Disease*. Available at: https://www.alz.org/alzheimers-dementia/what-is-dementia/types-of-dementia/creutzfeldt-jakob-disease.

[32] Hao Gong et al. "Amyloidogenicity of p53: A Hidden Link Between Protein Misfolding and Cancer". In: *Current Protein & Peptide Science* 16.2 (2015).

[33] Jerson L Silva et al. "Ligand Binding and Hydration in Protein Misfolding: Insights from Studies of Prion and p53 Tumor Suppressor Proteins". In: *Accounts of Chemical Research* 43.2 (2010).

[34] Ana P D Ano Bom et al. “Mutant p53 aggregates into prion-like amyloid oligomers and fibrils: implications for cancer”. In: *Journal of Biological Chemistry* 287.33 (2012).

[35] Abhisek Mukherjee et al. “Type 2 Diabetes as a Protein Misfolding Disease”. In: *Trends in Molecular Medicine* 21.7 (), pp. 439–449.

[36] R Evans et al. “De novo structure prediction with deep-learning based scoring”. In: *Thirteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstracts) 1-4 December 2018*. 2018.

[37] Rob L M van Montfort and Paul Workman. “Structure-based drug design: aiming for a perfect fit”. In: *Essays in Biochemistry* 61 (2017), pp. 431–437.

[38] Pablo Echenique. “Introduction to protein folding for physicists”. In: *Contemporary Physics* 48 (Mar. 2007), pp. 81–108. DOI: 10.1080/00107510701520843. arXiv: 0705.1845 [physics.bio-ph].

[39] Amit Kessel and Nir Ben-Tal. *Introduction to Proteins*. Boca Raton, Florida: CRC Press, 2011.

[40] Biological Magnetic Resonance Data Bank. *ASCII table summarizing all amino acid data*. Available at: http://www.bmrb.wisc.edu/referenc/aa_tables.shtml (12/01/2019).

[41] NEUROtiker. *Structure of histidine*. Available at: https://commons.wikimedia.org/wiki/File:Histidin_-_Histidine.svg (12/01/2019). 2007.

[42] Sylvanie Surget, Marie P. Khoury, and Jean-Christophe Bourdon. “Uncovering the role of p53 splice variants in human malignancy: a clinical perspective”. In: *Onco Targets Ther.* 7 (2013), pp. 57–68.

[43] Bioinformatics. *Primary information of p53 gene*. Available at: http://www.bioinformatics.org/p53/introduction.html (12/01/2019).

[44] Jevgenij Raskatov and David Teplow. “Using chirality to probe the conformational dynamics and assembly of intrinsically disordered amyloid proteins”. In: *Scientific Reports* 7 (Dec. 2017). DOI: 10.1038/s41598-017-10525-5.

[45] Ken A. Dill and Justin L. MacCallum. “The Protein-Folding Problem, 50 Years On”. In: *Science* 338.6110 (2012), pp. 1042–1046. ISSN: 00368075, 10959203. URL: http://www.jstor.org/stable/41703986.

[46] Ken A Dill et al. “The protein folding problem: when will it be solved?” In: *Current Opinion in Structural Biology* 17.3 (2007). Nucleic acids / Sequences and topology, pp. 342–346. ISSN: 0959-440X. DOI: https://doi.org/10.1016/j.sbi.2007.06.001. URL: http://www.sciencedirect.com/science/article/pii/S0959440X07000772.

[47] Brankica G Jankovic and Natalija D Polovic. “The protein folding problem”. In: *Biologica Serbica* 39.1 (2017), pp. 105–111. DOI: https://doi.org/10.5281/zenodo.827151. URL: https://ojs.pmf.uns.ac.rs/index.php/dbe_serbica/article/view/6530/85.

[48] C. Levinthal. “How to Fold Graciously”. In: *Mossbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House, Monticello, Illinois*. 1969.

[49] Peter L Freddolino et al. “Ten-Microsecond Molecular Dynamics Simulation of a Fast-Folding WW Domain”. In: *Biophysical Journal* 96.10 (), pp. L75–L77.

[50] David E. Shaw et al. “Millisecond-scale Molecular Dynamics Simulations on Anton”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC ’09. Portland, Oregon: ACM, 2009, 39:1–39:11. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654099. URL: http://doi.acm.org/10.1145/1654059.1654099.

[51] D. E. Shaw et al. “Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer”. In: *SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014, pp. 41–53.

[52] Ngaam J. Cheung and Wookyung Yu. “De novo protein structure prediction using ultra-fast molecular dynamics simulation”. In: *PLOS ONE* 13 (Nov. 2018), pp. 1–17. URL: https://doi.org/10.1371/journal.pone.0205819.

[53] Jinbo Xu. “Distance-based protein folding powered by deep learning”. In: *Proceedings of the National Academy of Sciences* 116.34 (2019), pp. 16856–16865. DOI: 10.1073/pnas.1821309116.

[54] Mohamad Yousef, Tamer Abdelkader, and Khaled El-Bahnasy. “Performance comparison of ab initio protein structure prediction methods”. In: *Ain Shams Engineering Journal* (2019). ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2019.03.004. URL: http://www.sciencedirect.com/science/article/pii/S2090447919300565.

[55] Nicola Yanev et al. “Protein Folding Prediction in a Cubic Lattice in Hydrophobic-Polar Model”. In: *Journal of Computational Biology* 24.5 (2017), pp. 412–421.

[56] Cheng-Hong Yang et al. “Protein folding prediction in the HP model using ions motion optimization with a greedy algorithm”. In: *BioData Mining* 11.17 (2018).

[57] Yanjun Li et al. “FoldingZero: Protein Folding from Scratch in Hydrophobic-Polar Model”. In: *CoRR* abs/1812.00967 (2018). arXiv: 1812.00967. URL: http://arxiv.org/abs/1812.00967.

[58] Daniel Varela and José Santos. “Automatically Obtaining a Cellular Automaton Scheme for Modeling Protein Folding Using the FCC Model”. In: 18.2 (June 2019), pp. 275–284. ISSN: 1567-7818. DOI: 10.1007/s11047-018-9705-y. URL: https://doi.org/10.1007/s11047-018-9705-y.

[59] Sanzo Miyazawa and Robert L. Jernigan. “Residue–Residue Potentials with a Favorable Contact Pair Term and an Unfavorable High Packing Density Term, for Simulation and Threading”. In: *Journal of Molecular Biology* 256 (1996), pp. 623–644.

[60] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[61] *Dynamics: Description and Discussion*. URL: https://www.charmm.org/charmm/documentation/by-version/c42b1/params/doc/dynamc/.

[62] Arnold Neumaier. “Molecular Modeling of Proteins and Mathematical Prediction of Protein Structure”. In: *SIAM Review* 39 (Jan. 1997), pp. 407–460.

[63] Wendy D. Cornell et al. “A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules”. In: *Journal of the American Chemical Society* 117.19 (1995), pp. 2179–2197.

[64] Mingchen Chen et al. “Protein Folding and Structure Prediction from the Ground Up: The Atomistic Associative Memory, Water Mediated, Structure and Energy Model”. In: *Journal of the American Chemical Society* 120.33 (2016), pp. 8557–8565.

[65] Robert Schleif. *A Concise Guide to CHARMM and the Analysis of Protein Structure and Function*. Tech. rep. Johns Hopkins University, 2013. URL: http://pages.jh.edu/~rschlei1/Random_stuff/publications/charmmbook.pdf.

[66] Helen M Berman et al. “The Protein Data Bank”. In: *Nucleic Acids Research* 28.1 (1 2000), pp. 235–242.

[67] Jinbo Xu and Sheng Wang. “Analysis of distance-based protein structure prediction by deep learning in CASP13”. In: *Proteins: Structure, Function, and Bioinformatics* (). DOI: 10.1002/prot.25810. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.25810. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.25810.

[68] Chengxin Zhang et al. “Template-based and free modeling of I-TASSER and QUARK pipelines using predicted contact maps in CASP12.” In: *Proteins: Structure, Function and Bioinformatics* 86 (2018), pp. 136–151.

[69] Jianlin Cheng et al. “Estimation of model accuracy in CASP13”. In: *Proteins* (), pp. 1–17.

[70] Mohammed AlQuraishi. “AlphaFold at CASP13”. In: *Bioinformatics* (May 2019). btz422. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz422. eprint: http://oup.prod.sis.lan/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btz422/28835737/btz422.pdf. URL: https://doi.org/10.1093/bioinformatics/btz422.

[71] Matthew Hutson. “AI protein-folding algorithms solve structures faster than ever”. In: *Nature* (July 2019). DOI: 10.1038/d41586-019-01357-6.

[72] Online Etymology Dictionary. *computer (n.)* Available at https://www.etymonline.com/word/computer (2019/08/24).

[73] Ernst Martin. *The Calculating Machines (Die Rechenmaschinen) Their History and Development*. 16th ed. Cambridge, Massachusetts: The MIT Press, 1992.

[74] B. Jack Copeland. *Colossus: The secrets of Bletchley Park’s code-breaking computers*. 1st ed. New York: Oxford University Press, 2006.

[75] Gordon E. Moore. “Cramming more components onto integrated circuits”. In: *Electronics* 38.8 (1965).

[76] F L Marquezino, R Portugal, and F D Sasse. “Obtaining the Quantum Fourier Transform from the classical FFT with QR decomposition”. In: *Journal of Computational and Applied Mathematics* 235 (2010), pp. 74–81.

[77] Michael A Nielsen and Issac L Chuang. 2nd ed. University Printing House, Cambridge CB2 8BS, United Kingdom: Cambridge University Press, 2010.

[78] Richard P. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* 21.6/7 (1982).

[79] Thomas G Draper and Samuel A Kutin. *¡q—pic¿: Quantum Circuit Diagrams in LaTeX*. Available from: `https://github.com/qpic/qpic` (25/10/2019). 2016.

[80] Andrew M. Childs and Wim van Dam. "Quantum algorithms for algebraic problems". In: *Rev. Mod. Phys.* 82 (1 Jan. 2010), pp. 1–52. DOI: `10.1103/RevModPhys.82.1`. URL: `https://link.aps.org/doi/10.1103/RevModPhys.82.1`.

[81] C. Monroe et al. "Demonstration of a Fundamental Quantum Logic Gate". In: *Phys. Rev. Lett.* 75 (25 Dec. 1995), pp. 4714–4717. DOI: `10.1103/PhysRevLett.75.4714`. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.75.4714`.

[82] M. H. Devoret, A. Wallraff, and J. M. Martinis. *Superconducting Qubits: A Short Review*. 2004. arXiv: `cond-mat/0411174 [cond-mat.mes-hall]`.

[83] Philip Krantz et al. *A Quantum Engineer's Guide to Superconducting Qubits*. 2019. arXiv: `1904.06560 [quant-ph]`.

[84] Sergei Slussarenko and Geoff J Pryde. *Photonic quantum information processing: a concise review*. 2019. arXiv: `1907.06331 [quant-ph]`.

[85] Simon J. Devitt. "Performing quantum computing experiments in the cloud". In: *Physical Review A* 94.3 (Sept. 2016). ISSN: 2469-9934. DOI: `10.1103/physreva.94.032329`. URL: `http://dx.doi.org/10.1103/physreva.94.032329`.

[86] Joseph Bardin et al. "A 28nm Bulk-CMOS 4-to-8GHz ¡2mW Cryogenic Pulse Modulator for Scalable Quantum Computing". In: *Proceedings of the 2019 International Solid State Circuits Conference*. 2019, pp. 456–458.

[87] M W Johnson et al. "Quantum annealing with manufactured spins". In: *Nature* 473 (), pp. 194–198.

[88] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510. ISSN: 1476-4687. DOI: `10.1038/s41586-019-1666-5`. URL: `https://doi.org/10.1038/s41586-019-1666-5`.

[89] Edwin Pednault et al. *On "Quantum Supremacy"*. Available from: `https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/`. 2019.

[90] Michael Brooks. "Beyond quantum supremacy: the hunt for useful quantum computers". In: *Nature* 574 (2019), pp. 19–21.

[91] Jingbo Wang. Personal conversation. Oct. 2019.

[92] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A Quantum Approximate Optimization Algorithm". In: *arXiv e-prints*, arXiv:1411.4028 (Nov. 2014), arXiv:1411.4028. arXiv: `1411.4028 [quant-ph]`.

[93] Stuart Hadfield et al. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz". In: *arXiv e-prints*, arXiv:1709.03489 (Sept. 2017), arXiv:1709.03489. arXiv: `1709.03489 [quant-ph]`.

[94] Samuel Marsh and Jingbo Wang. *A quantum walk assisted approximate algorithm for bounded NP optimisation problems*. 2018. arXiv: 1804.08227 [quant-ph].

[95] Weng-Long Chang. "Fast Quantum Algorithm of Solving the Protein Folding Problem in the Two-Dimensional Hydrophobic–Hydrophilic Model on a Quantum Computer". In: *Emerging Research in Artificial Intelligence and Computational Intelligence*. Ed. by Hepu Deng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 483–490. ISBN: 978-3-642-24282-3.

[96] S IRIYAMA, Masanori Ohya, and I YAMATO. "QUANTUM ALGORITHM FOR PROTEIN FOLDING AND ITS COMPUTATIONAL COMPLEXITY". In: Mar. 2013, pp. 193–202. DOI: 10.1142/9789814460026_0017.

[97] Ryan Babbush et al. "Construction of Energy Functions for Lattice Heteropolymer Models: A Case Study in Constraint Satisfaction Programming and Adiabatic Quantum Optimization". In: *arXiv e-prints*, arXiv:1211.3422 (Nov. 2012), arXiv:1211.3422. arXiv: 1211.3422 [quant-ph].

[98] Preethika Kumar. "Direct implementation of an N-qubit controlled-unitary gate in a single step". In: *Quantum Information Processing* 12.2 (Feb. 1, 2013), pp. 1201–1223.

[99] Mehdi Saeedi and Massoud Pedram. "Quantum Circuits for n-qubit Toffoli gate with no Ancilla". In: (2013).

[100] Python Software Foundation. *Python 3.7.3*. Available from: https://www.python.org/. 2019.

[101] Travis E Oliphant. *A guide to NumPy*. USA: Trelgol Publishing, 2006.

[102] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30.

[103] Lisandro D. Dalcin et al. "Parallel distributed computing using Python". In: *Advances in Water Resources* 34.9 (2011). New Computational Methods and Software Tools, pp. 1124–1139. ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2011.04.013. URL: http://www.sciencedirect.com/science/article/pii/S0309170811000777.

[104] Lisandro Dalcin et al. "MPI for Python: Performance improvements and MPI-2 extensions". In: *Journal of Parallel and Distributed Computing* 68.5 (2008), pp. 655–662. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2007.09.005. URL: http://www.sciencedirect.com/science/article/pii/S0743731507001712.

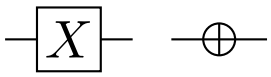
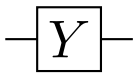
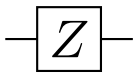
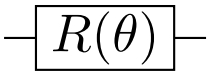
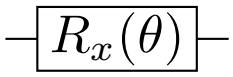
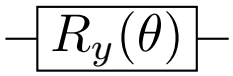
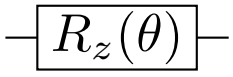
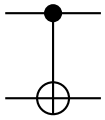
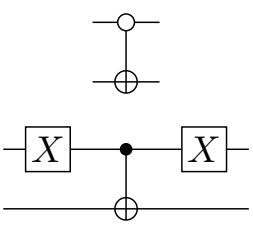
[105] Lisandro Dalcin, Rodrigo Paz, and Mario Storti. "MPI for Python". In: *Journal of Parallel and Distributed Computing* 65.9 (2005), pp. 1108–1115. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2005.03.010. URL: http://www.sciencedirect.com/science/article/pii/S0743731505000560.

[106] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[107] Pawsey Supercomputing Centre. *Magnus*. Available from: `https://pawsey.org.au/systems/magnus/`. 2019.

[108] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum networks for elementary arithmetic operations". In: 54.1 (July 1996), pp. 147–153.

# Appendix 1: Quantum Gate Reference Guide

| **Gate name(s)**<br>**Description** | **Symbol(s)** | **Matrix** |
|---|---|---|
| Hadamard (H)<br>Maps $\|0\rangle$ to $\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$<br>and $\|1\rangle$ to $\frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$ |  | $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$ |
| Pauli-X ($\sigma_x$) / NOT (X)<br>'Flips' the qubit, like a traditional NOT gate |  | $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$ |
| Pauli-Y ($\sigma_y$/Y) |  | $\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$ |
| Pauli-Z ($\sigma_z$/Z) |  | $\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$ |
| Phase |  | $\begin{pmatrix}1 & 0\\ 0 & e^{i\theta}\end{pmatrix}$ |
| X Rotation |  | $e^{-i\sigma_x\theta/2} = \begin{pmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$ |
| Y Rotation |  | $e^{-i\sigma_y\theta/2} = \begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$ |
| Z Rotation |  | $e^{-i\sigma_z\theta/2} = \begin{pmatrix}e^{-\frac{i\theta}{2}} & 0\\ 0 & e^{\frac{i\theta}{2}}\end{pmatrix}$ |
| Controlled NOT (CNOT)<br>Applies 'NOT' to the second/target qubit if the first/control qubit is $\|1\rangle$ |  | $\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$ |
| Open controlled NOT (CNOT)<br>Applies 'NOT' to the secnod/target qubit if the first/control qubit is $\|0\rangle$ |  | $\begin{pmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$ |

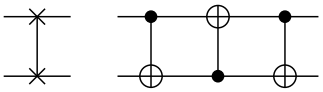
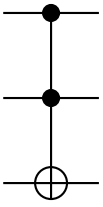
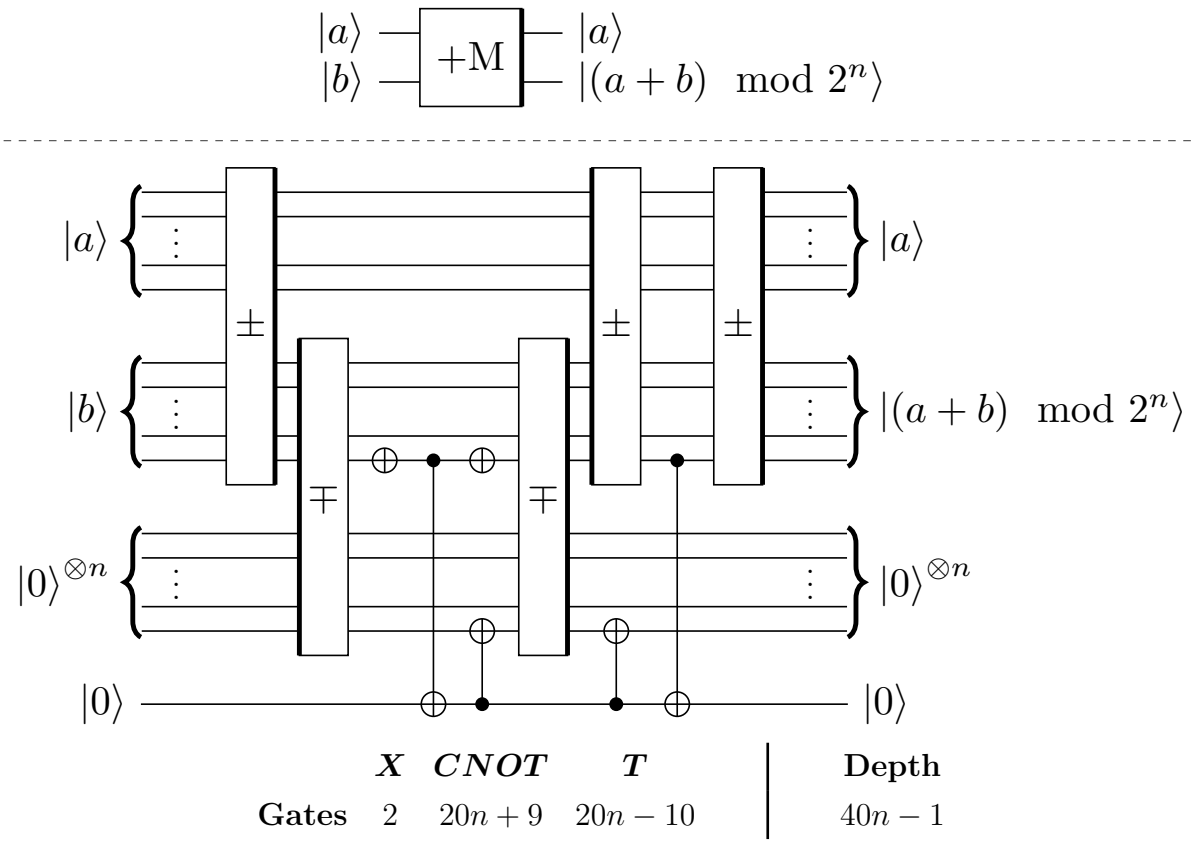
| | | |
|---|---|---|
| SWAP |  | $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ |
| Toffoli (T)<br>Applies 'NOT' to third/target qubit if other/control qubits are all $\|1\rangle$ |  | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$ |

Table 5: A table of common gates in the Quantum Gate model of Quantum Computing

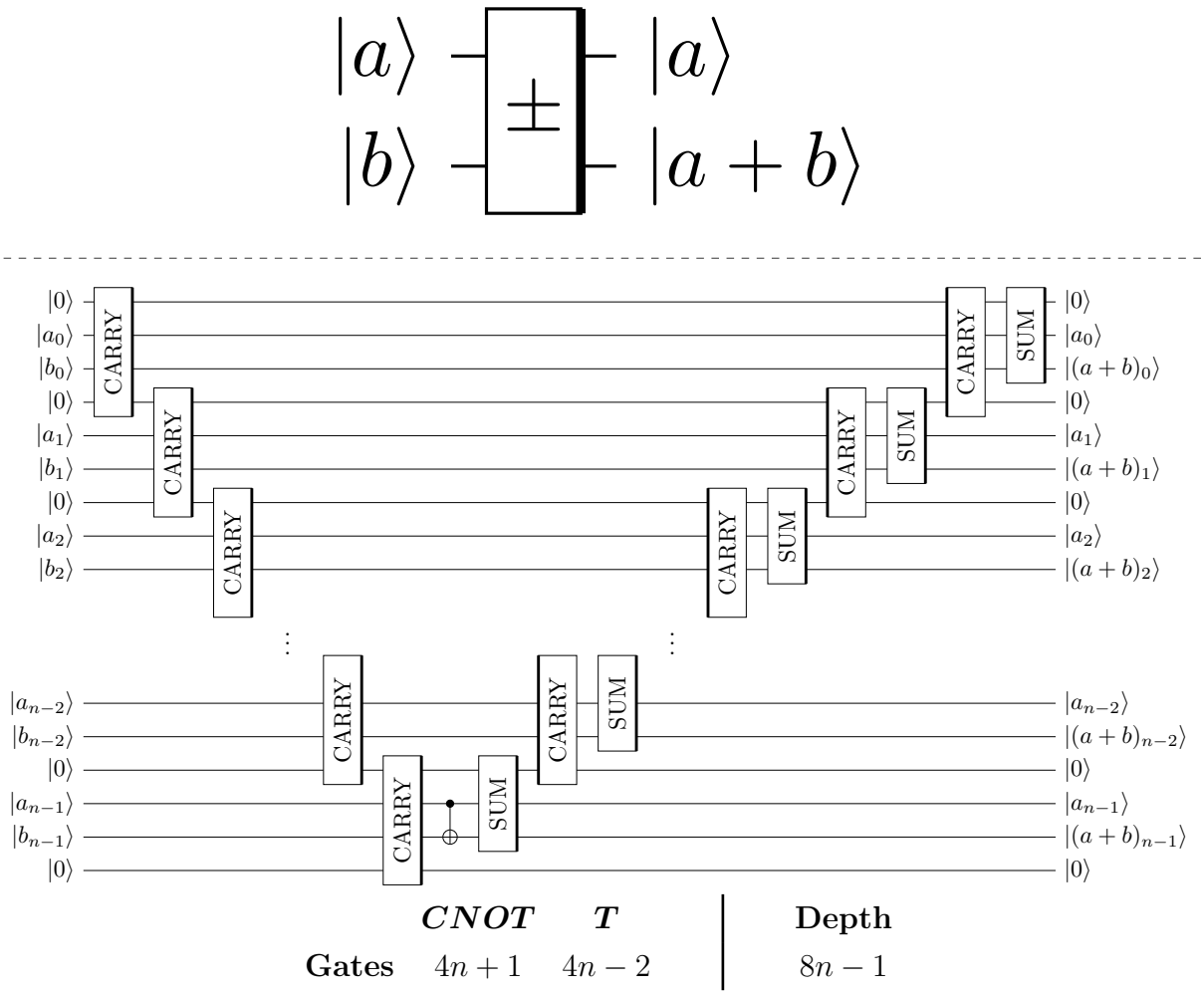
Circuit 16: $[+M]$




| | ***X*** | ***CNOT*** | ***T*** | **Depth** |
|---|---|---|---|---|
| **Gates** | 2 | $20n+9$ | $20n-10$ | $40n-1$ |

Circuit 16: Modular adder circuit from [108]

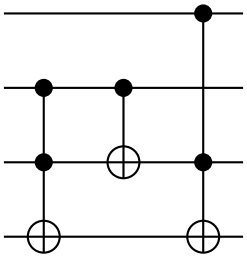
Circuit 17: [±]




| | ***CNOT*** | ***T*** | **Depth** |
|---|---|---|---|
| **Gates** | $4n+1$ | $4n-2$ | $8n-1$ |

Circuit 17: Adder circuit from [108]. The ± gate performs the operation $|a\rangle\,|b\rangle \to |a\rangle\,|a+b\rangle$, and we denote the operation $|a\rangle\,|b\rangle \to |a+b\rangle\,|b\rangle$ as $\mp$. Note that in this circuit, $a_i$ is the $i$th term in the binary expansion of $a$, and $a_{n-1}$ represents the most significant bit in the binary expansion.

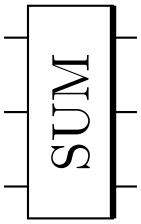
Circuit 18: [CARRY]

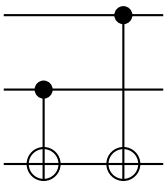



| | *CNOT* | *T* | **Depth** |
|---|---|---|---|
| **Gates** | 1 | 2 | 3 |

Circuit 18: Carry circuit from [108]

Circuit 19: [SUM]



| | *CNOT* | **Depth** |
|---|---|---|
| **Gates** | 2 | 2 |

Circuit 19: Sum circuit from [108]

# Appendix 2: Project Proposal and Deviations

This appendix contains the proposal document for this project, presented near the beginning of the project.

The major deviation in this thesis from the proposal is the use of a lattice protein folding model, instead of the scheme in the proposal which aimed to simulate the classical dynamics of protein folding by treating chains of residues (referred to as 'linkages'). This decision was made as there were many challenges foreseen with the scheme, such as how to extract information for the evolution of a linkage over time. Previous literature describing the use of quantum computing with lattice protein models also influenced this decision.

## 1. Aims

This project aims to develop a quantum algorithm that can determine the minimal energy configuration of a complex fixed-angle linkage, given the linkage parameters (such as link lengths and fixed angles). Such an algorithm would have applications in mechanical engineering, but perhaps more significantly in biochemistry, contributing understanding towards the Protein Folding problem.

The Protein Folding problem is one of the great open questions in biochemistry, first posed more than 50 years ago (Dill & MacCallum, 2012). Proteins are long branched chains of amino acid monomers, and typically contain 400-500 monomers each (Stoker, 2016), and are vital for living organisms. Due to their structure and composition, a single protein can be modelled as a linkage with many links and joints, and have an enormous number of configurations, yet are mostly able to fold into their minimal energy configurations consistently and quickly (Dill & MacCallum, 2012). The Protein Folding process investigates aspects of this folding, and can be stated in three parts (Dill & MacCallum, 2012) (Dill et. al., 2007):

1. How is the folded structure of a protein determined by its amino acid sequence?
2. What are the physical mechanisms which fold proteins?
3. Can the native structure of a protein be predicted from its amino acid sequence?

Quantum computation has emerged recently as a potential tool to solve computational problems that would otherwise be impossible with classical computers. One reason for quantum computing's advantage over its classical counterpart is the clever use of the phenomenon of quantum superposition. Whereas a single classical computer can only consider one case at a time, a quantum computer can, in a sense, consider many cases at once. Quantum algorithms already exist which take advantage of this to solve problems faster, such as Shor's algorithm (Shor, 1997), or the Deutsch-Jouza algorithm (Deutsch & Jouza, 1992).

## 2. Significance

In the 50 odd years since the introduction of the Protein Folding problem, a huge amount of progress has been made on each of the three parts of the problem. For the purposes of this proposal, we restrict ourselves to results pertaining to the prediction of structure from the amino acid sequence.

Two main computational approaches exist for modelling protein folding using classical computing, a bioinformatics-based approach, and a physics-based approach. The latter has seen success since 1998, with the successful folding simulation of a 36-residue polypeptide, accurate up to 4.5 $\mathring{A}$ (Duan & Kollman, 1998), followed by the simulation of a 20-residue peptide, accurate up to 1 $\mathring{A}$.

The successful simulation of a small protein or polypeptide folding via quantum computing may serve as a proof of concept for folding simulation using quantum computing, and would subsequently lead to more research into such algorithms. This in turn would contribute greatly towards the simulation of protein folding, and thus assist in the creation of new drugs and understanding of mutations in proteins via misfolding, which can lead to diseases such as Alzheimer's, Parkinson's or Huntington's (Bross & Gregersen, 2003).

The successful creation of such an algorithm would also be a new step in quantum computing and information, as to the writer's knowledge there currently do not exist any quantum algorithms for simulation of classical mechanics (Jordan, 2018).

## 3. Methods

The problem of Protein Folding is ideal for Quantum Computing, due to the exponential growth of the configuration space of linkages for increased links. When discretized to only allow a discrete rotation angles, a linear linkage with 3 vertices has a configurations, up to rotation and translation of the entire linkage in space. However, this is scaled exponentially with increased links, as 4 vertices results in order $a^2$, 5 vertices in order $a^3$, and $n$ vertices in $a^{n-2}$. The base a can also increase to allow for less discrete changes in link orientations, though the number of allowed configurations only increases in polynomial time with such increases.

One such approach to develop such an algorithm is as follows:

1. Develop a way to represent linkages in terms of link length, fixed link angles and the linkage configuration space using qubits
2. Develop a quantum algorithm which can perform calculations with linkages e.g. a 'product' of two configuration spaces of two linkages to produce a configuration space of the combined linkage

3. Determine the configuration space of an equiangular, equilateral linkage of arbitrary length and investigate its properties, including connectesness
4. Relax the equiangular and equilateral constraints, still only obtuse link angles and repeat steps 3 and 4
5. Assign energies to each configuration of such a linkage
6. Develop an algorithm to determine the configuration with minimal energy given a linkage

Quantum algorithms can be simulated on classical computers, but especially for more complex problems, can be extremely time-consuming, so the algorithms can be tested classically by considering small systems only, or by using a supercomputer.

The algorithms can also be tested on the free IBM Q Experience app (IBM, 2018), which is a free app which allows users to send quantum algorithms to one of IBM's small quantum computers (5 or 20 qubits in size).

## 4. Benefits

The success of this project will have implications for both the fields of biochemistry and of quantum computation.

Biochemists will have a practical way to simulate proteins folding and potentially other large molecules as well, and these can be used to verify new theoretical models of protein folding.

Such an algorithm may also aid in designing linkages for specific motions or tasks.

As far as the writer is aware, there does not yet exist a widely-accepted[10] quantum algorithm which simulates classical mechanical systems, so such an algorithm would be the first of its kind and may form a basis for further study into simulation of physical systems via quantum computation.

## 5. Publications

### 5.1. Erik Demaine

Erik Demaine is a Professor of Electrical Engineering and Computer Science at MIT, having completed a Bachelor's degree in Science, a Master's degree in Mathematics, and a PhD with thesis entitled 'Folding and Unfolding'. Erik's PhD thesis addressed the Carpenter's Rule

[10]The private company D-Wave Systems published a paper in 2012 claiming the successful prediction of folded protein states. However, the correct protein ground states were only obtained 13 out of 105 measurements (Perdomo-Ortiz, Dickson, Drew-Brook, Rose & Aspuru-Guzik, 2012) and many in the quantum computing community dispute D-Wave's claims that their systems actually perform quantum computation (Hsu, 2013) (Shin, Smith, Smolin & Vazirani, 2014) (Gibney, 2017).

Problem (Demaine, 2012) and was awarded the 2003 NSERC Doctoral Prize (Demaine, 2018a), and he was also awarded with a MacArthur Fellowship the same year. Erik was also the youngest professor at MIT, accepting the position at the age of 21 (NSERC, 2003).

Erik's research interests span mathematics and computer science, particularly towards algorithms (Demaine, 2018b), and has published many articles, from those on computational complexity of origami crease patterns (Akitaya, Demaine & Ku, 2017) to geometric models of protein construction and folding (Demaine, Langerman & O'Rourke, 2006) (Aichholzer et al, 2003) and in many other areas.

### 5.2. Ken Dill

Ken Dill is a SUNY Distinguished Professor of Chemistry and Physics at Stony Brook University, and the director of the University's Laufer Center for Physical and Quantitative Biology, having completed Bachelor's and Master's degrees in Mechanical Engineering, a PhD in Biology and post-doctoral studies in Chemistry. Ken is an editor for Annual Reviews Biophysics for Protein Engineering and Physical Biology amongst other fields, and has won numerous awards and fellowships over his career (Dill Research Group, 2018).

The Dill Research Group studies the physics of proteins through methods in statistical physics and computational biology and has published two books on the subject: Molecular Driving Forces and Protein Action, Principles and Modeling.

## 6. Costs

As at this stage the writer's knowledge of quantum computation and information and biochemistry pertaining to protein folding is quite elementary, it is difficult to predict the timescale of this project or the manpower required. However, the project is estimated to take 2 years.

Other costs associated with the project include computational power to test the developed algorithms. There may be costs associated with using a supercomputer for this, or in future there may be ways to use real, privately owned quantum computers to test the algorithm, for a fee.

## 7. References

Aichholzer, O., Bremner, D., Demaine, E. D., Meijer, H., Sacristan, V., & Soss, M. (2003). Long proteins with unique optimal foldings in the H-P model. Computational Geometry, 25(1), 139-159. doi:https://doi.org/10.1016/S0925-7721(02)00134-7

Akitaya, H. A., Demaine, E. D., & Ku, J. S. (2017). Simple Folding is Really Hard. Journal of Information Processing, 25, 580-589. doi:10.2197/ipsjjip.25.580

Bross, P., & Gregersen, N. (2003). Protein Misfolding and Disease. Totowa, New Jersey: Humana Press.

Demaine, E. (2012). Carpenter's Rule Theorem. Retrieved from http://erikdemaine.org/linkage/

Demaine, E. (2018a). Erik Demaine's Papers. Retrieved from http://erikdemaine.org/papers/

Demaine, E. (2018b). More about Erik Demaine. Retrieved from http://erikdemaine.org/about/

Demaine, E. D., Langerman, S., & O'Rourke, J. (2006). Geometric Restrictions on Producible Polygonal Protein Chains. Algorithmica, 44(2), 167-181. doi:10.1007/s00453-005-1205-7

Deutsch, D., & Jozsa, R. (1992). Rapid Solution of Problems by Quantum Computation. Proceedings: Mathematical and Physical Sciences, 439(1907), 553-558.

Dill, K. A., & MacCallum, J. L. (2012). The Protein-Folding Problem, 50 Years On. Science, 338(6110), 1042-1046. doi:10.1126/science.1219021

Dill, K. A., Ozkan, S. B., Weikl, T. R., Chodera, J. D., & Voelz, V. A. (2007). The protein folding problem: when will it be solved? Current Opinion in Structural Biology, 17(3), 342-346. doi:https://doi.org/10.1016/j.sbi.2007.06.001

Duan, Y., & Kollman, P. A. (1998). Pathways to a Protein Folding Intermediate Observed in a 1-Microsecond Simulation in Aqueous Solution. Science, 282(5389), 740-744.

Gibney, E. (2017). Quantum computer gets design upgrade. Nature, 541(7638), 447-448.

Hsu, J. (2013). D-Wave's Year of Computing Dangerously. Retrieved from https://spectrum.ieee.org/computing/hardware/dwaves-year-of-computing-dangerously

IBM. (2018). IBM Q Experience. Retrieved from https://quantumexperience.ng.bluemix.net/qx

Jordan, S. (2018). Quantum Algorithm Zoo. Retrieved from https://math.nist.gov/quantum/zoo/

NSERC. (2003). Past Winner 2003 NSERC Doctoral Prize. Retrieved from http://www.nserc-crsng.gc.ca/Prizes-Prix/Doctoral-Doctorat/Profiles-Profils_eng.asp?ID=1026

Perdomo-Ortiz, A., Dickson, N., Drew-Brook, M., Rose, G., & Aspuru-Guzik, A. (2012). Finding low-energy conformations of lattice protein models by quantum annealing. Scientific Reports, 2, 571. doi:10.1038/srep00571

Shin, W. S., Smith, G., Smolin, J. A., & Vazirani, U. (2014). How 'Quantum' is the D-Wave Machine? Retrieved from https://arxiv.org/pdf/1401.7087.pdf

Shor, P. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal on Computing, 26(5), 1484-1509. doi:10.1137/S0097539795293172

Stoker, H. S. (2016). Organic and Biological Chemistry (7th ed.): Cengage Learning.

The Dill Research Group. (2018). Retrieved from `http://dillgroup.org/#/home`