NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
INTER-FACULTY POSTGRADUATE STUDIES PROGRAMME
DATA SCIENCE AND MACHINE LEARNING

Hybrid Quantum - Classical Machine Learning for the Problem of Protein-Folding

comparison of VQE and QAOA
& implementation on IBM's real devices

Master Thesis by

Maria Papanikolaou

maripapanikola@gmail.com
S.N. 03400188

Instructors

Prof. Loulakis Michail
Prof. Kominis Iannis
Dr. Blekos Kostas

QUANTUM NEURAL
TECHNOLOGIES S.A.

June, 2024

Table of Contents

List of Figures

List of Tables

Abstract

The present study experimentally tested the ability of hybrid quantum-classical machine learning to solve the problem of Protein Folding. Protein folding is a challenging problem to be solved with classical computing, due to its complexity which increases exponentially as the size of the amino-acids chain - that the protein consists of - is increased. Thus, since quantum computing offers a speedup in convergence, we manage to solve this task by utilizing quantum gates that will search the conformational space faster. For this, we used and compared two types of Variational Quantum Algorithms - Variational Quantum Eigensolver (VQE) and Quantum Approximate Optimization Algorithm (QAOA) - and we present the results in this thesis. We implemented them on a quantum simulator and, additionally, we implemented QAOA -who had better efficiency- on the real quantum devices of IBM. Finally, despite of the limited accuracy that we observe, we propose a solution that will leverage the beneficial outcomes of our implementation.

Περίληψη

Η παρούσα μελέτη δοκίμασε πειραματικά την ικανότητα της υβριδικής κλασικής-κβαντικής μηχανικής μάθησης να επιλύσει το πρόβλημα της αναδίπλωσης πρωτεϊνών. Η αναδίπλωση πρωτεϊνών είναι ένα δύσκολο πρόβλημα για να επιλυθεί με κλασικούς υπολογιστές, λόγω της πολυπλοκότητάς του που αυξάνεται εκθετικά καθώς το μέγεθος της αλυσίδας αμινοξέων που συνθέτουν την πρωτεΐνη αυξάνεται. Έτσι, καθώς η κβαντική υπολογιστική προσφέρει επιτάχυνση στη σύγκλιση, καταφέραμε να λύσουμε αυτό το έργο χρησιμοποιώντας κβαντικές πύλες που θα αναζητήσουν τον διαμορφωτικό χώρο πιο γρήγορα. Για αυτό, χρησιμοποιήσαμε και συγκρίναμε δύο τύπους Παραμετροποιημένων Κβαντικών Αλγορίθμων - Παραμετροποιημένου Κβαντικού Υπολογιστή (VQE) και της Κβαντικής Προσεγγιστικής Βελτιστοποιητικής Μεθόδου (QAOA) - και παρουσιάζουμε τα αποτελέσματα αυτής της εργασίας. Τις υλοποιήσαμε σε κβαντικό εξομοιωτή και, επιπλέον, υλοποιήσαμε την QAOA -η οποία είχε καλύτερη αποδοτικότητα- στις πραγματικές κβαντικές συσκευές της IBM. Τέλος, παρά την περιορισμένη ακρίβεια που παρατηρούμε, προτείνουμε μια λύση που θα αξιοποιήσει τα ωφέλιμα αποτελέσματα της υλοποίησής μας.

Chapter 1

Quantum Machine Learning

1.1 Quantum Computing and its Supremacy

Using the ideas of quantum physics, quantum computing is an evolution in computational power. Unlike classical computers that rely on bits, quantum computers employ qubits. These qubits leverage superposition, enabling them to exist in a state of both 0 and 1 simultaneously. Furthermore, the entanglement of qubits allows for interconnected states, where the state of one qubit directly influences others. This interconnectedness empowers quantum computers to process and analyze complex datasets with unparalleled efficiency, surpassing the capabilities of classical computers.

Operating on the foundational principles of quantum mechanics, which explain how matter and energy behave at the atomic and subatomic levels, quantum computing offers unprecedented computational advantages. These advantages are particularly evident in tasks like optimising complex systems, simulating quantum systems, and factoring big numbers – all of which pose significant challenges for classical computing.

The transformative potential of quantum computing spans numerous fields. In cryptography, quantum computers hold the potential to break existing encryption algorithms, while simultaneously enabling the development of new, more secure cryptographic systems. In drug discovery, they could accelerate the process by accurately simulating molecular interactions. Additionally, fields like logistics and finance could benefit from quantum computing's optimization capabilities.

However, quantum computing is not without its challenges. The development of stable, error-resistant qubits remains a significant hurdle. Current quantum computers require extremely low operating temperatures and are prone to errors caused by factors such as quantum noise and decoherence.

Despite these challenges, continuous research and development efforts are creating the foundation for advancements in qubit technology, error correction techniques, and quantum algorithms. While the timeline for fully realized quantum computers remains uncertain, the trajectory of progress points towards a future where quantum mechanics will revolutionize practical applications across various industries. Quantum computing signifies a monumental leap in computing, poised to reshape technology and information science by tackling challenges that are currently insurmountable for classical computers.

1.2 Quantum Mechanics: A Foundational Theory

Quantum mechanics is axiomatically a foundational theory of physics that describes the physical properties of matter and light at the atomic and subatomic levels. It differs from classical mechanics as it quantizes certain physical quantities, such as energy and momentum, to have discrete spectra. Quantum mechanics also attributes both particle and wave-like natures to matter and light. Due to Heisenberg's Uncertainty Principle, there are limits to the accuracy of predictions. The theory is considered fundamental, as according to the Correspondence Principle, classical mechanics is a special case of quantum mechanics [21]. The conclusions of this theory often defy common sense, with events improbable in the macroscopic world holding a possibility in the microscopic realm. However, these phenomena are represented through mathematical models that structure the theory. These models include Schrödinger's equation, Heisenberg's Uncertainty Principle, entanglement, and laws of Symmetry and Conservation.

1.2.1 Quantum States, Hilbert Space, and Schrödinger's Equation

Quantum states of a system are points in the Hilbert space. Hilbert space is a complete infinite-dimensional complex vector space equipped with the inner product operation. It is complete in terms of the completeness definition: a Cauchy sequence of vectors always converges to a well-defined limit within this space—necessary to describe continuous quantities. It is a complex vector space because it is over the field of complex numbers, providing ”a linear representation of the rotation group in an n-dimensional space, with each vector having 2^n components, where $n = 2\nu + 1$ or $n = 2\nu$” [22]. Thus, in a space with dimensions 2ν or $2\nu + 1$, a complex vector can be represented as a vector of 2^ν complex numbers, necessary for describing the quantum world. Additionally, it contains the inner product, which is connected to the concept of magnitude (norm) and generalizes the concept of the angle between two elements of the vector space. This equipment makes the space measurable. For each physical system, there is a square-integrable function $\psi(r, t)$, the wave function, belonging to a Hilbert space, which contains all the information that can be extracted about the system. The Copenhagen interpretation states that the probability of finding a particle in space is the square of its wave function's magnitude. The time evolution of it, $\psi(t)$, is governed by the time-dependent Schrödinger equation:

$$\frac{d}{dt}\psi(t) = -i\hat{H}\psi(t)$$

where $\hat{H}$ is a Hermitian operator representing the system's Hamiltonian. Generally, an appropriate Hermitian operator $\hat{A}$ is associated with every physical quantity, whose eigenvalues are real numbers and are the only possible outcomes of a measurement. The physical quantity corresponding to the Hamiltonian is energy.

1.3 Quantum Information and Computation

Suppose we want to communicate an idea; we could achieve this by sending a letter, a drawing, a song, or even Morse code signals. The common element in these communication methods is the language, each time represented differently. Therefore, we can define a scale of the quantity of information contained in each communication method, which corresponds to the ideas we can share using given signals from each method. For this reason, we define bits as the elementary unit of information responding to ”yes or no?” questions. The more independent ”yes or no?” questions a received message can answer, the more information it contains. In the classical world where events are deterministic, the answer is absolute - it will be either ”yes” or ”no” with complete certainty. For this world, we use Boolean algebra (mathematical system of binary logic) in computer language, where every signal, whether input or output, is represented by 0 or 1, each holding 1 bit of information corresponding to ”yes” or ”no.”

1.3.1 Quantum World and Two-State Systems

In the world dominated by quantum phenomena, within a two-state system (yes or no, up or down...), the answer is not absolute, and the system dwells simultaneously in both states with a certain probability to observe each after measurement. The unit of information in quantum information is the qubit, where each state is a superposition of, mutually orthogonal states $|0\rangle$ and $|1\rangle$ (representing 0 and 1 states in Boolean algebra respectively). Therefore, these states are represented as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

such that $|\alpha|^2 + |\beta|^2 = 1$.

These states are points on the surface of a unit sphere, known as the Bloch sphere. By analogy, and with the notation where θ is the polar angle and ϕ is the azimuthal angle, we parametrize the amplitude probabilities α and β as follows:

$$\alpha = \cos(\theta/2), \qquad \beta = e^{i\phi}\sin(\theta/2)$$

Thus, the state can be written as:

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle = (\cos(\theta/2))\left(e^{i\phi}\sin(\theta/2)\right)$$

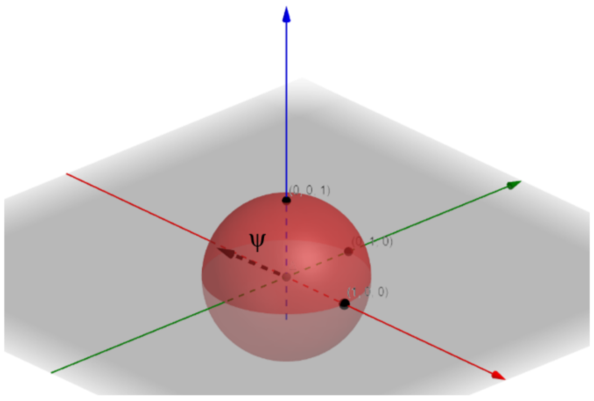


Figure 1.1: The Bloch Sphere

1.3.2 Quantum Gates and Quantum Information Processing

A measurable quantity in quantum mechanics is represented by an operator - let’s denote it as A. After acting on a state, it will yield a new one. To yield another state in the vector space of the Bloch sphere, it needs to be unitary (causing unitary transformations of states). Thus, for the transformation of one or more qubits, we have Quantum Gates, analogous to the Logic Gates in Classical Information Theory.

Some examples of one-qubit transformations are:

- Pauli transformations: $\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\hat{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $\hat{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ (phase rotation)

- Hadamard gate: $\hat{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
- Phase shift gate: $S(\theta) = \begin{pmatrix}1 & 0\\ 0 & e^{i\theta}\end{pmatrix}$
- $\pi/8$ gate (or T gate): $T = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}$

Some gates acting on multiple qubits include:

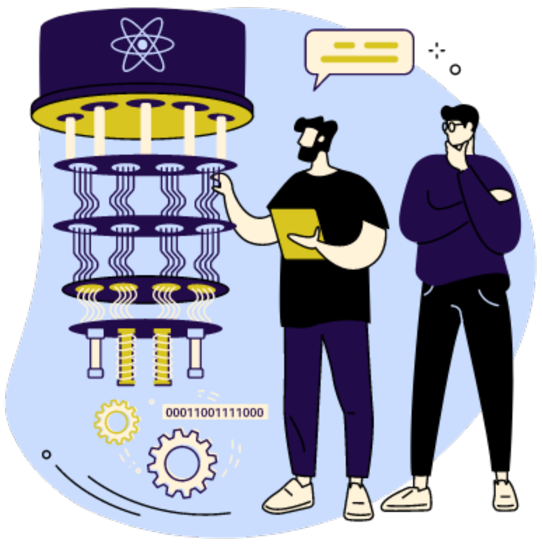
- Controlled-NOT (or CNOT) for 2 qubits: $\text{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$
- Swap gate for 2 qubits: $\text{SWAP} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$
- Toffoli gate for 3 qubits: 8×8 matrix with 1s at the diagonal elements and zeroes the others, everywhere except the right-hand down side where we have the elements of CNOT matrix.

One potential exponential speedup that quantum computers promise lies in executing linear algebra computations. The inherent ability of quantum computers to execute these computations rapidly is theoretically sound; however, practical implementations face challenges like data encoding. Quantum devices would need efficient methods to load large matrices to perform such tasks seamlessly. Thus, in practical use, quantum gates —interpreted mathematically as linear layers— would be beneficial for complex transformations.

1.4 What is Quantum Machine Learning?

Quantum machine learning is the mix of quantum algorithms and machine learning programs. It extends the hardware of machine learning using the quantum computer - a new type of computer device. This type of computers processes the information using the quantum theory, where the information is encoded in the state of a quantum system. In order to extract this information, they use quantum information processing techniques, performing logical operations. The basic unit of quantum information is the qubit - just like bit is the basic unit of classical information. Both classical and quantum information can be processed and analyzed using computers and mathematics, can be manipulated using algorithms, can be transmitted through space and be measured using Von Neumann entropy.

While quantum computers hold the potential for significant speedups over classical algorithms, achieving this advantage hinges on overcoming several key challenges. One crucial aspect is managing matrix sparsity and conditioning, ensuring the efficient representation and manipulation of data within quantum algorithms. Additionally, leveraging advanced quantum hardware like Quantum Random Access Memory (QRAM) is essential for handling and accessing data effectively.



The realization of ”fault-tolerant QPUs,” or ideal, universal quantum computers capable of tackling large-scale, real-world problems, necessitates millions of physical qubits to produce hundreds or thousands of error-resistant ”logical qubits.” This requirement stems from the need for robust error correction methods, which rely on encoding information across multiple physical qubits to create more stable logical qubits. This demand for vast qubit resources has led researchers to explore algorithms tailored for the current landscape of quantum computing.

The ”noisy intermediate-scale quantum” (NISQ) era describes the present state of quantum devices, characterized by noisy qubits lacking error correction. This limitation restricts their computational power. NISQ devices often employ hybrid architectures, where quantum processors handle specific tasks, while classical processors manage others. This hybrid approach, often accessible through cloud-based services, allows for greater flexibility and resource utilization. While NISQ devices may not yet be capable of realizing the full potential envisioned for quantum computing, ongoing research indicates their suitability for specific, specialized tasks.

In their current form, quantum technologies share similarities with Application-Specific Integrated Circuits (ASICs), designed for specific purposes, rather than general-purpose CPUs. While they currently excel at implementing a limited set of quantum algorithms, their continued development and integration could position them as a valuable component within the broader landscape of AI hardware.

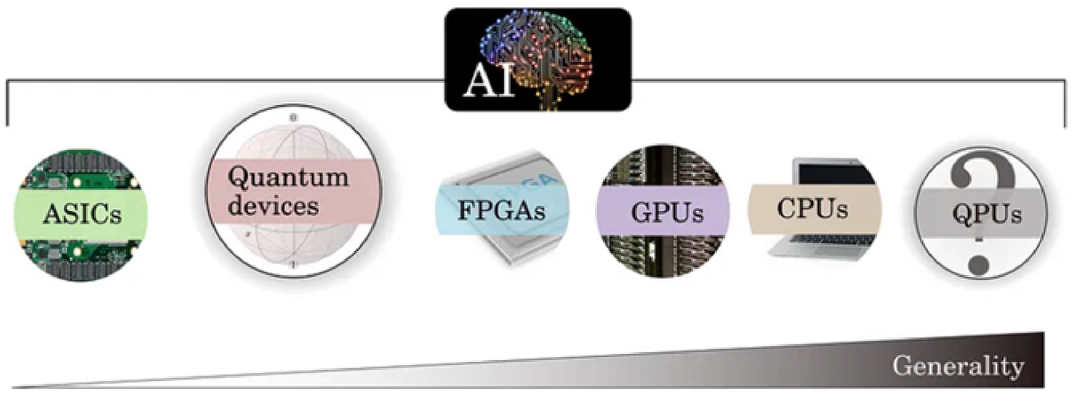
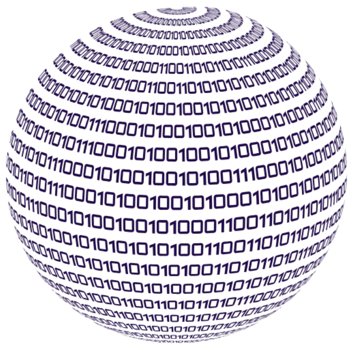


Figure 1.2: Generality of hardware platforms - retrieved from medium.com/xanaduai



Chapter 2

Variational Quantum Algorithms

Variational Quantum Algorithms (VQAs) represent a powerful class of hybrid quantum-classical algorithms (models as shown in Figure 2.1), designed for use on both near-term and future fault-tolerant quantum computers. These algorithms harness the principles of quantum mechanics to explore vast computational spaces, while relying on classical optimization techniques to guide the search for optimal solutions. This allows VQAs to tackle complex problems that are intractable for classical computers alone.

A key motivation behind VQAs is the constraint posed by current and near-term quantum computers, which are limited in both qubit count and coherence times. VQAs address these limitations by employing relatively short quantum circuits, known as ansatz, which are less susceptible to errors caused by noise and decoherence. These ansatz circuits are parameterized, meaning their behavior can be adjusted by modifying a set of control parameters.

At the heart of VQAs lies the concept of a variational quantum circuit, this parameterized quantum circuit whose parameters can be adjusted to modify the output state. This circuit is designed to encode a trial solution to the problem at hand. By iteratively adjusting the circuit parameters and evaluating the resulting quantum state on a quantum computer, VQAs effectively explore a vast space of potential solutions. A classical optimizer then uses the measurement data from the quantum computer to guide the search for optimal parameter values.

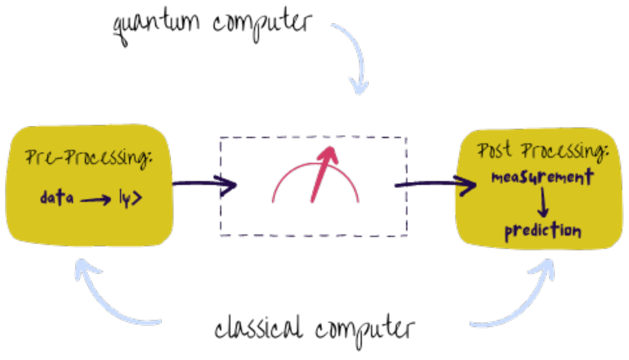
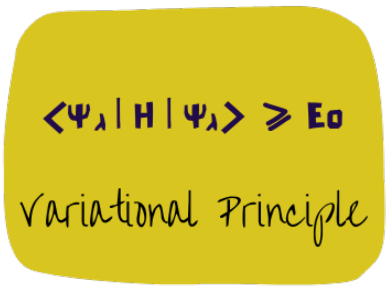


Figure 2.1: Hybrid quantum-classical machine learning procedure

2.1 Variational Quantum Eigensolver

In order to tackle optimization problems that are exponentially hard to solve on classical computers - but also run the algorithms in the near-term devices - Variational Quantum Eigensolver (VQE) was introduced. VQE is a member of VQAs family and uses quantum circuits to run subroutines of a larger algorithm. It works with short-depth circuits and will approximate results when the *quantum volume* [1] (the power of the quantum computer) is large enough.



$\langle \Psi_\lambda | H | \Psi_\lambda \rangle \geqslant E_0$
Variational Principle

Figure 2.2: Variational Principle

VQE aims to find an upper bound of the lowest eigenvalue of a given Hamiltonian. Inspired by the **Variational Principle** (Fig. 2.2) it calculates approximate solutions for a given problem. Seeking to identify an upper bound as near to the ground state energy as feasible, it minimizes the expectation value of H. For this, it fluctuates throughout all possible normalized states so that they're inherently an eigenstate of H too. A problem though - is that varying over the entire Hilbert space is quite complex to realize for physical calculations. In order to get a practical upper bound, ansatz solves the problem by elegantly taking into account a suitable subset of the Hilbert Space in order to select the state that produces the lowest expectation value.

Since VQE belongs to the general category of VQAs, it is described by:

- **Problem Initialization**
- **Circuit Ansatz**
- **Cost Function**
- **Training Procedure**

2.1.1 Problem Initialization

Quantum computer is initialized in a default state - typically the ground state $|00...0\rangle$. This state is then evolved using a non-parameterized unitary operator U_r and the reference state $|R\rangle$ is obtained. The selection of U_r is strategic, ensuring that $|R\rangle$ encapsulates properties relevant to the problem domain, which for this thesis involves protein folding.

2.1.2 Circuit Ansatz

The ansatz (Fig 2.3), or the trial wavefunction for VQE, is represented by a parameterized unitary transformation $U(\theta)$. This transformation is structured as a product of L unitary operations, $U_i(\theta_i)$, each dependent on a subset of the parameters θ. These unitaries are applied sequentially to the input state. The decomposition of each $U_i(\theta_i)$ into a combination of parameterized and fixed quantum gates allows for the construction of a flexible and tunable model. This modular approach enables the exploration of a vast space of possible quantum states, adapting and refining the circuit architecture to effectively address the intricacies of protein structures in our study.

[1] This measure quantifies the greatest random circuit, that the computer can successfully create and that is equal in width and depth. Large quantum volume is equivalent to a large number of qubits, their coherence and the connectivity, and is independent of the hardware.

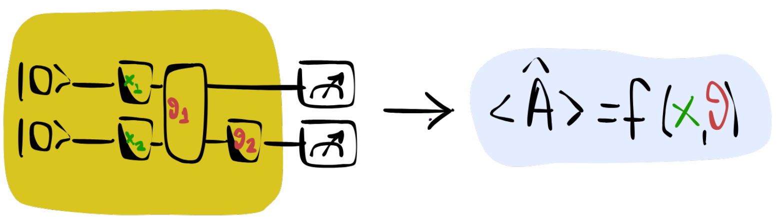


Figure 2.3: The circuit-ansatz of a variational quantum circuit

2.1.3 Training Procedure

We can train parameterized quantum circuit models to perform data-driven tasks. The mathematical expression for the task of learning any function from data is the minimization of a cost, reflecting the performance of the quantum model in solving the targeted problem (Fig 2.4). More specifically, we must assign scores to each parameter based on a set of criteria in order to compare them. The function that assigns a score to our parameters is known as the "cost" or "loss" function - also known as the objective function, with respect to the parameter vector.
Training in a VQE framework involves the iterative adjustment of the parameters θ. During each iteration, quantum measurements are performed, and the resulting data are used to evaluate the cost function. The gradient or derivative of the cost with respect to the parameters θ is often computed to direct the optimization process, enhancing the algorithm's efficiency in identifying the optimal parameter set that minimizes the cost function, hence solving the problem.

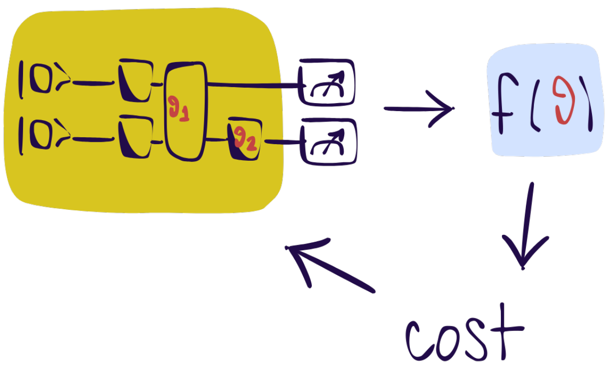


Figure 2.4: The training procedure of a hybrid quantum-classical machine learning model

2.1.4 Cost Function

The cost function is tailored to reflect the specific nuances of the problem being addressed, which in the context of this thesis involves the protein folding problem. It is designed as the expectation value of a cost Hamiltonian H_C (Fig 2.5), which encodes the protein folding landscape. This Hamiltonian is strategically formulated to include terms that represent interactions and constraints critical to protein folding. The optimization process seeks to find the parameter set θ that minimizes this expectation value, thus identifying the quantum state that best represents the lowest energy state of the protein – analogous to its most stable folded structure.

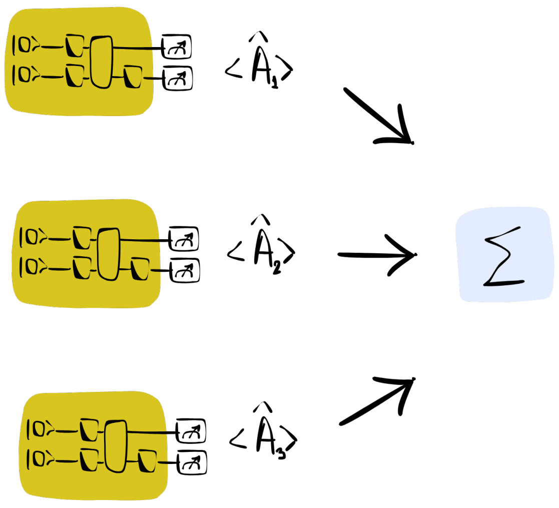

$\langle\hat{A}_1\rangle$
$\langle\hat{A}_2\rangle$
$\langle\hat{A}_3\rangle$
$\sum$

Figure 2.5: The cost function of a hybrid quantum-classical machine learning algorithm.

2.2 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is another hybrid quantum-classical algorithm from the VQAs family, designed to solve combinatorial optimization problems. Thus, as the rest members of the family, it operates by creating a parameterized quantum state and iteratively refining these parameters to minimize a given cost function. What distinguishes QAOA from other VQAs is that it employs a structured ansatz inspired by adiabatic quantum computing, where a system is slowly evolved from an initial ground state to the ground state of a Hamiltonian encoding the optimization problem. This adiabatic evolution is approximated in QAOA by a sequence of alternating layers of two specific types of quantum operations:

- Cost Hamiltonian Operations: These operations depend on the specific problem being solved and are designed to encode the cost function as a quantum operator.
- Mixing Hamiltonian Operations: These operations are typically chosen to be problem-independent and help to explore the solution space more effectively.

We will now go into a little more details for the mathematics of the QAOA algorithm, so that we fully understand the VQAs in general and how we will construct such circuits.

The algorithm begins by initializing a quantum system in a superposition of all possible states - this is typically achieved by applying a Hadamard gate to each qubit in the system, creating an equal superposition of all computational basis states. The algorithm then applies a sequence of unitary transformations to the system, which are determined by two sets of parameters, denoted as β and γ. These transformations are designed to steer the system towards the ground state of a problem Hamiltonian, which represents the optimal solution to the optimization problem.

As we mentioned before, the unitary transformations consist of two types of operations: the mixing operator $U_M(\beta)$ and the problem operator $U_C(\gamma)$. The mixing operator, typically a rotation around the X-axis, drives transitions between different states in the superposition, thereby exploring the solution space. The problem operator, a rotation around the Z-axis, rewards or penalizes states based on their cost function value, guiding the system towards lower energy states.

The parameters β and γ are initially chosen at random and then iteratively optimized using a classical optimization algorithm to minimize the expectation value of the cost function. This hybrid approach leverages the quantum system's ability to explore the solution space efficiently and the classical system's ability to perform robust optimization.

In essence, QAOA can be seen as a specific instance of VQE where the ansatz has a particular structure motivated by the problem of combinatorial optimization. Both algorithms follow the paradigm of the variational quantum algorithm, which leverages the strengths of both quantum and classical computation to solve complex problems.

The QAOA may be used to solve quadratic unconstrained binary optimization problems (QUBOs)

- Quadratic objective function
- No variable constraints
- Binary optimization variables

$$\text{minimize:} \qquad x^T Q x + c^T x$$

$$x \in \{0,1\}^n \quad Q \in \mathbb{R}^{n\times n}, c \in \mathbb{R}^n$$

In the context of protein folding, the QAOA can also be combined with other terms, like counter-diabatic (CD) driving terms, to enhance its performance. CD driving is a technique from adiabatic quantum computation that accelerates the adiabatic evolution of a system, enabling it to reach its ground state more quickly. In an ideal adiabatic quantum process, the system remains in its instantaneous ground state throughout the evolution. However, in real-world quantum systems, various factors can cause the system to deviate from this ideal adiabatic path. These factors can include the speed of the evolution (if it's too fast, the system may not be able to keep up and remain in its ground state), as well as external disturbances such as decoherence and noise. Such additional terms are added to the system's Hamiltonian to counteract these deviations and help maintain the adiabaticity of the evolution.

2.2.1 QAOA circuit

The goal of the QAOA circuit is to efficiently find a quantum representation of a classical cost function **C(x)** through a Hamiltonian operator H_C. The encoding ensures that the eigenstate $|x\rangle$ has corresponding eigenvalue $C(x)$ which directly represents its cost:

$$H_C |x\rangle = C(x) |x\rangle$$

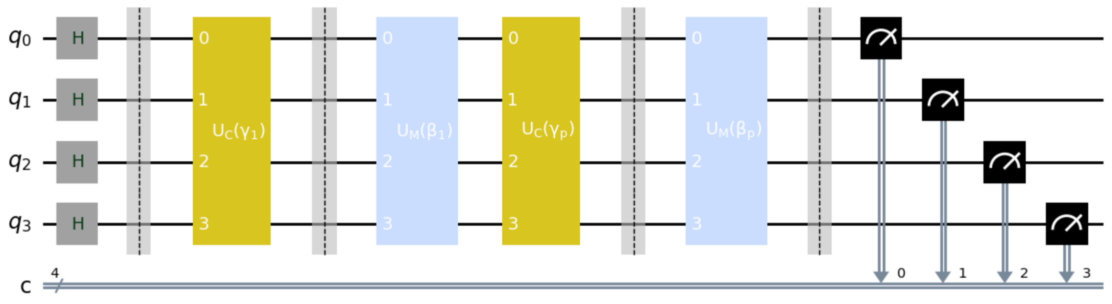
A standard formulation in combinatorial optimization is the QUBO model, where the cost function $C(x)$ is expressed as a quadratic polynomial over binary variables:

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i=1}^{n} c_i x_i$$

Here, x_i are binary decision variables, Q_{ij} represents the interaction between variables x_i and x_j, and c_i is the linear coefficient for x_i.

The corresponding Hamiltonian operator H_C for the QUBO is constructed using Pauli Z matrices, reflecting the binary nature of x_i:

$$H_C = \sum_{i,j=1}^{n} \frac{1}{4} Q_{ij} Z_i Z_j - \frac{1}{2} \sum_{i=1}^{n} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) Z_i + \sum_{i,j=1}^{n} \frac{Q_{ij}}{4} \sum_{i,j=1}^{n} \frac{c_i}{2}$$



p repetitions of alternating cost and mixing layers

Preparation of equal superposition state $|+\rangle^n = \sum_{x \in \{\mathbf{0},\mathbf{1}\}^n} \frac{|x\rangle}{\sqrt{\mathbf{2}^n}}$

$U_C(\gamma_i) = e^{-i\gamma_i \mathrm{H_C}}$
cost layers
$U_M(\beta_i) = e^{-i\beta_i \mathrm{H_M}}$
mixer layers

Measurement in computational basis

As shown in the above circuit, we started the implementation of QAOA by preparing a superposition of all possible binary states $|x\rangle$, creating a uniform superposition $|\psi\rangle$ to allow the algorithm to explore all computational paths simultaneously. Then, the typical QAOA algorithm involves an alternation between applying phase shifters and mixers for p iterations. The phase shift operation, $U(\gamma) = e^{-i\gamma H_C}$, adjusts the phases of the components of the state $|\psi\rangle$ based on their respective costs, amplifying the probability amplitudes of lower-cost states in the superposition:

$$|\psi(\gamma)\rangle \equiv U(\gamma)|\psi\rangle = \sum_x e^{-i\gamma E_x} \psi_x |x\rangle.$$

This transformation is crucial as it phases the states, biasing the system towards low-cost configurations when measured.

However, maintaining a closed system ensures the conservation of the quantum system's total energy. When the Hamiltonian H_C operates within this closed system, the expectation of the Hamiltonian does not change, even under unitary transformations applied by the QAOA algorithm:

$$\langle \psi(\gamma) | \mathrm{H_C} | \psi(\gamma) \rangle = \left\langle \psi(\gamma) \middle| U^\dagger(\gamma) \mathrm{H_C} U(\gamma) \middle| \psi(\gamma) \right\rangle = \langle \psi | \mathrm{H_C} | \psi \rangle$$

To make it possible to change the expected value, we also add the mixed layers

$$U_M(\beta_i) = e^{-i\beta_i \mathrm{H_M}},$$

as is shown in the circuit. The form of H_M depends on the specific optimization problem being addressed, but its main purpose is to break the symmetries in the search space and promote state manipulation by introducing off-diagonal elements in the density matrix. In general, it can be written as a linear combination of Pauli-X gates acting on the qubits in the computational basis. The coefficients of the linear combination are determined by the optimization problem at hand and are typically chosen to maximize the fidelity of the algorithm.

Matrix Exponentials

The matrix exponential component $e^M = \sum_{k=0}^{\infty} \frac{M^k}{k!}$ plays a significant role in calculating the unitary transformations. These transformations are effectively applied using exponential functions of Pauli matrices to instantiate the mixing and cost layers within the QAOA circuit. The Pauli rotations $R_X(2\theta)$, $R_Y(2\theta)$, and $R_Z(2\theta)$ are derived from these matrix exponentials, applied for qubit rotations conditional on the quantum state preparation:

$$
\begin{aligned}
e^{-i\theta X} &= R_X(2\theta) = \begin{pmatrix} \cos(\theta) & -i\sin(\theta) \\ -i\sin(\theta) & \cos(\theta) \end{pmatrix} \\
e^{-i\theta Y} &= R_Y(2\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \\
e^{-i\theta Z} &= R_Z(2\theta) = \begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix}
\end{aligned}
$$

Pauli matrices

$$
X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
$$

Implementation of the repetitive layers

During the operation of the QAOA circuit, the application of cost and mixing layers are iteratively performed, modulated by these transformations:

The Cost Layers

$$
H_C = \sum_{i,j=1}^{n} \frac{1}{4} Q_{ij} Z_i Z_j - \frac{1}{2} \sum_{i=1}^{n} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) Z_i + \sum_{i,j=1}^{n} \frac{Q_{ij}}{4} \sum_{i,j=1}^{n} \frac{c_i}{2}
$$

The operation of the cost layer involves phase shifts dictated by $e^{-i\gamma H_C}$, which effectively applies a phase rotation on the qubits based on their interactions and individual biases:

$$
e^{-i\gamma H_C} = \prod_{i,j=1}^{n} R_{Z_i Z_j} \left(\frac{1}{4} Q_{ij} \gamma \right) \prod_{i=1}^{n} R_{Z_i} \left(\frac{1}{2} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) \gamma \right)
$$

This adjusts the amplitudes of the states in direct correlation to their associated costs in the classical problem. The circuit then is implemented as shown in Fig 2.6:

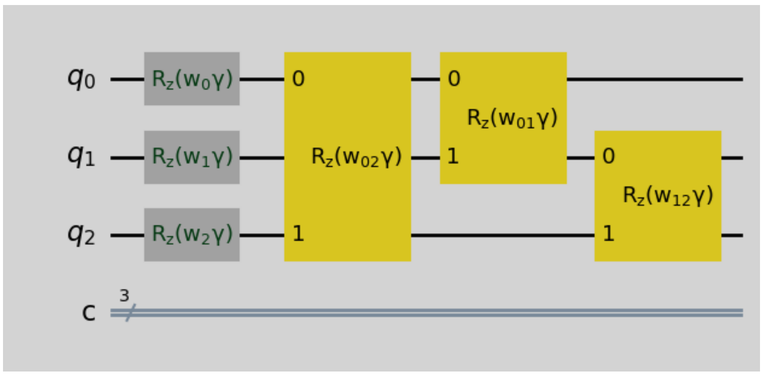


Figure 2.6: The Implementation of Cost Layers

The Mixing Layers

$$H_M = \sum_{i=1}^{n} X_i$$

and the evolution:

$$e^{-i\beta H_M} = \prod_{i=1}^{n} R_{X_i}(2\beta)$$

is implemented as shown in Fig 2.7:

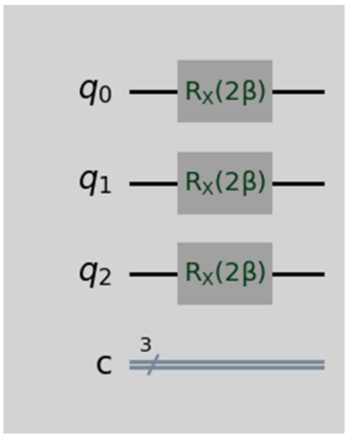


Figure 2.7: The Implementation of Mixer Layers

The Trotter Suzuki Formula

In quantum mechanics, the Trotter-Suzuki decomposition is a fundamental technique used to approximate the exponential of the sum of non-commuting operators. This has profound importance in quantum simulations, particularly in approximating the time evolution operator. Given a Hamiltonian H that can be decomposed into a sum of terms $H = H_1 + H_2$, the Trotter-Suzuki formula allows for an efficient approximation of the time evolution operator e^{-iHt}.

The approximation is represented as:

$$e^{-it(H_1+H_2)} \approx \left(e^{-i\frac{H_1 t}{r}} e^{-i\frac{H_2 t}{r}}\right)^r$$

where t is the total time, and r is the number of time steps into which t is divided

Adiabatic Theorem and QAOA

The adiabatic theorem (Fig 2.8) in quantum mechanics states that if the Hamiltonian of a quantum system in its ground state is perturbed slowly enough, the system remains in its ground state. This principle can be connected to the Quantum Approximate Optimization Algorithm (QAOA), which aims to find a good ansatz for a given problem Hamiltonian with high accuracy.

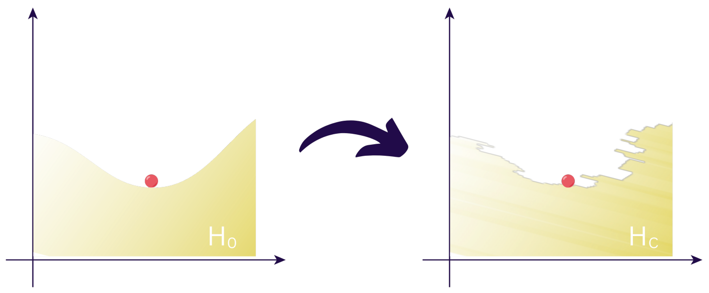


Figure 2.8: The Adiabatic Theorem

In the context of QAOA, by approximating the evolution operator as:

$$e^{-itH} \approx \left(e^{-i\frac{H_1 t}{r}} e^{-i\frac{H_2 t}{r}} \right)^r ,$$

the algorithm can efficiently simulate the evolution under a gradually changing Hamiltonian, ensuring that the system closely follows its instantaneous ground state. For the QAOA the r represents the number of alternating sets of cost and mixer layers.

2.3 Metrics to Compare Ansatzes

In order to compare different ansatzes we use two metrics; the "Expressibility" and the "Entangled Capability" which try to capture the generality of the quantum hardware platforms. [26]

2.3.1 Expressibility

Expressibility within the context of parameterized quantum circuits (PQCs) is a crucial descriptor that essentially gauges a circuit's ability to thoroughly explore the state space associated with quantum computations. Formally defined, expressibility measures a circuit's capacity to generate a set of states that are representative of larger, often complex, Hilbert spaces. This is particularly pertinent in the realm of quantum computing, where the ability to span an expansive range of quantum states correlates directly with a quantum circuit's utility and effectiveness in performing diverse computational tasks.

Analytically, the concept of expressibility can be quantified by comparing the distribution of output states generated from a PQC, under varied parameter settings, against the idealized uniform distribution of quantum states known as the Haar measure. This comparison involves statistical methods where one often employs the Hilbert-Schmidt norm to assess the extent to which the states produced by a PQC deviate from an approximate uniform (Haar) distribution. The lesser the deviation, the higher the expressibility of the circuit, indicating a greater capability of the circuit to mimic the desired Haar distribution of states across the Hilbert space.

2.3.2 Entanglement Capability

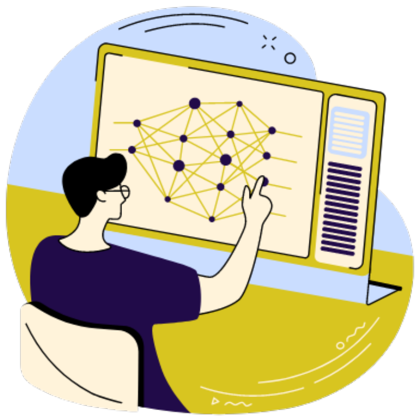
In contrast, entangling capability is another fundamental aspect that defines the potential of PQCs to induce entanglement among qubits through their operational dynamics. Entanglement is a quantum phenomenon that allows particles to become interconnected such that that, regardless of their distance from one another, the state of one particle instantly affects the state of another. In quantum circuits, the ability to robustly create and control entanglement is vital for leveraging quantum mechanical properties that underpin quantum computing.

Scientifically, the entangling capability of a quantum circuit can be assessed by the types and configurations of two-qubit or multi-qubit gates employed within the circuit. These gates are instrumental in manipulating the quantum states in a manner that fosters entanglement. For instance, circuits that deploy sequences of controlled rotation gates might exhibit different entangling characteristics compared to those that utilize simpler two-qubit operations. This capability is quantitatively estimated through simulations that evaluate how effectively these gates generate and manage entanglement across the circuit. The geometric or topological arrangement of qubits and the choice of gates significantly influence this capability, thus shaping the overall performance of the quantum circuit in computational tasks requiring intricate and high-degree entanglement.

In sum, understanding and optimizing both expressibility and entangling capability are paramount in designing PQCs that are not only powerful and versatile but also tailored for specific quantum computing applications. These metrics collectively inform the suitability of a circuit design in achieving the desired computational goals while adhering to operational constraints such as circuit depth and coherence times.

Chapter 3

Classical Optimizers



This chapter delves into the intricacies of classical optimization techniques employed in hybrid quantum-classical algorithms, with a particular focus on their applicability to the protein folding problem. By thoroughly examining the strengths and limitations of each optimizer, particularly in the presence of noise and complex energy landscapes, this analysis aims to identify the most suitable optimization strategy for our specific protein folding model. The insights gained from this comparative assessment will not only inform the optimization choices made within this work (Section 6.4), but also provide valuable guidance for future implementations and advancements in the field of quantum-assisted protein structure prediction.

In the complex landscape of protein folding simulations, two primary classes of optimization algorithms play pivotal roles in refining the prediction models: gradient-based and gradient-free optimizers. Gradient-based optimizers, such as Adam and Adagrad, leverage the gradient of the cost function to guide the search for an optimal set of parameters. These algorithms calculate the gradient at each iteration, which indicates the direction and magnitude of the step to take towards minimizing the cost function. The Adam optimizer, for instance, is highly favored for its adaptive learning rate management, which adjusts the update steps for each parameter, enhancing the convergence speed and stability.

On the other hand, gradient-free optimizers such as COBYLA (Constrained Optimization BY Linear Approximations) and Powell's method, operate without the need for gradient information. These algorithms are particularly useful in scenarios where the cost function does not have a well-defined gradient, or when the gradient computation is computationally prohibitive. COBYLA, for example, uses linear approximations to handle constraints, making it suitable for problems with complex landscapes and multiple constraints. Powell's method, focusing on a direction set-based search, circumvents the need for gradient calculations altogether, which can be advantageous in non-smooth optimization problems.

The objective in both approaches is to minimize the cost function C, which serves as a measure of the quality of a given solution or parameter set in the context of the protein folding problem. The process begins with an initial parameter set, from which the cost function is calculated. Based on this evaluation, parameters are adjusted according to the optimization algorithm's strategy—either moving along the gradient or exploring the parameter space without it. This iterative adjustment continues until the cost function reaches a satisfactorily minimized state, indicating an optimal or near-optimal solution to the protein folding configuration.

3.1 Gradient-Based Optimizers

3.1.1 Standard Gradient Descent

A basic optimisation technique called gradient descent modifies parameters iteratively in order to minimise the cost function. The cost function's negative gradient denotes the direction of the steepest descent, which is where you are supposed to be moving. Traditional gradient descent, however, can run into problems like delayed convergence and getting trapped in local minima.

3.1.2 Momentum

Momentum incorporates the concept of inertia into gradient descent, helping to accelerate convergence and reduce oscillations:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla C_t \tag{3.1.1}$$

$$\theta_{t+1} = \theta_t - \alpha v_t \tag{3.1.2}$$

Here, v_t represents the velocity update, β is the momentum factor, and θ_t are the parameters at iteration t. The momentum term ensures that the algorithm maintains its direction through small oscillations, leading to faster convergence. The momentum parameter β typically has a value of 0.9, but can be adjusted as long as $\beta \in [0, 1)$. A higher β places greater importance on past gradients, whereas $\beta = 0$ reverts to the standard gradient descent (SGD) method.

The incorporation of momentum can be likened to a ball rolling down a hill. The ball's momentum will continuously increase as its velocity rises until air resistance halts further acceleration at terminal velocity, where the "air resistance" is represented by the parameter β. As the gradient converges towards a minimum, previous gradients aligned in the same direction will accumulate momentum, while gradients in other directions will diminish. This results in oscillations in non-minimum directions being dampened, and momentum propelling us more rapidly towards the minimum.

3.1.3 Adagrad

Adagrad is the first of the so called "adaptive learning" algorithms. It adapts the learning rate for each parameter based on historical gradient information, thus adjusting the learning rate dynamically:

$$r_t = r_{t-1} + \nabla C_t^2 \tag{3.1.3}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{r_t} + \epsilon}\nabla C_t \tag{3.1.4}$$

r_t accumulates the sum of the squared gradients, and ϵ is a small constant for numerical stability (to ensure we do not divide by zero). This method decays the learning rate over time, making it particularly beneficial for problems involving sparse data.

The spaces over which we are optimizing are high-dimensional spaces. It is feasible to optimize in one direction of the space while failing to optimize in another. If the learning rate only decreases over time, such as $r_t = \frac{r_0}{\sqrt{t+\epsilon}}$, we might successfully optimize a specific parameter w_1 but fail to optimize another parameter w_2; the step size might become too small to continue effective minimization in the direction of w_2. Therefore, it is insufficient to merely decrease the learning rate, α, over time. It must also be adapted individually for each parameter. This is achieved by dividing the learning rate by the square root of the past gradients associated with that parameter. If the gradients for a particular parameter, w_1, are consistently large, a slightly smaller learning rate is desirable (to prevent oscillations in valleys). Conversely, if the gradients for another parameter, w_2, are consistently small, dividing by them will result in a larger learning rate, ensuring that we can move more swiftly along a shallow decline.

A challenge with AdaGrad is that the accumulated squared gradients, r_t, continuously increase in size, which eventually reduces the learning rate, α, to a very small value after numerous iterations. Consequently, AdaGrad is seldom used, as there are now more advanced algorithms that have improved upon it.

3.1.4 RMSprop

RMSprop, or Root Mean Square Propagation, enhances Adagrad by introducing a decay term to control the growth of the denominator:

$$r_t = \beta r_{t-1} + (1-\beta)\nabla C_t^2 \tag{3.1.5}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{r_t}+\epsilon}\nabla C_t \tag{3.1.6}$$

This optimizer maintains an exponentially decaying average of past squared gradients, ensuring stabilized learning rates and addressing Adagrad's diminishing learning rate issue over time. This is done by dividing the learning rate by the square root of an exponentially weighted moving average (EWMA) of past squared gradients instead of an accumulating sum of gradients.

3.1.5 Adam

The Adam optimizer, short for Adaptive Moment Estimation, combines the benefits of Momentum and RMSprop by maintaining both first (mean) and second (uncentered variance) moment estimates of the gradients.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla C_t \tag{3.1.7}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla C_t^2 \tag{3.1.8}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3.1.9}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.1.10}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \tag{3.1.11}$$

Adam adaptively adjusts learning rates based on both the first moment (mean) and second moment (variance) of the gradients. It uses bias corrections to the estimates of both terms to account for their initialisation at the origin, i.e. $m_0 = 0$ and $v_0 = 0$. This adaptive adjustment significantly stabilizes and accelerates the convergence process, making Adam a preferred choice in many machine learning applications.

Also, note that the ϵ term in the denominator is placed outside the square root in this instance. Although positioning it inside the square root is also feasible, practical experience has shown slightly better performance when it is placed outside.

3.2 Visual Comparisons

Visualizing the behavior of these optimizers reveals distinctive patterns in their convergence paths:

- **Momentum** smoothens and accelerates the path by incorporating a velocity term that carries through iterations. It shows quicker descent along gradient paths.
- **Adagrad** modifies learning rates for each parameter based on their gradient history, ideal for paths involving more frequent updates. It performs gradual downsizing of steps, which is advantageous for sparse features.
- **RMSprop** prevents gradients from exploding or vanishing through an exponential decay factor. It effectively balances learning rates, resulting in stabilized traversal.
- **Adam** combines the benefits of Momentum (inertia-based acceleration) and RMSprop (adaptive learning rates), resulting in an efficient and stable optimization trajectory. It combines stabilization and acceleration features, emonstrating efficient convergence patterns.

3.3 Gradient-Free Optimizers

3.3.1 Constrained Optimization by Linear Approximations (COBYLA)

Constrained Optimization by Linear Approximations (COBYLA) is a gradient-free optimization technique that employs a direct search method developed by Mike J. D. Powell. This optimizer is known for its ability to work with constraints, making it adaptable to wide range of uses - especially in hybrid quantum-classical machine learning.

COBYLA operates through iterative refinement of a simplex—a geometric figure formed by $n + 1$ points in n-dimensional space—that is strategically adjusted to minimize the objective function. Unlike gradient-based methods, COBYLA approximates the objective and constraint functions by linear interpolation, which makes it particularly effective when the gradient information is noisy, unavailable, or computationally expensive to obtain.

The COBYLA algorithm can be mathematically stated as follows:

$$||x - x_o|| \leq \Delta$$

where $F(x_o)$ is the minimum value found so far.
In unconstrained COBYLA, we have

$$x = x_o - \left(\frac{||\Delta \nabla L||}{||\nabla L||}\right) \nabla L.$$

3.3.2 Powell

Mike J. D. Powell introduced an optimization technique that has become a cornerstone for gradient-free optimization—often referred to simply as Powell's method. This method is designed to minimize a function without requiring the computation of gradients, employing a directional search strategy to efficiently explore the parameter space.

Powell's method works by moving iteratively along a set of directions, and at each step, it determines the optimal length to move along these directions by performing line searches. It then updates the direction set based on the attained optimum, ensuring that it converges effectively towards the minimum. The method adapts over iterations to better navigate the topography of the objective function, making it highly effective for complex, high-dimensional, and non-convex optimization problems.

In the domain of hybrid quantum-classical machine learning, Powell's method has proven to be highly effective, especially for variational quantum algorithms. Benchmarking studies indicate that Powell's method consistently performs well across various tasks, including simulation of molecular properties. For instance, it has shown remarkable efficiency in scenarios where the number of parameters is relatively small but the parameter space is sufficiently intricate to disrupt gradient-based methods. This makes Powell's method a strong candidate for optimizing parameterized quantum circuits where quantum noise and hardware imperfections render gradient information unreliable.

The Powell algorithm optimizes a quadratic function:

$$f(x) = x^T Ax - 2b^T x + c$$

where A is positive definite symmetric, $b \in \mathbb{R}^n$.

The theorem states that if $u_1, \ldots, u_m$ is a set of non-zero conjugate directions, then the minimum of $f(x)$ in the space spanned by $u_1, \ldots, u_m$ can be found at the point:

$$\beta_i = \frac{u_i^T b}{u_i^T A u_i}$$

The Powell algorithm can be summarized in the following steps:

1. Let x_0 be the initial point and let $u_1, \ldots, u_n$ be the set of conjugate vectors.
2. For $i = 1, \ldots, n$: - Compute β_i to minimize $f(x_{i-1} + \beta_i u_i)$. - Define $x_i = x_{i-1} + \beta_i u_i$.
3. For $i = 1, \ldots, n-1$: - Replace u_i by u_{i+1}.
4. Replace u_n by $x_n - x_0$.
5. Compute β to minimize $f(x_0 + \beta u_n)$, and replace x_0 by $x_0 + \beta u_n$.
6. Repeat the process until the function value or gradient change is below a set threshold.

3.4 Performance of Optimizers for Applications in Quantum Chemistry with VQAs

The performance of various classical optimizers for variational quantum algorithms (VQAs) was assessed by simulating quantum chemistry applications for simple molecules, namely, Hydrogen (H_2), Lithium Hydride (LiH), Beryllium Hydride (BeH_2), water (H_2O), and Hydrogen Fluoride (HF). The optimizers evaluated include both gradient-based and gradient-free methods. The simulations were carried out using quantum simulators in Qiskit. [35]

A set of key performance indicators were considered for benchmarking:

- Ground state energy errors
- Dissociation energy
- Dipole moment
- Simulation convergence

- Number of iterations to reach convergence

The simulations used the Unitary Coupled Cluster (UCC) ansatz, and the number of qubits varied from two for the Hydrogen molecule to ten for Hydrogen Fluoride.

3.4.1 Gradient-based Optimizers

Among the gradient-based optimizers, the following performed the best:

- Conjugate Gradient (CG)
- Limited-memory Broyden-Fletcher-Goldfarb-Shanno bound (L-BFGS-B)
- Sequential Least Squares Programming (SLSQP)

These, showed minimal errors and consistent performance in molecular simulations.

3.4.2 Gradient-free Optimizers

For gradient-free methods, the most efficient optimizers identified were:

- Constrained Optimization BY Linear Approximations (COBYLA)
- Powell

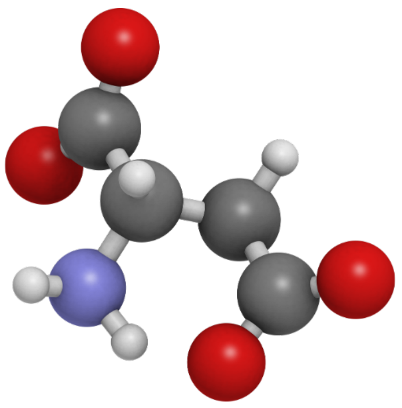
These methods exhibited robust performance, particularly for molecular systems with different electron configurations, bond types, and geometries. They efficiently minimized the error in molecular properties such as dissociation energy and dipole moment.

3.4.3 Overall Insights

From the comprehensive benchmarking, it was evident that:

- Gradient-based optimizers generally provided faster convergence with fewer iterations. In real devices however, they can be more sensitive to noise since they involve calculating gradients, which can be negatively impacted by the noise in NISQ devices.
- Gradient-free optimizers, while sometimes slower, displayed robustness across varied molecular configurations. Moreover, they may be a better option in the NISQ era, due to their robustness and ability to handle noise in quantum circuits.

The above, establish guidelines for selecting appropriate optimizers based on the specific requirements of quantum chemistry simulations using VQAs. Quantum hardware-specific optimizers were also included but need further study to assess their practical effectiveness.



Chapter 4

Protein Folding

In this section, we will discuss the basics of protein folding, including the amino acid composition and protein structure. Protein folding is the process by which a protein molecule assumes its functional 3D structure - a process crucial for the proper functioning of the protein and, by extension, the organism it belongs to.

4.1 Protein's Structure

Proteins are made up of amino acids, which are linked together by peptide bonds to form long chains. There are 20 different types of amino acids (see Appendix A) that can be found in them and each one has a distinct side chain that controls the specific chemical characteristics. The sequence of amino acids in a protein determines its primary structure. However, the folding process also involves interactions with other molecules, and these - together with the amino acid sequence - will determine the ultimate structure (there are also the secondary and higher-order structures that need to be determined). An overview of these compounds and the kinds of structures we are attempting to anticipate, is provided here.

4.1.1 Molecules in Protein Folding

Amino Acids

- **Definition**: Amino acids are the basic building blocks of proteins. Each amino acid has a central α-carbon attached to a hydrogen atom, a carboxyl group (-COOH), an amino group (-NH_2), and a unique R-group or side chain.
- **Variety**: There are 20 different naturally occurring amino acids, each with distinct properties based on their side chains.
- **Role in Folding**: The sequence of amino acids in a polypeptide chain determines the protein's primary structure, which subsequently influences the formation of secondary structures.

Water Molecules

- **Role**: Water molecules are crucial in the folding process due to hydrophilic (water-attracting) and hydrophobic (water-repelling) interactions. Hydrophobic side chains tend to cluster away from water, promoting specific folding patterns that contribute to secondary and tertiary structures.
- **Hydrogen Bonding**: Water can form hydrogen bonds with amino acid side chains, stabilizing the protein structure.

Ions and Metal Cofactors

- **Examples**: Ions such as calcium (Ca^{2+}), magnesium (Mg^{2+}), and metal ions like iron (Fe^{2+}) and zinc (Zn^{2+}) often play roles in stabilizing protein structures or participating in catalytic activity.
- **Coordination**: Metal ions can coordinate with amino acid side chains, influencing the folding and stability of proteins.

Chaperone Proteins

- **Function**: Chaperone proteins assist in the correct folding of other proteins, preventing misfolding and aggregation that could lead to malfunction or disease.
- **Types**: Examples include heat shock proteins (HSPs) and chaperonins.

4.1.2 Four Types of Structures

Primary Structure

The primary structure of a protein is simply the arrangement of amino acids that form the protein chain. The sequence is determined by the genetic code, which specifies the order of amino acids in a protein.

Secondary Structure

The secondary structure of a protein refers to the protein chain's local foldings, which are stabilized by hydrogen bonds between the amino acid residues. There are two main types of secondary structure: alpha helices (tightly coiled structures) and beta sheets (flat, extended structures).

Tertiary Structure

The tertiary structure of a protein refers to the overall 3D shape of the protein (reffering to an entire polypeptide chain). It is determined by a combination of hydrophobic interactions, hydrogen bonds, disulfide bonds, and other non-covalent interactions.

Quaternary Structure

Some proteins are made up of multiple subunits, each with its own tertiary structure. The way these subunits (in a multisubunit protein) come together to form a functional protein is referred to as the quaternary structure. The quaternary structure is stabilized by a combination of non-covalent interactions and sometimes covalent bonds.

4.2 Folding Pathways

The understanding of protein folding pathways has evolved significantly over time, transitioning from a classical model of discrete intermediates to a modern perspective characterized by a multipath energy landscape.

4.2.1 Historical Perspective

Understanding protein folding and structure has significantly benefitted from the pioneering work of several scientists throughout the past century. Notable among these contributors is Linus Pauling, whose groundbreaking research laid the foundation for much of our current knowledge about protein chemistry.

Pauling's primary contributions focused on the stabilization of protein structures through hydrogen bonds and the overall arrangement of polypeptide chains. He proposed the alpha-helix and beta-sheet as fundamental elements of protein secondary structure, highlighting the importance of hydrogen bonding in maintaining these structures. His work emphasized the geometric and chemical principles that lead to stable protein conformations, contributing immensely to the understanding of how protein folding occurs on a fundamental level.

Pauling was among the first to use concepts of structural chemistry to explain the stability and formation of complex biopolymers like proteins. One of his major contributions to protein chemistry was his identification and characterization of hydrogen bonds. Along with his colleague Alfred Mirsky, posited that hydrogen bonds are critical for maintaining the stability of protein structures. They demonstrated that hydrogen bonds, in addition to peptide bonds between amino acids, play a fundamental role in stabilizing the intricate configurations of protein molecules.

Expanding his research to functional proteins, he hypothesized that the distinct structural features of haemoglobin are essential prerequisites for its oxygen-binding capabilities. His work underscored the relationship between a protein's structure and its biological function, a concept that remains a central tenet in molecular biology.

4.2.2 The Classical View

Early assumptions about protein folding were largely influenced by the concept of distinct intermediate states. Researchers envisioned proteins folding in a manner analogous to classical biochemical pathways, where a series of well-defined steps guided the polypeptide chain from an unfolded state to its native conformation. This idea was epitomized in the pioneering work by Anfinsen, who demonstrated that proteins could spontaneously attain their native structure without external assistance, implying that the folding information is encoded within the amino acid sequence itself.

Also, Levinthal's paradox posed a significant challenge to the understanding of protein folding. Given the astronomical number of possible conformations, a purely random search for the native state would take an impractically long time. Levinthal proposed that proteins, therefore, must fold through predetermined, distinct pathways, leading to their native structure via specific intermediates.

- **Key Contributions:**
  - Levinthal's arguments highlighted the implausibility of random folding.
  - Experimentalists sought to identify and characterize these intermediates through various biochemical techniques.
- **Limitations:**
  - While this view provided a straightforward conceptual framework, it struggled to account for the diversity and complexity observed in protein folding behaviors.

4.2.3 The New View

The modern understanding of protein folding has transcended the simplistic view of a linear, sequential process dictated solely by the amino acid sequence. While the intrinsic properties encoded in the primary structure remain crucial, we now recognize that protein folding is a dynamic interplay between these inherent factors and the extrinsic environment in which it occurs. This new perspective emphasizes the cellular context as a key player in shaping the folding landscape and influencing the fate of a protein.

Early studies, while groundbreaking in establishing the importance of the primary sequence, often studied proteins in isolation, neglecting the crowded and complex environment of the cell. This reductionist approach, though valuable in elucidating fundamental principles, failed to capture the intricate dance between a nascent polypeptide chain and its surroundings.

The new view recognizes protein folding as a carefully orchestrated process, fine-tuned by a network of interactions within the cellular milieu. These interactions, far from being passive bystanders, actively participate in guiding the folding pathway, stabilizing intermediates, and ensuring the efficient production of functional proteins.

Intrinsic Factors

- **Primary Structure**: The unique sequence of amino acids in a protein encodes information crucial for forming secondary, tertiary, and quaternary structures. The folding process is largely driven by interactions such as hydrophobic packing, hydrogen bonding, van der Waals forces, and electrostatic interactions.
- **Hydrophobic Interactions**: The distribution of hydrophobic and polar residues guides the folding pathway, indicating that proteins with conserved hydrophobic contents but different sequences can still fold into their native structures.

Extrinsic Factors

- **Chaperones and Cofactors**: Proteins often require assistance to fold correctly in the crowded cellular environment. Molecular chaperones, such as the heat shock proteins, assist in folding by preventing aggregation and facilitating the formation of intermediate states. Chaperones can also help in refolding misfolded proteins and targeting irreversibly misfolded proteins for degradation.
- **Solution Properties and Environmental Conditions**: Factors such as pH, ionic strength, temperature, and the presence of co-factors or ligands can significantly influence protein folding. Cooperative binding of cofactors can stabilize specific conformations or facilitate the transition through folding intermediates.
- **Ribosome and Co-translational Folding**: As proteins are synthesized on the ribosome, they start folding co-translationally. The rate of translation and the vectorial nature of folding can affect the folding pathway and the formation of intermediate states, which tend to fold correctly if they have higher rates of codon translation.

The modern view of protein folding refines the classical model, proposing a multipath energy landscape approach to describe the process.

Energy Landscape Theory

- The modern understanding conceptualizes protein folding as a journey through a rugged, funnel-shaped energy landscape, introduced by Bryngelson, Onuchic, Socci, and Wolynes. This model posits that proteins fold through a multiplicity of pathways, rather than a single, defined route.

Key Features

- **Funnel Shape**: The energy landscape narrows as the protein approaches its native state, indicating an overall decrease in free energy and a reduction in conformational entropy.
- **Ruggedness**: The landscape is populated with various kinetic traps and energy barriers that transiently capture the folding protein, adding a statistical dimension to the folding process.
- **Frustration and Traps:** Configurational frustration can lead to intermediate states that are energetically favorable but not optimal, representing local minima that the protein must overcome.

Experimental approaches have evolved to provide insights into the dynamics of protein folding, despite significant challenges. Classical techniques such as crystallography and NMR spectroscopy offer detailed structural information but are limited by their inability to capture rapid, transient intermediate states. Instead, spectroscopic methods like fluorescence, circular dichroism (CD), and infrared (IR) spectroscopy have been employed for real-time kinetic studies, although they lack detailed structural resolution.

Case Studies

- **Thermodynamic Studies:** Work by Anfinsen and others solidified the concept that the native state is determined by the overall free-energy minimum.
- **Kinetic Intermediates:** Baldwin's investigations highlighted that folding intermediates could form asynchronously, supporting the multipathway model.

Computational Models

- Advances in computational power have facilitated simulations of the protein folding process. These models allow exploration of the energy landscape and prediction of folding pathways, albeit with challenges in capturing the full complexity of residue-level interactions and dynamics.

4.3 As a Physical Problem

Understanding the thermodynamic principles underlying protein folding is crucial in determining how proteins transition to their functional three-dimensional forms. The folding process is inherently linked to the principles of entropy and enthalpy, governed by the Boltzmann equation and the concept of Gibbs free energy. The Boltzmann entropy equation, given by:

$$S = k \ln \Omega$$

where S is the entropy, k is the Boltzmann constant, and Ω represents the number of microstates, is fundamental in describing the statistical spread of molecular configurations. For protein folding, this suggests a move towards a state with minimized entropy under biologically favorable conditions.

The change in entropy (ΔS) during protein folding can be calculated from the heat exchanged (ΔQ) over the temperature (T) of the system, following the equation:

$$\Delta S = \int \frac{\Delta Q}{T}$$

Protein folding is also a process driven by changes in enthalpy (ΔH), which measures the heat absorbed or released under constant pressure, crucial in describing the energy landscape that proteins navigate during folding. The relationship between these thermodynamic quantities dictates the spontaneity of folding; wherein higher temperatures ($T\Delta S > \Delta H$), folding is spontaneous ($\Delta G < 0$), indicating a decrease in Gibbs free energy.

Anfinsen's dogma holds that the amino acid sequence of a protein determines its 3D structure. This means that all the information necessary for folding the peptide chain into its native structure is encoded in its primary structure, and implies that the entropy of the system decreases guided by its amino acid sequence - as it folds into a stable conformation.

Protein folding is characterized by a significant reduction in entropy, as the flexibility of the amino acid chain becomes constrained upon adopting a specific structure. This reduction is often compensated for by enthalpic contributions, particularly favorable interactions like hydrogen bonding and van der Waals forces, which are guided by the protein's amino acid sequence. Anfinsen's dogma, stating that the amino acid sequence determines the 3D structure, underscores this principle. The sequence guides the protein towards lower entropy by dictating these interactions, driving the folding towards its native state in a thermodynamically favorable process ($\Delta G < 0$, where ΔG is the change in Gibbs free energy).

Furthermore, the folding process is influenced by the primary structure of proteins, where the sequence of amino acids dictates the complex folding pathways leading to functionally active structures. This complexity is further modulated by secondary structures like alpha helices and beta sheets, stabilized by hydrogen bonds, which contribute significantly to the decrease in system's total entropy during folding. Understanding these interactions and thermodynamic incentives provides a profound insight into the molecular mechanics of life and the pivotal role of entropy and enthalpy in biological self-organization.

4.4 Biological Significance

The functioning of biological systems relies heavily on the correct folding of proteins. Functional proteins perform a myriad of roles, including enzymatic catalysis, signal transduction, structural support, and immune responses. Misfolding or aggregation of proteins can lead to diseases such as Alzheimer's and Parkinson's. This underscores the necessity for a clear understanding of the protein folding process to develop therapeutic strategies for these conditions.

Research into protein folding not only advances our understanding of fundamental biological processes but also has significant implications for drug discovery and design. The ability to predict protein structures accurately can facilitate the development of drugs that specifically target certain proteins or protein-protein interactions. Moreover, understanding protein folding pathways can aid in the development of interventions to prevent or correct misfolding, contributing to treatments for protein misfolding diseases.

Chapter 5

Related Work

5.1 Simulations

5.1.1 Dynamic Simulations on Molecular Level (MD)

Molecular Dynamic Simulations provide a microscopic view of molecule interactions over time, pivotal for understanding complex biochemical processes. These simulations involve computationally intense methods where atoms and molecules follow the laws of physics, allowing researchers to observe the folding pathways of proteins in a dynamic and time-dependent manner. However, the applications are limited by substantial computational resource requirements. Furthermore, they fall short in terms of sampling efficacy due to long relaxation times.

To tackle these, enhanced sampling strategies and integration with machine learning methods are used to improve their efficiency. Also, technologies like GPU acceleration and multi-processor computing have further pushed the boundaries, enabling the study of longer timescale events and more complex systems which were previously unattainable. This computational evolution brings us closer to more accurate, real-time simulations of protein dynamics, revealing essential mechanisms that underpin biological functions and disease mechanisms.

Nature of Method	Deterministic (Deterministic algebraic equations)
Properties Studied	Static and dynamic
Time Evolution	Provides detailed time evolution information
Sampling Efficacy	Less effective than MC simulations due to long relaxation times
Challenges	Computationally intensive, sensitive to long simulation times

Table 5.1.1: Features of Molecular Dynamics Simulations

5.1.2 Monte Carlo Simulations (MC)

Monte Carlo stochastic method helps in exploring various conformational states of proteins but requires nuanced expertise to achieve high accuracy, striking a balance between computation cost and simulation reliability. It employs randomized trials to explore the vast conformational space of proteins, offering insights into stable structures under various conditions. Through iterative refinement and the ability to bypass local minima in energy landscapes, Monte Carlo simulations enhance our predictive accuracy and speed, essential for engineering robust proteins and understanding disease mechanisms linked to protein misfolding.

This predictive prowess was particularly highlighted in the study of Dihydrofolate Reductase mutations, showcasing its utility in practical, biomedical applications. As this technique continues to be honed, its integration with experimental methods promises a more comprehensive elucidation of protein behaviors.

MC simulations utilize stochastic elements based on finite Markov chains to navigate the conformational space of proteins. These simulations are renowned for their heuristic nature and iterative refinement processes, which help avoid local minima, thereby enhancing predictive accuracy. They are generally faster than MD simulations but are limited to static properties and do not provide time evolution information.

They are highly effective in sampling configurational space, making them suitable for exploring numerous conformations. Although they typically lack the sophisticated time-dependent dynamics provided by MD simulations, they are valuable in providing insights into the static configurational landscape of proteins.

Nature of Method	Stochastic (Stochastic elements based on finite Markov chains)
Properties Studied	Static only
Time Evolution	Does not provide time evolution
Sampling Efficacy	Highly effective in sampling configurational space
Challenges	Limited to static properties, no kinematic or dynamic information

Table 5.1.2: Features of Monte Carlo Simulations

5.2 Other Conventional Computational Techniques

5.2.1 Template-Based Modeling (TBM)

Template-Based methods rely on established structural databases to model protein structures using approaches like threading and homology:

- **Threading/Fold Recognition:** This method is used to model proteins that have the *same fold* as proteins of known structures, but do not necessarily have high sequence similarity to any known structures.
- **Homology Modeling:** This approach predicts the structure of a protein by using homologous proteins with known structures, relying on significant *sequence similarity* to identify structurally conserved regions.

Threading, particularly, matches unknown protein sequences with known structures, enhancing prediction reliability. Its foundational premise is that proteins with similar structures can be identified even without high sequence similarity. Homology (or comparative) modeling employs the structural data already available in databases such as the Protein Data Bank (PDB) to model unknown protein structures. Essentially, it begins by identifying a parent structure close to the sequence of interest, onto which the target sequence is mapped. This initial model is then refined through a series of steps that adjust the backbone, optimize side chain packing, and model loops to enhance the accuracy of the prediction.

Challenges arise mainly from achieving a reliable sequence alignment that correctly maps the 1D protein sequence onto the 3D structure of the template and from refining this initial model to resemble the native structure more closely. Misalignments, even if minor, can significantly skew the model away from its true form, underscoring the critical need for precise alignment and effective refinement strategies to approach the native conformation more closely.

TBM is particularly useful when suitable templates are available, allowing for relatively accurate predictions. The main limitation of TBM is its dependency on existing protein structures, which might not be available for all proteins.

Feature	**Description**
Dependency	Relies on existing structural databases
Approach Types	Threading/Fold Recognition and Homology Modeling
Accuracy	Relatively accurate when suitable templates are available
Limitations	Limited by the availability and quality of known templates

Table 5.2.1: Features of Template-Based Modeling

5.2.2 Free Modeling

In computational protein structure prediction, free modeling methods are crucial as they do not rely on known templates, making them ideal for novel protein fold predictions. Notable among these approaches are **Ab Initio** methods, which predict structures from scratch using physical principles, and **Fragment-Based Sampling**, which uses short protein fragments from the Protein Data Bank (PDB) to construct the full protein structure.

Ab Initio

Ab Initio technique is employed when no suitable template exists, relying purely on the first principles to predict protein structures, usually applied to smaller proteins due to high computational demands. It represents a pivotal strategy, particularly valuable for proteins lacking homologous structures in existing databases. This method relies on the principle that a protein in its native state attains the lowest free energy conformation.

This method typically employ advanced optimization processes that can vary in the representation of proteins, the level of detail in sidechains, and the positions of carbon alpha atoms. Techniques such as Molecular Dynamics and Monte Carlo simulations are commonly used, helping simulate the folding process and explore the vast conformational space despite the astronomical number of possible protein folds.

A significant obstacle presented by ab initio technique in protein structure prediction is its demanding computation requirements. These method necessitate substantial computational resources, making them less accessible for widespread use. Additionally, ab initio model face inherent difficulties in replicating the rapid folding processes of proteins, a phenomenon highlighted by the Levinthal paradox, which posits that a protein does not randomly sample all possible configurations to achieve its native structure due to the astronomical number of possibilities. Instead, proteins fold quickly into their functional forms, a process that current ab initio techniques struggle to simulate both efficiently and accurately.

Feature	Description
Thermodynamic Hypothesis	Based on the principle that the native protein structure presents the lowest free energy possible
Template Free Approach	Breaks protein sequence into amino acid fragments, united using Monte Carlo simulations with force fields and energy terms
Computational Complexity	Computationally intensive due to large number of conformational possibilities
Advantages	Capable of predicting novel and unknown protein folds

Table 5.2.2: Features of Ab Initio Methods

Fragment-Based Methods

Fragment-based sampling operates by leveraging small protein fragments, typically derived from known protein structures in the Protein Data Bank (PDB), to construct the entire protein structure. These fragments represent common structural motifs and local conformations, key components in the hierarchical assembly of protein structures. Fragment-Based methods help reduce the vast conformational space that proteins can adopt, thus enabling more efficient and accurate folding predictions. They have demonstrated substantial success in protein structure prediction over the past decade.

Using the pre-existing structural motifs, fragment-based methods incrementally build up the protein structure by assembling fragments in a manner consistent with the target sequence's constraints. This approach drastically narrows down the number of potential conformations that need to be explored, significantly lowering computational complexity compared to ab initio methods. Although, despite the evident computational efficiency, these methods have the limitation of possibly missing entirely novel folds, as they rely heavily on existing structural data. However, for many practical applications, the trade-off is acceptable due to the higher prediction accuracy and the ability to predict larger protein structures compared to purely ab initio methods. Advanced techniques like Rosetta [1] utilize fragment-based approaches combined with energy minimization and refinement steps to achieve highly accurate models.

One of the strengths of fragment-based methods lies in their ability to blend with other computational techniques, such as Molecular Dynamics or enhanced sampling methods, to refine and validate the predicted structures. This integrative approach can improve the robustness and reliability of the predictions made by fragment-based sampling.

[1] Rosetta assembles fragments from known protein structures via a Monte Carlo procedure, evaluating the models with a scoring function. In the fragment assembly phase, local energy minimization is performed that helps in refining the placement of fragments to create plausible initial models.

5.3 Machine Learning

Machine learning (ML) models are transforming protein folding prediction by leveraging vast datasets to infer complex patterns in protein structures. The quality of the training data is crucial for their efficacy, enabling significant refinement and improvement in predictive accuracy.

Categories

AlphaFold-Based Methods: Utilize the AlphaFold Protein Structure Database (AlphaFold DB), featuring extensive entries of protein structures.
Protein Language Models: Adapt natural language processing (NLP) techniques to analyze amino acid sequences, capturing evolutionary and structural patterns for predicting key structural features.
Machine/Deep Learning Techniques: Employ advanced neural networks, including Graph Neural Networks (GNNs), Convolutional Neural Networks (CNNs), and Artificial Neural Networks (ANNs), trained on vast datasets like those from the Protein Data Bank (PDB) to predict distances and torsion angles accurately.
Integration with Molecular Simulations: Integrate machine learning models with molecular dynamics (MD) simulations, enhancing protein trajectory and interaction analysis through machine learning-adapted force fields, improving timescales and precision.

Key Features

Data-Driven Approach: Extract information from large and representative datasets; the quality of these datasets is pivotal for accurate predictions.
Accuracy and Efficiency: ML-based protein folding methods are highly accurate, used for designing new drugs and understanding protein structures and functions. They benefit from improved real-time adaptive force fields, enhancing prediction speed and accuracy.
Challenges: Machine learning models often act as ’black boxes,’ providing limited insight into the underlying biophysical mechanisms they simulate, which presents an ongoing interpretability challenge.

5.4 AlphaFold

Developed by DeepMind, AlphaFold utilized machine learning for predicting protein structures from amino acid sequences. While AlphaFold 1 showcased potential at the CASP competition, it did not exhibit the precision necessary for extensive scientific application. Subsequent optimizations over the following years have resulted in remarkable accuracy.

Feature	AlphaFold 1 (2018)	AlphaFold 2 (2020)	AlphaFold 3 (2024)
Architecture	Initial Deep Learning Model	Introduction of the Evoformer	Advanced Evoformer and Diffusion Network
Main Components	Typical neural networks	Evoformer, MSA Transformer	Enhanced Evoformer, Diffusion Network
Accuracy	Moderate	Near-experimental accuracy	Significantly improved accuracy
Speed	Slower predictions	Fast predictions	Optimized for speed
Input	Protein sequence, MSA stats	Protein sequence, MSA, pairwise features	Includes interactions with DNA, RNA, small molecules
Output	Distograms	End-to-end 3D structure	Joint 3D structures of multiple molecules
Optimization	CNN on PDB, Gradient Descent	Iterative SE(3)-Transformer	Improved Evoformer, diffusion from atoms

Table 5.4.1: Features of AlphaFold Models

5.4.1 AlphaFold 1 (2018)

The initial version of AlphaFold was introduced in 2018, marking a substantial step forward in protein structure prediction. It utilized a deep learning model trained on data primarily from the Protein Data Bank (PDB), employing supervised learning techniques. While AlphaFold 1 demonstrated potential by performing competitively at the Critical Assessment of protein Structure Prediction (CASP) competitions, it was limited by its precision, restricting its widespread adoption in scientific applications.

5.4.2 AlphaFold 2 (2020)

The advent of AlphaFold 2 in 2020 marked a revolutionary improvement in both the accuracy and speed of protein structure prediction. This version introduced the Evoformer, an architecture incorporating attention-based mechanisms adept at capturing long-range interactions between amino acids by focusing on multiple sequence alignments (MSA) and pairwise representations. AlphaFold 2 achieved near-experimental accuracy, as evidenced by its remarkable performance at CASP competitions, providing predictions within minutes and including confidence ratings to enhance usability and reliability.

5.4.3 AlphaFold 3 (2024)

AlphaFold 3, launched in 2024, builds upon and extends the capabilities of its predecessor. It not only predicts protein structures but also interactions among a broad spectrum of biomolecules, including proteins, DNA, RNA, and ligands. AlphaFold 3 exhibits significant improvements, achieving double the prediction accuracy in specific types of interactions and a 50% enhancement over prior methods for predicting biomolecular interactions. This is enabled by an advanced Evoformer coupled with a Diffusion Network, refining a cloud of atoms into a precisely defined molecular structure.

5.5 Hybrid Quantum-Classical Machine Learning Models

5.5.1 Resource-Efficient Quantum Algorithm

The paper [11] introduces a quantum algorithm designed to address the protein folding problem on a tetrahedral lattice using a coarse-grained model. The central component of the algorithm is the Hamiltonian $H(q)$, which encodes the conformational energy landscape of the protein. The Hamiltonian is represented as a sum of Pauli strings:

$$H(q) = \sum_{i=1}^{N} h_i q_i,$$

where h_i are real coefficients, and $q_i = \frac{1-z_i}{2}$, with z_i being Pauli-Z matrices for each qubit i, and N_t being the total number of terms. The scaling is governed by the number of terms (Pauli strings) in the Hamiltonian corresponding to the qubit Hamiltonian, which scales quadratically with the number of amino acids N:

$$N_t = O(N^4).$$

The optimization process utilizes the Conditional Value-at-Risk Variational Quantum Eigensolver (CVaR-VQE) algorithm, which is more resilient to noise. This method focuses on optimizing diagonal Hamiltonians and provides a drastic speed-up compared to conventional VQE. The key feature involves defining an objective function based on the average over the tail of a distribution, which enhances the optimization process. The classical part of the optimization is facilitated by a differential evolution optimizer. During optimization, the wavefunctions $|\psi(p)\rangle$ of different individuals p are measured, collapsing them into binary strings uniquely corresponding to specific configurations and energies.

For practical implementation, the authors applied the algorithm to the folding of the 10 amino acid peptide Angiotensin. Initially, this required 35 qubits using a coarse-grained model on the tetrahedral lattice. However, a more efficient encoding was introduced as explained below, requiring only 2 qubits per turn:

$$t_i = q_{2i-1}q_{2i},$$

reducing the total qubit count to 22. Consequently, this led to the generation of 5-local terms in the qubit Hamiltonian instead of 3-local, while maintaining the number of Pauli strings within manageable limits for small instances.

Tetrahedral Lattice

The tetrahedral lattice is a geometrical arrangement where each bead has four possible neighboring positions. This structure is ideal for modeling the three-dimensional conformation of polymers, as it allows modeling turns and interactions efficiently. In this lattice, the spatial distance between two beads is a function of the number of turns separating them along the main chain.

Turn Encoding

Researchers introduce an indicator for the axis chosen at turn t_i. This function, $f_a(q_{2i-1}, q_{2i}) = f_a(i)$, returns 1 if the axis $a \in \{0, 1, 2, 3\}$ is chosen at turn t_i. For the denser encoding, this function is given by:

$$f_0(i) = (1 - q_{2i-1})(1 - q_{2i}) \tag{5.5.1}$$
$$f_1(i) = q_{2i}(q_{2i} - q_{2i-1}) \tag{5.5.2}$$
$$f_2(i) = q_{2i-1}(q_{2i-1} - q_{2i}) \tag{5.5.3}$$
$$f_3(i) = q_{2i-1}q_{2i} \tag{5.5.4}$$

For the sparser encoding, the expression of $f_a(i)$ is straightforward:

$$f_0(i) = q_{4i-3} \tag{5.5.5}$$
$$f_1(i) = q_{4i-2} \tag{5.5.6}$$
$$f_2(i) = q_{4i-1} \tag{5.5.7}$$
$$f_3(i) = q_{4i} \tag{5.5.8}$$

Alternate Lattices A and B

To enforce correct chirality and interaction dynamics, researchers use two alternating sub-lattices, A and B. The polymer starts with a bead on sub-lattice B. Due to the alternation of these sub-lattices, all sites of sub-lattice A (denoted by even integers) and sub-lattice B (denoted by odd integers) follow a consistent pattern.

This alternation ensures that the beads' positions reflect their parity. Parity g_i is defined as:

$$g_i = 1 - \frac{(-1)^i}{2}$$

Each turn's directionality is captured in the turn parameter t_i, with a specific constraint table ensuring that the turns comply with the chirality. This table guarantees that every turn aligns appropriately with the directionality imposed by the alternating lattices.

In the implementations on this thesis, we use the same geometric constraints and interaction terms (Miyazawa & Jernigan terms), to construct the Hamiltonian, and the same tetrahedral lattice and qubit encoding methods to represent amino acid positions and directions. We will use Quantum Approximate Optimization Algorithm (QAOA) to solve the problem.

5.5.2 Digitized-Counterdiabatic Quantum Algorithm

The Theory

For the Counterdiabatic Quantum Algorithm (CDQA) for protein folding [1], the key idea is to construct an ansatz inspired by counterdiabatic (CD) terms and utilize gradient-based optimization to minimize a cost function.

Shortcuts to adiabaticity are used to speed up the adiabatic process in quantum computing. The adiabatic theorem states that a quantum system remains in its ground state if a given perturbation is slow enough. However, this process can be quite slow, which is not ideal for quantum computing applications. If the Hamiltonian changes quickly, the system may transition to a different eigenstate, known as a non-adiabatic transition.

The concept of Counter-Diabatic (CD) driving was introduced to overcome this limitation. The idea is to add an extra term to the Hamiltonian (the CD field) that cancels out the non-adiabatic transitions, effectively driving the system through its evolution faster while still maintaining the system in its instantaneous ground state. We will call such a field the *counter-diabatic field*, H_{CD}.

If H_S is the isolated system Hamiltonian and H_{SF} is the system-field interaction, then it follows that:

$$H(t) = H_S + H_{SF}(t) + H_{\mathrm{CD}}(t)$$

Considering that $UHU^\dagger$ – where U is a time-dependent unitary transformation – is diagonal for Hamiltonians that generate adiabatic transfer between states, it follows that:

$$H_{\mathrm{CD}}(t) = i\hbar \left(\frac{\partial U^\dagger(t)}{\partial t} \right) U(t)$$

The Hamiltonian changes slowly with time, and the function $\lambda(t)$ controls this rate of change.

With the CD term, the condition on $\lambda(t)$ is relaxed, as the CD term can suppress non-adiabatic transitions; the Hamiltonian can change more quickly while keeping the system in its instantaneous eigenstate.

The researchers use the adiabatic gauge potentials (related to the rate of change of the Hamiltonian) to obtain approximate Counter-Diabatic (CD) terms, and the Nested Commutator (NC) method to calculate these. The NC method involves calculating a series of commutators of the Hamiltonian and its derivatives with respect to the parameter $\lambda(t)$. The CD term is then given by a sum over these commutators, weighted by the rate of change of $\lambda(t)$.

Since this sum is difficult to implement directly in a quantum circuit, we digitize the CD terms (approximate them using a finite number l of terms). Therefore, we use the series expansion:

$$A_\lambda^l = i \sum_{k=1}^{l} a_k(t)[H_a, [H_a, \dots, [H_a, \partial_\lambda H_a]]]$$

So, the quantum adiabatic evolution with counter-diabatic protocol Hamiltonian is:

$$H(t) = (1 - \lambda(t))H_{\text{mixer}} + \lambda(t)H + \dot{\lambda}(t)A_\lambda^l$$

The CD-inspired ansatz is then given by the evolution:

$$U_{\text{cd}}(\theta_p)U_b(\beta_p)U_c(\gamma_p)U_{\text{cd}}(\theta_{p-1})U_b(\beta_{p-1})U_c(\gamma_{p-1}) \times \cdots \times U_{\text{cd}}(\theta_1)U_b(\beta_1)U_c(\gamma_1)$$

However, since the CD term can play a dominant role in the evolution of the system, instead of trying to implement the full evolution of the system using all three terms (U_c, U_b, and U_{cd}), the researchers focus only on the contributions from the CD term:

$$U_{\text{cd}}(\theta) = e^{-i\theta A}$$

Cost Function and Ansatz

The cost function C used in the optimization is given by:

$$C = \langle\psi(\theta)|H|\psi(\theta)\rangle, \tag{5.5.9}$$

where H is the problem Hamiltonian and $|\psi(\theta)\rangle$ is the parameterized quantum state.

For the CD-inspired ansatz, we employ parameterized Y-rotations followed by YZ-rotations. Specifically, the ansatz can be parameterized as follows:

$$Y(\theta) = \exp\left(-i \sum_{m=1}^{N} \theta_m Y_m\right), \tag{5.5.10}$$

$$YZ(\theta) = \exp\left(-i \sum_{i,j=1}^{N} J_{ij}\theta_{ij}Y_iZ_j\right), \tag{5.5.11}$$

where Y_m and Y_iZ_j are the Pauli operators, and θ represents the set of parameters to be optimized.

Quantum Adiabatic Evolution

We consider the quantum adiabatic evolution with the counterdiabatic protocol Hamiltonian H_{cd}:

$$H_{cd}(t) = (1 - \lambda(t))H_{\text{mixer}} + \lambda(t)H + \gamma(t)A_l, \tag{5.5.12}$$

where H_{mixer} is a Hamiltonian whose ground state is easy to prepare, $\lambda(t)$ and $\gamma(t)$ are scheduling functions, and A_l represents the approximate CD term of order l.

The scheduling functions satisfy the boundary conditions:

$$\lambda(0) = 0, \quad \lambda(T) = 1, \tag{5.5.13}$$

$$\gamma(0) = 0, \quad \gamma(T) = 0, \tag{5.5.14}$$

where T is the total evolution time.

Digitized Counterdiabatic Evolution

In the digitized counterdiabatic approach, the continuous scheduling functions are replaced with discrete time steps. The Hamiltonian for digitized counterdiabatic quantum approximate optimization algorithm (DC-QAOA) is then:

$$H_{dcq}(k) = H_{\text{mixer}} + \Delta t_k \left(H + A_l\right), \tag{5.5.15}$$

where Δt_k represents the time step, and k indexes the time step in the digitized evolution.

Chapter 6

Understanding the Code

6.1 Qiskit's Libraries

In the field of protein folding, hybrid quantum-classical machine learning approaches have demonstrated significant potential. Qiskit, a framework developed by IBM, stands at the forefront, facilitating the integration of classical and quantum computing techniques. This section delves into the various Qiskit libraries pertinent to protein folding: `qiskit`, `qiskit_research`, `qiskit.algorithms`, and `qiskit_nature`. The last three libraries had been used for the implementations in chapter 7, but since now the corresponding files have been deleted, I will also provide their key functions and features that encoded the problem (Sec. 6.3).

6.1.1 Qiskit

Qiskit serves as the foundational library providing the tools necessary for quantum computing. It supports the creation, manipulation, and execution of quantum circuits on both simulators and real quantum hardware. This library is crucial for setting up the basic elements of quantum computations that are used in larger algorithms and applications, including those in protein folding.

6.1.2 Qiskit Research

The `qiskit_research` repository, particularly the `protein_folding` module, introduces specialized tools for simulating and analyzing protein structures using quantum computing.
For instance, in `qiskit_research/protein_folding/peptide/beads/base_bead.py`, the protein's 3D Tetrahedral Lattice Model and turn encoding mechanisms are established. This involves encoding peptide chain turns as quantum states within a tetrahedral lattice, using Pauli operators to represent turn directions, thereby allowing for a quantum representation of the protein's 3D structure.

6.1.3 Qiskit Algorithms

The `qiskit.algorithms` library includes advanced quantum algorithms that are pivotal in protein folding studies. This includes Variational Quantum Eigensolver (VQE) and Quantum Approximate Optimization Algorithm (QAOA), both of which are employed to find the ground state of the Hamiltonian representing the protein's folded state. These algorithms leverage quantum mechanics to optimize the protein configuration more efficiently than classical-only approaches.

6.1.4 Qiskit Nature

`qiskit_nature` deals directly with applications of quantum computing in the natural sciences, including chemistry and biology. This library interacts seamlessly with `qiskit_research` to model molecular and biomolecular data. In the context of protein folding, it interprets and processes the biochemical properties of amino acid chains, facilitating the translation of biological data into quantum operations that can be used by `qiskit.algorithms`.

6.2 The Implementation of the Hybrid Algorithms

In this section, I am attaching the general outline for the implementation of hybrid quantum-classical machine learning algorithms. The code used in Chapter 8 is based on that.

1. Classical Representation of the Protein Folding Problem:

- **Interaction Energy Dictionary:** A dictionary `interaction_energy` stores the pairwise interaction energies between different amino acids. This dictionary captures the energetic favorability or unfavorability of different amino acid pairs being in proximity.
- **Protein Sequence Encoding:** The input protein sequence is provided as a string (e.g., "APGGPR").
- **Pyomo Model:** The protein folding problem is formulated as a classical optimization problem using the Pyomo library. The `folding_hamiltonian` function constructs a Pyomo model that encapsulates:
  - **Variables:** Binary variables (`model.f`, `model.interaction`) represent the folding configuration and amino acid interactions.
  - **Constraints:** Geometric constraints prevent unrealistic protein folds (e.g., consecutive amino acids folding back on themselves).
  - **Objective Function:** The objective function (`model.obj`) represents the total energy of the protein conformation, combining terms for geometric constraints and amino acid interactions.

2. Transformation to a Quantum Representation:

- **Objective Function Extraction:** The classical objective function, expressed in terms of binary variables, is extracted from the Pyomo model.
- **Conversion to Quantum Hamiltonian:** The classical objective function is transformed into a quantum Hamiltonian. This involves:
  - Replacing binary variables with Pauli-Z operators.
  - Constructing a `SparsePauliOp` object that represents the Hamiltonian in terms of Pauli operators and their corresponding coefficients.

3. Quantum Optimization using QAOA:

- **QAOA Ansatz:** A parameterized quantum circuit (`qaoa_ansatz`) is created to represent the QAOA ansatz. This circuit consists of alternating layers of:
  - **Cost Hamiltonian:** Derived from the protein folding problem's objective function, this Hamiltonian favors low-energy protein configurations.
  - **Mixer Hamiltonian:** Enables exploration of the solution space by introducing transitions between different configurations.

- **Parameter Initialization:** Initial values for the QAOA parameters (`gamma_values`, `beta_values`) are randomly generated.
- **Quantum Simulation:** The QAOA circuit is executed on a quantum simulator or real quantum hardware. This involves:
  - **Transpilation:** The circuit is transpiled to match the topology and gate set of the chosen backend.
  - **Execution:** The transpiled circuit is executed, and the results are collected.
- **Measurement and Energy Calculation:** The measurement results from the quantum simulation are used to estimate the average energy of the protein configurations.

4. Hybrid Optimization Loop:

- **Classical Optimization:** A classical optimizer (e.g., COBYLA) is employed to find the optimal QAOA parameters that minimize the average energy of the protein.
- **Iterative Process:** The classical optimizer iteratively updates the QAOA parameters, and the quantum simulation is repeated with the new parameters until convergence or a stopping criterion is met.

5. Results Interpretation:

- **Optimal Parameters:** The optimization process yields the best set of QAOA parameters found.
- **Minimum Energy:** The average energy corresponding to the optimal parameters represents an approximation of the ground state energy of the protein folding problem.
- **Protein Conformation:** By analyzing the measurement results from the quantum simulation, the most probable protein configurations, corresponding to low-energy solutions, can be identified.

6.3 Encoding the Problem with Qiskit's Libraries & MJ model

6.3.1 Creating the Hamiltonian

In the `qiskit_research` library, the handling of bead contacts and distance mapping is particularly noteworthy. For example, `contact_map_builder.py` implements a method to define contacts between beads (representing amino acids) using parity checks that ensure beads fall into alternating lattice types in the tetrahedral model. This aids in mapping the protein's folding pattern onto quantum systems.

The Hamiltonian construction in `qiskit_research` uses these mapped positions and interactions. The distance map and contact data are synthesized into a Hamiltonian, whose minimization (achieved using the VQE or QAOA) guides the protein to fold into its energetically favorable conformation. This includes penalty terms enforcing biological and chemical plausibility, such as correct chirality and avoidance of unphysical overlaps between amino acids.

6.3.2 3D Tetrahedral Lattice Model with Turn Encoding and Miyazawa-Jernigan (MJ) Interactions

In the model I am using, each point in the lattice can be connected to four other points, corresponding to the four vertices of a tetrahedron. Therefore, at each point, an amino acid can turn in one of four directions. This is why two qubits are used to encode each turn—four different states (00, 01, 10, 11).

MJ Interactions

The Miyazawa-Jernigan (MJ) interaction model is utilized to estimate contact energies within protein structures. A fundamental part of this model is the Bethe approximation, also known as the quasi-chemical approximation. This approximation achieves an exact solution for the Bethe lattice and treats the formation of contact pairs as a chemical reaction. The equilibrium relationship under this approximation can be expressed as:

$$e^{-e_{ij}} = \frac{n_{ij}n_{00}}{n_{i0}n_{j0}} \tag{6.3.1}$$

In this equation, e_{ij} represents the contact energy between residues i and j, and n_{ij} denotes the statistical average number of contacts between these residues. The contact energies are dimensionless and measured in units of RT. While this relationship traditionally derives n_{ij} from known e_{ij} values, it can also invert to estimate contact energies from observed contact numbers in protein structures.

The parameter n_{00}, which represents the number of contacts between effective solvent molecules, proves challenging to estimate directly. However, the relative differences in contact energies can be more reliably estimated than their absolute values as they do not depend on n_{00}:

$$e^{-\Delta e_{ij}^{kl}} = \frac{n_{ij}n_{k0}n_{l0}}{n_{i0}n_{j0}n_{kl}} \tag{6.3.2}$$

Each protein's effective number of solvent molecules is estimated based on their expected contact numbers under hypothetical hard-sphere interaction conditions with $e_{ij} = 0$. Assumptions are made that effective solvent molecules possess the same volume as average residues.

Contact numbers, n_{ij}, are tallied for each protein structure, and the sums N_{ij} over all protein samples are computed. For proteins, contacts with solvent molecules, n_{i0}, are estimated through the volume of each amino acid type and the average volume of its surrounding residues, thereby filling incomplete coordination spheres with solvent molecules.

This modeling aligns with the structural consistency principle, initially proposed by Go (1983) and revisited by Bryngelson and Wolynes (1987) in the context of the minimal frustration principle in protein energy landscapes. According to this principle, the inherent contact interactions should be consistent with the high stability of native protein structures.

Here are some key points highlighting its aspects:

1. **Statistical Averaging**: The model determines statistical average numbers of contacts (n_{ij}) between various residues, making use of extensive data from known protein structures.
2. **Pseudo-Potentials**: The contact energies in the MJ model are derived from pseudo-potentials, which are empirical potentials based on observed statistical features in protein structures.
3. **Sample Weighting**: A sampling weight is used for each protein based on sequence homology to ensure an unbiased sampling from the Protein Data Bank (PDB).
4. **Bethe Approximation**: The model employs the Bethe lattice in estimating attractive contact energies, which is a statistical mechanics approach.
5. **Empirical Contact Potentials**: The terms used in calculating energies (side chain packing, hydration, hydrogen bonding, and local conformational potentials) are derived from empirical observations and statistical features among a significant number of protein structures.

3D Tetrahedral Lattice Model and Turn Encoding

Qubits are used to represent the turns taken by an arbitrary protein in 3D. The lattice consists of two sets, P and Q, which are exactly inverse to each other. The turns taken by the amino acids can be in one of four directions. Two qubits are devoted to encode each turn. At every turn, P and Q are alternated so that even turns are represented by P and odd turns by Q.

An "even-numbered position" refers to its place in the sequence of turns taken by the protein. If you list all the turns taken by the protein in order, the first turn is at position 1, the second turn at position 2, the third turn at position 3, and so on. Turns at positions 2, 4, 6, etc. (the "even-numbered" positions) are represented by the set P, while turns at positions 1, 3, 5, etc. (the "odd-numbered" positions) are represented by the set Q.

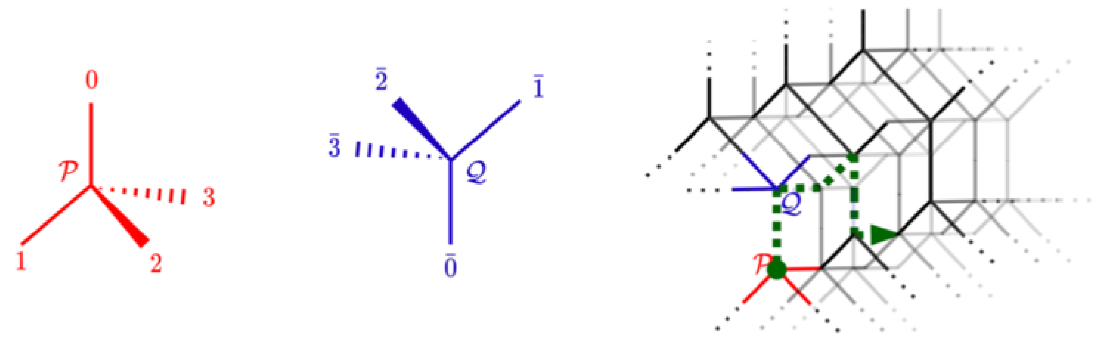
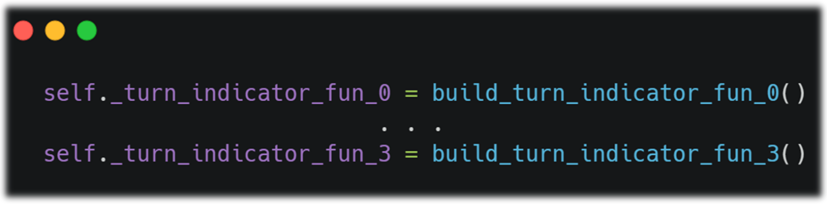


Figure 6.1: The tetrahedral lattice structure of the P lattice and its inverted lattice Q - Photo retrieved from [1]

In Qiskit, the presence of turn indicator functions (Figure 6.2) suggests encoding within a tetrahedral lattice structure. The peptide chain's turns are encoded as quantum states, with each turn corresponding to a direction in the lattice.



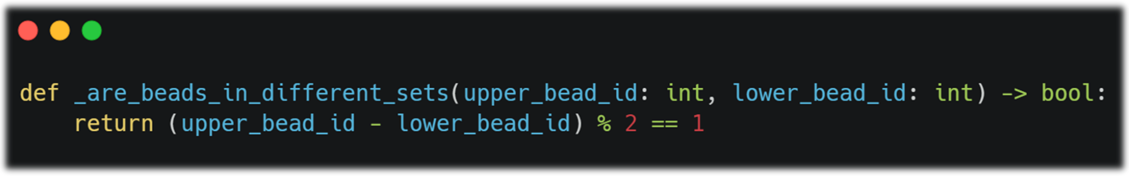
```
self._turn_indicator_fun_0 = build_turn_indicator_fun_0()
                            . . .
self._turn_indicator_fun_3 = build_turn_indicator_fun_3()
```

Figure 6.2: Turn Encoding with Qiskit Research (`base_bead.py`)

The `turn_qubits` parameter is a tuple of two Pauli operators, used to encode the direction of the turn that the peptide chain takes after the current bead. This allows the quantum simulation to represent the 3D structure of the peptide.

Alternating Lattices

The model alternates between two lattice types (P and inverted) at each turn to represent the protein's structure. The following function (Figure 6.3) checks whether two beads belong to different sets by checking if the difference in their IDs is odd:



```python
def _are_beads_in_different_sets(upper_bead_id: int, lower_bead_id: int) -> bool:
    return (upper_bead_id - lower_bead_id) % 2 == 1
```

Figure 6.3: Alternating Lattices with Qiskit Research (`contact_map_builder.py`)

If the IDs of the two beads differ by an odd number, they are in different sets, which correspond to the different sub-lattices in a tetrahedral lattice model. This simple parity check is a common method to alternate elements in a sequence.

6.4 Choosing the Classical Optimizer - A Baseline Optimizer

In the course of my research involving the application of VQE and QAOA methodologies to elucidate the 3D structures of peptides, the choice of an effective optimizer emerged as a pivotal consideration. The COBYLA optimizer was selected based on its robust performance detailed in relevant quantum computing literature [32] [35], including demonstrated applications in quantum chemistry simulations.

COBYLA is an optimization algorithm that does not require gradient information, which makes it admirably suited for quantum computations where such gradients can be computationally expensive or unreliable due to quantum noise. This attribute aligns well with scenarios in quantum chemistry, where the evaluation of gradients may introduce significant computational overhead.

Moreover, the documented utilizations of COBYLA in quantum simulation tasks highlight its efficacy in managing system noise — a frequent challenge in practical quantum computing applications. According to the research findings, COBYLA demonstrates a commendable robustness in maintaining the accuracy of the solution even when simulations are performed on noisy quantum processors. This capability is critical in my study as it entails executing algorithms on real quantum devices, where noise is an unavoidable factor.

Furthermore, the decision to implement COBYLA was also reinforced by its performance in reducing the number of iterations needed to achieve convergence in quantum circuit optimization, as observed in the reviewed studies. This efficiency in convergence not only saves computational resources but also enhances the feasibility of conducting more extensive simulations within practical time frames.

Finally, based on [35] and the given robustness and efficiency across different noise levels and optimization surface characteristics, COBYLA is recommended as a baseline optimizer for several reasons:

- It consistently delivers high performance with minimal requirement for parameter tuning.
- It adapts well to different problem landscapes, making it a versatile choice for hybrid quantum-classical algorithms.
- Its reliable performance in presence of noise aligns well with the demands of NISQ devices, ensuring that it remains a solid choice for practical applications.

Given these considerations — absence of gradient requirement, robustness against noise, and efficient convergence — COBYLA was deemed the optimal choice for the optimizer in my thesis work aimed at using quantum algorithms to predict peptide structures, ensuring both accuracy and practicality in the quantum computing landscape.

6.5 Choosing the Ansatz for the VQE's Simulations

For Quantum Machine Learning (QML), the choice of ansatz is crucial in determining the efficiency and accuracy of the models, especially when dealing with the inherent noise present in quantum hardware. For the experiments conducted in this thesis, the `RealAmplitudes` ansatz was employed as the ansatz for the VQE, due to its demonstrable robustness in noisy environments.

Rotation Gates

The `RealAmplitudes` circuit utilizes rotation gates, specifically *RY* gates, which rotate the qubits about the Y-axis. These rotations are parameterized by real-valued angles.

$$|\mathrm{RY}(\theta[i])| \quad \text{where } \theta[i] \text{ is a real number}$$

Entanglement

The circuit alternates layers of these rotation gates with entangling gates, typically *CX* (CNOT) gates, which create entanglement between pairs of qubits.

Real-Valued Parameters

The real-valued parameters in the *RY* gates ensure that the prepared quantum states have real amplitude components only. The mathematical form of an *RY* gate acting on a qubit does not introduce any imaginary components.

Applications

This characteristic is particularly useful in certain quantum computing applications, such as variational quantum algorithms for chemistry and machine learning, where it might be beneficial to work within the realm of real numbers due to either simplicity or problem constraints.

Except of the highlight from qiskit's documentation that the `RealAmplitudes` is suitable for chemistry, the decision to use it was guided by comprehensive evaluations reported in a study. In particular, [20] conducted an extensive analysis comparing different ansatzes, including `RealAmplitudes`, `TwoLocal`, and `PauliTwoDesign`, across various datasets and quantum conditions. Their research revealed that `RealAmplitudes` consistently yielded favorable results on noisy quantum computers.

Specifically, the study found that:

- **RealAmplitudes** performed significantly better in noisy settings for the KDD Cup and Rice datasets, while showing no performance degradation for the Glass Identification and Cover Type datasets in noise-affected environments.
- The stability of `RealAmplitudes` in terms of mean and standard deviation across different noise conditions was noted, making it a reliable choice for noisy quantum computations.

By integrating the `RealAmplitudes` ansatz, this thesis leverages a well-substantiated approach to enhance the performance of QML models under realistic quantum computing conditions. For this, I also chose L = 1 entangling layers.

6.6 Measuring the Capabilities of Quantum Ansatzes Using Density Matrices

This section describes the implementation details of a Python code designed to measure the expressive capabilities of quantum ansatzes. The code leverages Qiskit and other numerical libraries to perform these evaluations.

Overview

The goal of the code is to quantitatively assess the capabilities of quantum ansatzes by calculating their density matrices and comparing them to random unitary operations. The primary functions implemented include:

1. Generating random unitary matrices.
2. Calculating the Haar integral.
3. Evaluating parameterized quantum circuits (PQC).
4. Combining these elements to analyze the expressive capabilities of quantum ansatzes.

Generating Random Unitary Matrices

The `random_unitary` function creates Haar-distributed random unitary matrices that are uniformly distributed, making them ideal for generating random unitary operations.

```
def random_unitary(N):
    Z = np.random.randn(N, N) + 1.0j * np.random.randn(N, N)
    [Q, R] = sp.linalg.qr(Z)
    D = np.diag(np.diagonal(R) / np.abs(np.diagonal(R)))
    return np.dot(Q, D)
```

Code Snippet 6.1: The `random_unitary` function

Calculating Haar Integral

The `haar_integral` function calculates the Haar integral for a specified number of qubits and samples. It iterates through the samples, computes the tensor product (Kronecker product) of the generated states and their conjugates, and averages them to form the density matrix representation.

```
def haar_integral(num_qubits, samples):
    N = 2**num_qubits
    randunit_density = np.zeros((N, N), dtype=complex)
    zero_state = np.zeros(N, dtype=complex)
    zero_state[0] = 1

    for _ in range(samples):
        A = np.matmul(zero_state, random_unitary(N)).reshape(-1,1)
        randunit_density += np.kron(A, A.conj().T)

    randunit_density /= samples
    return randunit_density
```

Code Snippet 6.2: The `haar_integral` function

Evaluating Parameterized Quantum Circuits (PQC)

The `pqc_integral` function evaluates the PQC by assigning random parameters, running the circuit on a statevector simulator, and computing the resultant density matrix. It averages over multiple samples to obtain a representative density matrix of the circuit's expressive capability.

```python
def pqc_integral(num_qubits, ansatze, size, samples):
    N = num_qubits
    randunit_density = np.zeros((2**N, 2**N), dtype=complex)

    for _ in range(samples):
        params = np.random.uniform(-np.pi, np.pi, size)
        ansatz = ansatze.assign_parameters(params)

        result = qiskit.execute(ansatz, backend=Aer.get_backend('
    statevector_simulator')).result()
        U = result.get_statevector(ansatz)
        U = U.reshape(-1,1)

        randunit_density += np.kron(U, U.conj().T)

    return randunit_density / samples
```

Code Snippet 6.3: The `pqc_integral` function

Execution and Combining the Results

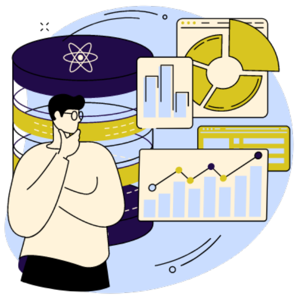
The overall procedure involves loading the ansatz, calculating both the Haar integral and the PQC density matrix, and then comparing these matrices to assess the ansatz's capabilities.

```python
ansatz = load_ansatz("ansatz.qasm")
print(ansatz)

# Example usage
num_qubits = ansatz.num_qubits
samples = 100
size = len(ansatz.parameters)

haar_density = haar_integral(num_qubits, samples)
pqc_density = pqc_integral(num_qubits, ansatz, size, samples)
```

Code Snippet 6.4: The `random_unitary` function



Chapter 7

VQE vs QAOA (p=1) - Implementation's Results

7.1 Peptide YGGFM

Figure 7.1 provides a visual representation of the 3D structure of the YGGFM peptide as determined by Nuclear Magnetic Resonance (NMR) spectroscopy in solution [5]. The two panels display the peptide from different angles, highlighting the spatial arrangement of its five amino acid residues: Tyrosine (TYR), Glycine (GLY), Phenylalanine (PHE), and Methionine (MET) - see Appendix A for the 1-letter symbolization. The figure emphasizes the peptide's non-linear conformation, showcasing how the amino acid chain folds in a specific manner.

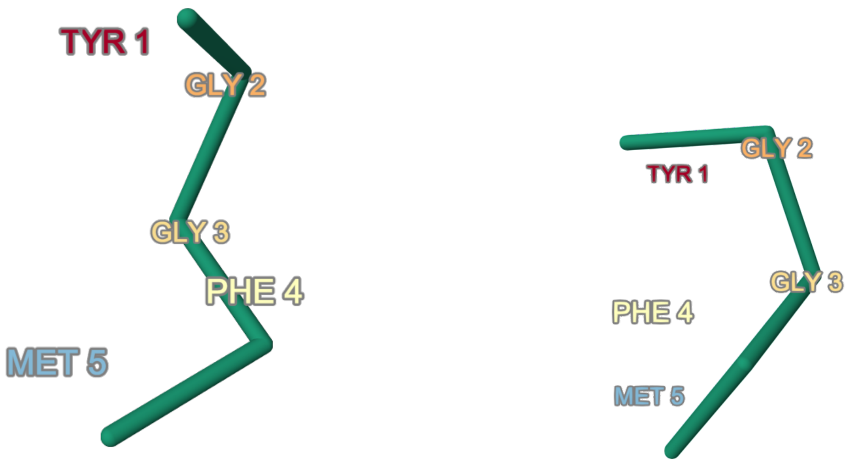
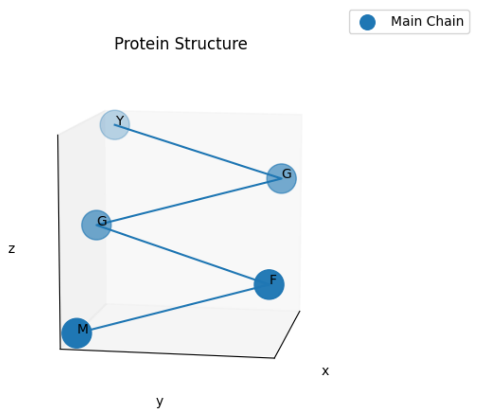


Figure 7.1: NMR solution structure of peptide YGGFM

We now have what we obtained from the hybrid quantum-classical machine learning implementations. The following figures (Fig. 7.2 and 7.3) present two perspectives of the predicted 3D structure of the YGGFM peptide obtained using the VQE algorithm and QAOA respectively. The blue spheres represent the peptide's main chain, highlighting the spatial arrangement of the amino acid backbone as determined by the calculations.

- **VQE Implementation** Using VQE we got the structures:



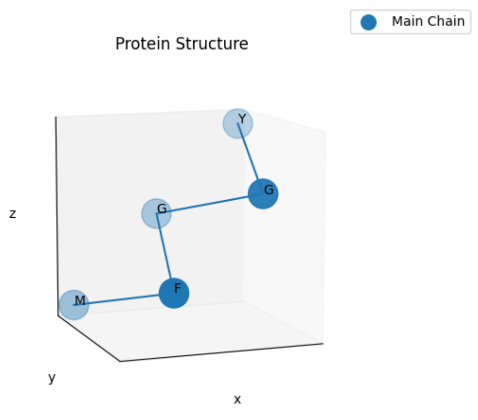
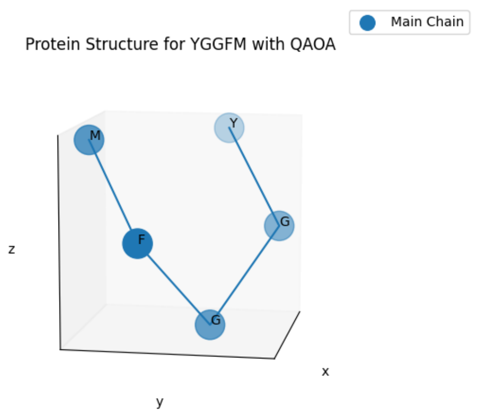


Figure 7.2: 3D structure of peptide YGGFM using VQE

- **QAOA Implementation** Using QAOA we got the structures:



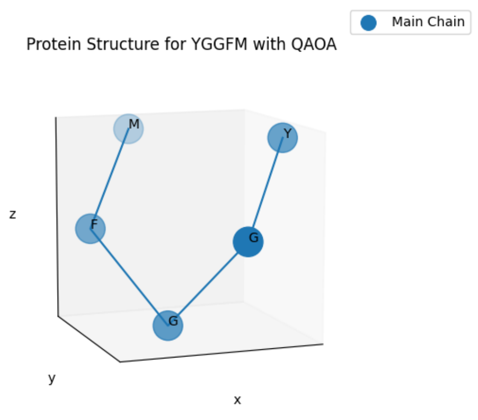
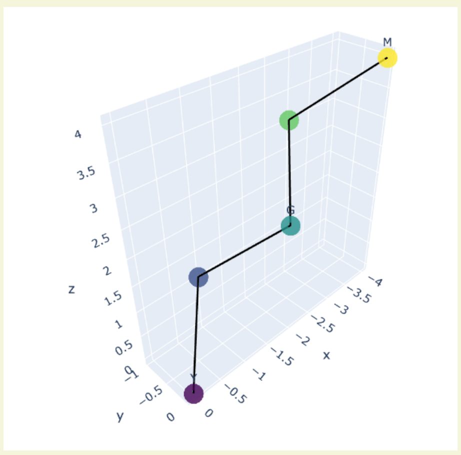


Figure 7.3: 3D structure of peptide YGGFM using QAOA

QAOA with 10 layers of alternating unitary operators

Using QAOA with $p = 5$ we got the structures:



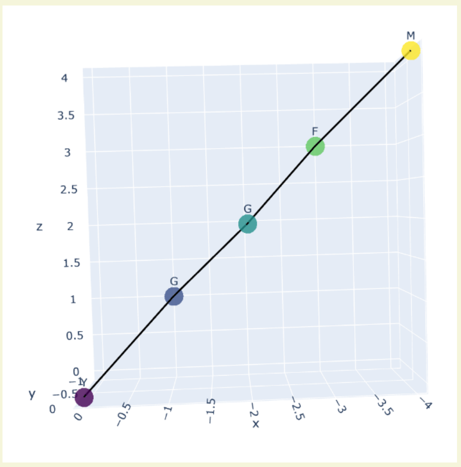


Figure 7.4: 3D structure of peptide YGGFM using simulator

- **Comparison of the Two Methods**

  The angles shown in (Fig. 7.5), between the planes of the NMR solution , correspond to the dihedral angles between adjacent amino acids in the peptide chain of YGGFM. The also figure highlights the angles formed between the surfaces defined by triplets of consecutive amino acids, providing insights into the peptide's local curvature and folding patterns.

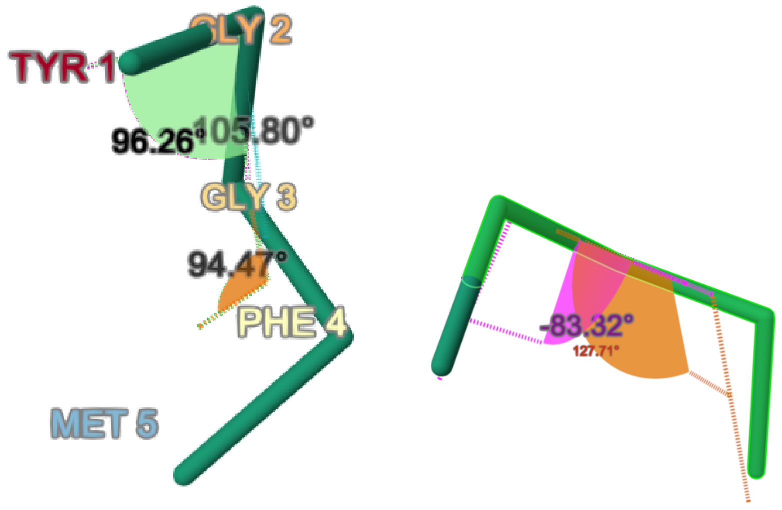


Figure 7.5: 3D solution of peptide YGGFM using NMR - angles between amino acids (left-hand side) and between planes (right-hand side)

Table 7.1.1 shows all the calculations from the implementations of the models, to compare the results.

Table 7.1.1: Comparison of 3D Structure Predictions for peptide YGGFM using VQE and QAOA

Parameter	**VQE**	**QAOA**	**NMR solution**
Evals	37	31	
penalties	(15, 400, 40)	(10, 100, 1000)	
Expressibility	0.133	0.135	
Overlap with ground state	0.129	0.134	
XYZ Structure	Y: (0.0, 0.0, 0.0) G: (0.58, 0.58, -0.58) G: (1.16, 0.0, -1.16) F: (1.73, 0.58, -1.73) M: (2.31, 0.0, -2.31)	Y: (0.0, 0.0, 0.0) G: (0.58, 0.58, -0.58) G: (1.15, 0.0, -1.15) F: (1.73, -0.58, -0.58) M: (1.16, -1.16, 0.0)	
Angles in Structure	109.5, 109.5, 109.5	109.5, 109.5, 109.5	96.3, 105.8 94.5
Angles between planes	0.0, 0.0	60.0, 60.0	83.2, 72.29

Based on the provided comparison data for the 3D structure predictions of oxytocin using the Variational Quantum Eigensolver (VQE), Quantum Approximate Optimization Algorithm (QAOA), and experimentally obtained NMR solution structure, we can evaluate their performance in several aspects. Below is a detailed review considering the outcomes, algorithmic efficiency, and structural accuracy:

Evaluation Criteria:

– **Number of Evaluations (Evals):**
  - VQE: 37
  - QAOA: 31
  Fewer evaluations indicate a more efficient optimization process. QAOA required considerably fewer evaluations compared to VQE, suggesting higher efficiency.
– **Penalty Parameters:**
  - Penalties for VQE didn't play important role - we had this same output for many different values.
  - However, the penalty that penalize local overlap between beads (penalty_1) in QAOA needs to be higher than the other two, potentially indicating stricter constraint enforcement in QAOA which could impact the quality of the solution. Otherwise, we have overlaps for some of the amino-acids.

Structural Outcome Analysis:

– **Angles in Structure:**
  - Both VQE and QAOA maintained standard tetrahedral angles (109.5 degrees) which is typical in many molecular structures showing good basic geometric handling.

- The NMR solution structure shows a variety of angles (96.3, 105.8, 94.5 degrees), indicating a more complex structure which neither quantum method replicated accurately.

– **Angles between Planes:**
  - VQE and QAOA maintained consistent angles of 0 degree and 60 degrees respectively, reflecting simplicity in their structural predictions.
  - The NMR solution demonstrates varied angles (83.2, 72.29 degrees), again signaling more intricate molecular interactions that quantum predictions failed to capture. QAOA is closer to this curved structure than VQE (which provided a structure placed on a plane).

7.2 Peptide KLVFFA

Fig. 7.6 shows the NMR solution as obtained from [5] for KLVFFA peptide:

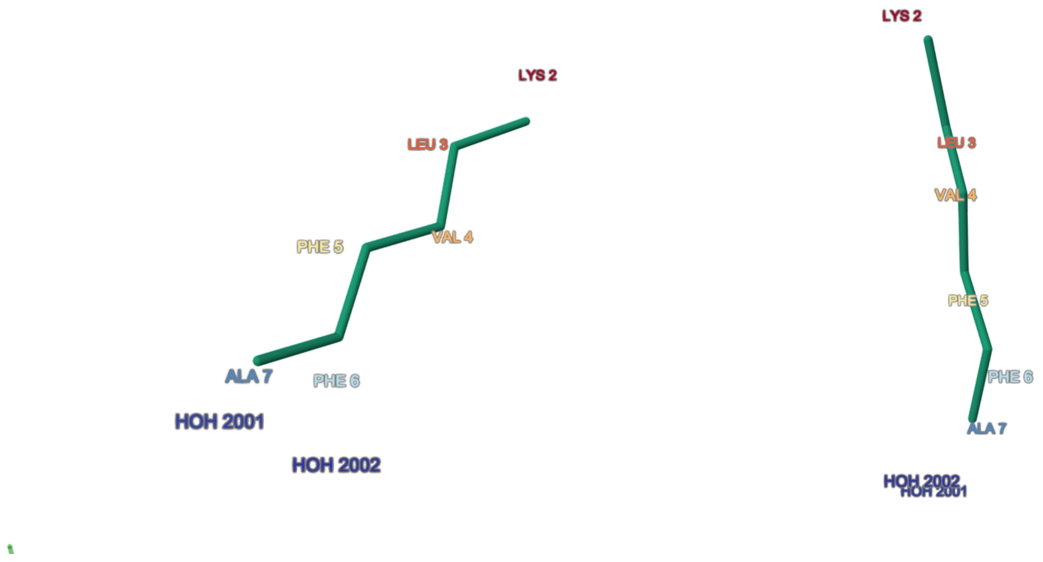


Figure 7.6: NMR solution structure of peptide KLVFFA

The following figures (Fig. 7.7 and 7.3) present two perspectives of the predicted 3D structure of the KLVFFA peptide obtained using the VQE algorithm and QAOA respectively.

- **VQE Implementation** Using VQE we got the structures:

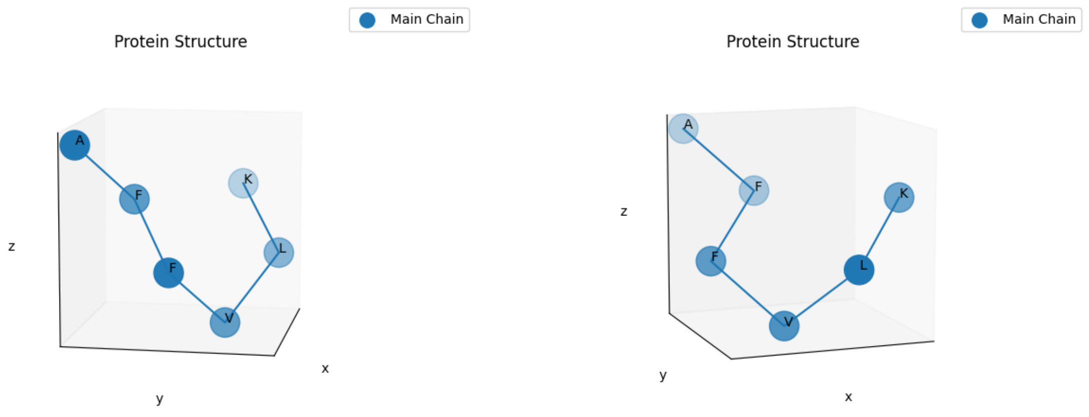


Figure 7.7: 3D structure of peptide KLVFFA using VQE

- **QAOA Implementation** Using QAOA we got the structures:

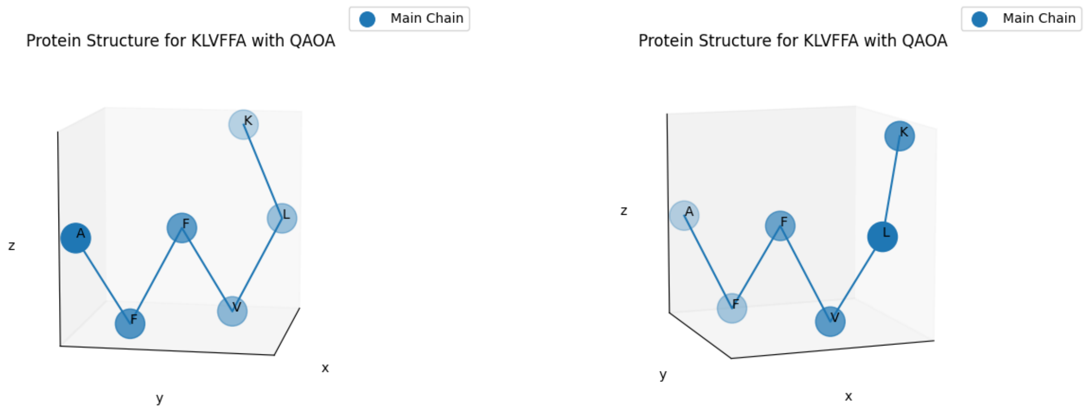


Figure 7.8: 3D structure of peptide KLVFFA using QAOA

QAOA with 10 layers of alternating unitary operators

Using QAOA with $p = 5$ we got the structures:

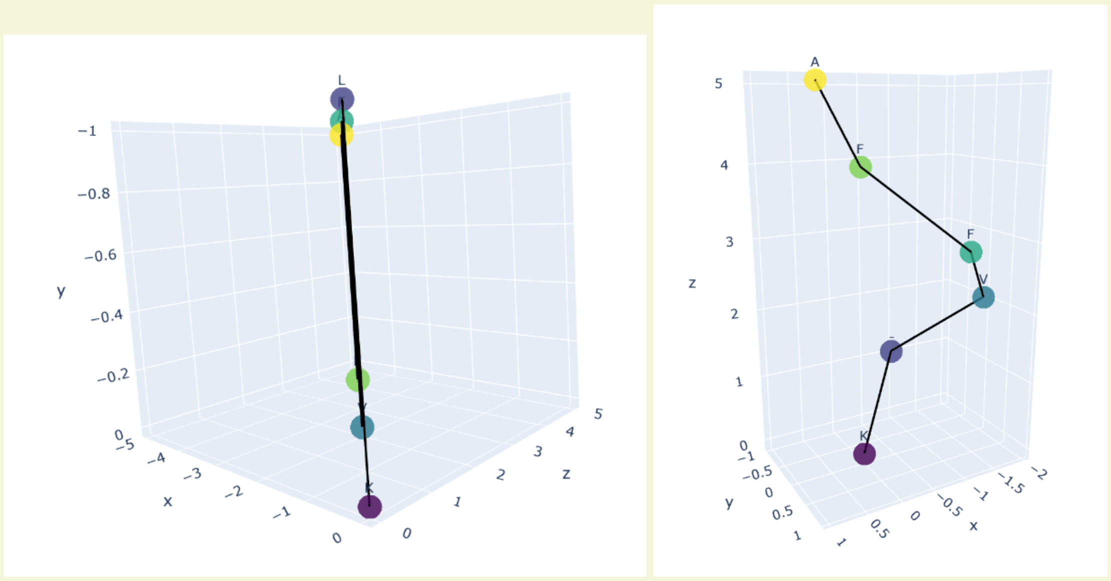


Figure 7.9: 3D structure of peptide KLVFFA using simulator

- **Comparison of the Two Methods**

The angles shown between the following planes of the NMR solution (Fig. 7.10), correspond to the dihedral angles between adjacent amino acids in the peptide chain of KLVFFA. As in the previous example, the figure highlights the angles formed between the surfaces defined by triplets of consecutive amino acids, providing insights into the peptide's local curvature and folding patterns.

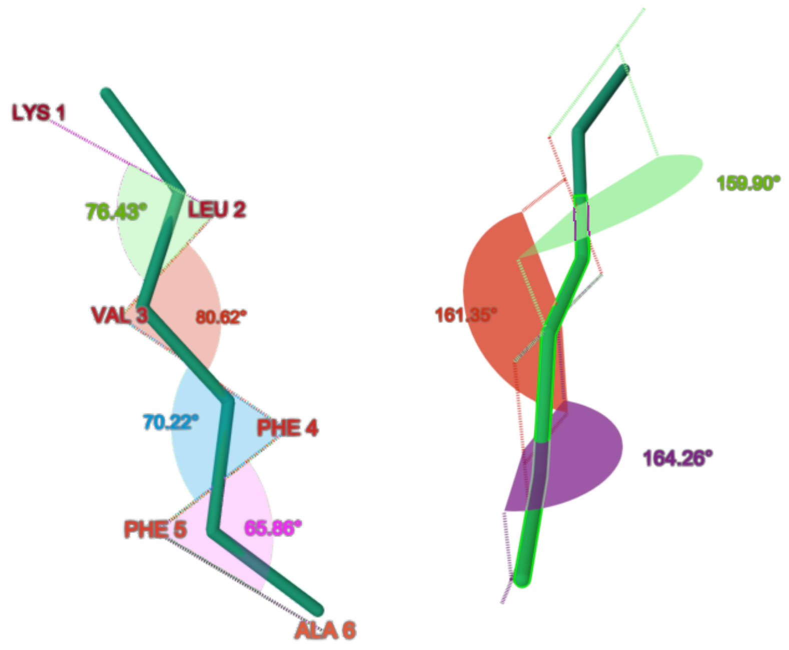


Figure 7.10: 3D solution of peptide KLVFFA using NMR - angles

Table 7.2.1 shows all the calculations from the implementations of the models, in order to compare them.

Table 7.2.1: Comparison of 3D Structure Predictions for peptide KLVFFA using VQE and QAOA

Parameter	**VQE**	**QAOA**	**NMR solution**
Evals	84	29	
penalties	(10, 100, 400)	(10, 100, 1000)	
Expressibility	0.143	0.138	
Overlap with ground state	0.025	0.011	
XYZ Structure	K: (0.0, 0.0, 0.0) L: (0.58, 0.58, -0.58) V: (1.16, 0.0, -1.16) F: (1.73, 0.58, -1.73) F: (1.16, 1.16, -2.31) A: (0.58, 1.73, -1.73)	K: (0.0, 0.0, 0.0) L: (0.58, 0.58, -0.58) V: (1.15, 0.0, -1.15) F: (1.73, -0.58, -0.58) F: (2.31, -1.15, -1.15) A: (2.88, -1.73, -0.58)	
Angles in Structure	109.5, 109.5, 109.5, 109.5	109.5, 70.5, 70.5, 70.5	76.4, 80.6, 70.2, 65.9
Angles between planes	60.0, 60.0, 0.0	60.0, 0.0, 0.0	40.1, 38.6, 38.6

Evaluation Criteria:

– **Number of Evaluations (Evals):**
  - VQE: 84
  - QAOA: 29
  Fewer evaluations indicate a more efficient optimization process. QAOA required considerably fewer evaluations compared to VQE, suggesting higher efficiency.

– **Penalty Parameters:**
  - Penalties for backbone (penalty_back) and chiral constraints (penalty_chiral) are the same for both VQE and QAOA (10 and 100 respectively). For KLVFFA the values of penalties gave 2 different structures - the one showing here and one with overlap (when penalty_back and penalty_1 were of the same order of magnitude).
  - The penalty for additional constraints (penalty_1) is higher in QAOA (1000) compared to VQE (400). KLVFFA had a variety of outputs when run with QAOA. The angles in the structure were always 109.5 degrees and the penalty_1 had to be at least 5 times bigger than the other two so that we won't have overlaps. A case - $penalties = (2, 4, 20)$ - gave the same output with VQE, and the other cases had a curvature that does not match the image we have for the peptide

Structural Outcome Analysis:

– **Angles in Structure:**
  - VQE and QAOA maintained again standard tetrahedral angles (109.5 degrees), while the NMR solution structure shows a variety of angles (76.4, 80.6, 70.2, 65.9 degrees).

– **Angles between Planes:**

- VQE and QAOA maintained consistent angles of 60 degrees and 0 degree, while the NMR solution demonstrates angles (40.1, 38.6, 38.6 degrees). - QAOA was closer to the zig-zag structure that the peptide has.

7.3 Peptide CYIQNCPLG

We have the following NMR solution (Fig. 7.11) as obtained from [5] for CYIQNCPLG peptide:

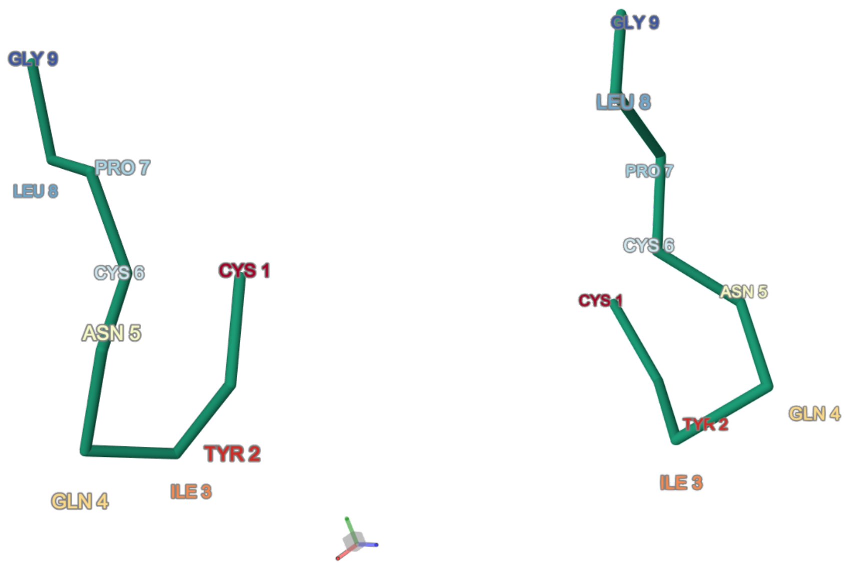


Figure 7.11: NMR solution structure of peptide CYIQNCPLG

The following figures (Fig. 7.12 and 7.13) present two perspectives of the predicted 3D structure of the CYIQNCPLG peptide obtained using the VQE algorithm and QAOA respectively.

- **VQE Implementation** Using VQE we got the structures:

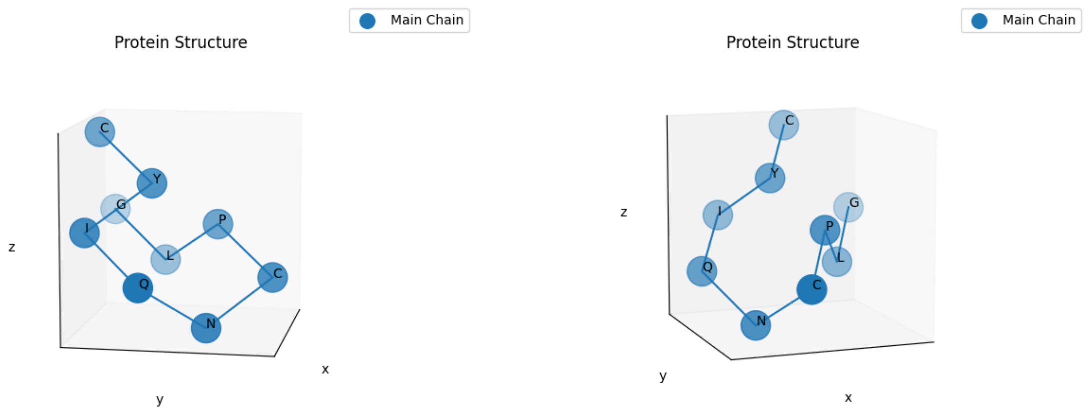


Figure 7.12: 3D structure of peptide CYIQNCPLG using VQE

- **QAOA Implementation** Using QAOA we got the structures:

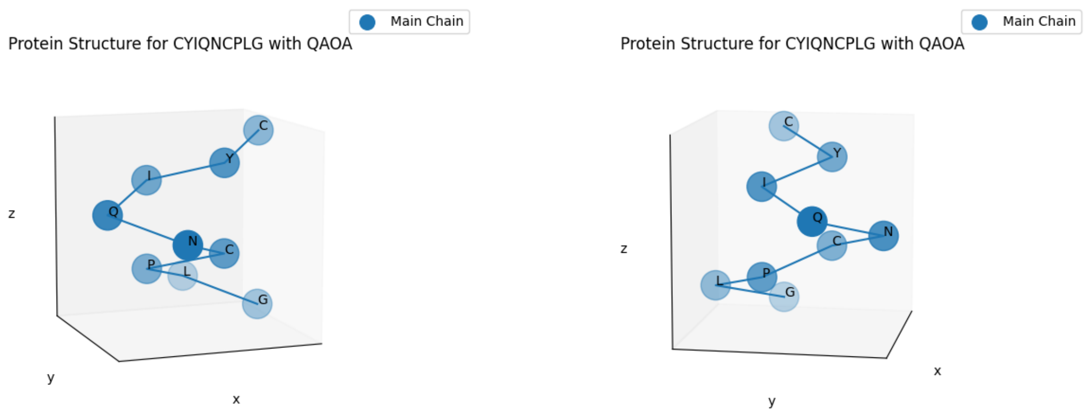


Figure 7.13: 3D structure of peptide CYIQNCPLG using QAOA

- **Comparison of the Two Methods**

  The angles shown between the following planes of the NMR solution (Fig. 7.14), correspond to the dihedral angles between adjacent amino acids in the peptide chain. Additionally, the figure highlights the angles formed between the surfaces defined by triplets of consecutive amino acids, providing insights into the peptide's local curvature and folding patterns.

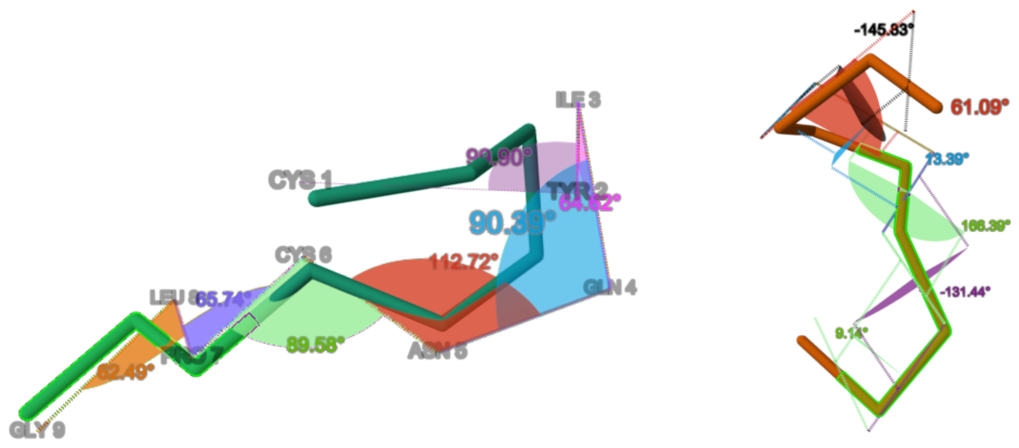


Figure 7.14: 3D solution of peptide CYIQNCPLG using NMR - angles between amino acids (left-hand side) and between planes (right-hand side)

Table 7.3.1 shows all the calculations from the implementations of the models, to compare the results.

Table 7.3.1: Comparison of 3D Structure Predictions for Oxytocin using VQE and QAOA

Parameter	VQE	QAOA	NMR solution
Evals	100 (maxiter)	29	
penalties	(10, 8, 100)	(20, 40, 200)	
Expressibility	ArrayMemoryError	ArrayMemoryError	
Overlap with ground state	ArrayMemoryError	ArrayMemoryError	
XYZ Structure	C: (0.0, 0.0, 0.0) Y: (0.58, 0.58, -0.58) I: (1.16, 0.0, -1.16) Q: (1.73, 0.58, -1.73) N: (1.16, 1.16, -2.31) C: (0.58, 1.73, -1.73) P: (0.0, 1,16, -1.154) L: (-0.58, -0.58, -1.73) G: (-1.16, 0.0, -1.16)	C: (0.0, 0.0, 0.0) Y: (0.58, 0.58, -0.58) I: (1.15, 0.0, -1.16) Q: (1.73, 0.58, -1.73) N: (1.16, 1.16, -2.31) C: (0.58, 0.58, -2.89) P: (1.15, 0.0, -3.46) L: (0.58, -0.58, -4.04) G: (0.0, 0.0, -4.62)	
Angles in Structure	109.52, 109.52, 109.49, 109.52, 109.52, 109.4, 109.42	109.64, 109.37, 109.55, 109.44, 109.43, 109.43, 109.86	99.9, 64.6, 90.4 112.7, 89.6, 65.7, 62.5
Angles between planes	0.0, 60.0, 60.0 60.0, 60.0, 0.0	0.0, 60.0, 60.0 60.0, 0.0, 60.0	34.2, 61.1, 13.4 13.6, 48.6, 9.14

Evaluation Criteria:

- **Number of Evaluations (Evals):**
  - VQE: 100 (maxiter)
  - QAOA: 29
  QAOA's few evaluations suggest high efficiency, while VQE couldn't converge.
- **Penalty Parameters:**
  - Penalties for VQE and QAOA are of the same order of magnitude.

Structural Outcome Analysis:

- **Angles in Structure:**
  - Both VQE and QAOA maintained standard tetrahedral angles (109.5 degrees) which is typical in many molecular structures showing good basic geometric handling.
  - The NMR solution structure shows a variety of angles (99.9, 64.6, 90.4, 112.7, 89.6, 65.7, 62.5 degrees), indicating a more complex structure which neither quantum method replicated accurately.
- **Angles between Planes:**
  - VQE and QAOA maintained consistent angles of 60 degrees and a 0 degree, reflecting simplicity in their structural predictions.
  - The NMR solution demonstrates varied angles (34.2, 61.1, 13.4, 13.6, 48.6, 9.14 degrees),

again signaling more intricate molecular interactions that quantum predictions failed to capture.

7.4 Comments on the Results

7.4.1 General Comments

Algorithm Efficiency

Across all three peptides, QAOA demonstrated higher optimization efficiency, requiring fewer evaluations compared to VQE.

Optimization Challenges

VQE faced challenges in converging, particularly for more complex peptides like CYIQNCPLG. And, in general, we can observe that as size of the amino-acid chain increases, so does the difference in the number of iterations it took for the two methods to converge - with VQE continually needing more.

Accuracy Trade-offs

While both quantum methods can approximate molecular structures, their predictions lack the finer details and complexity captured by NMR techniques, particularly in terms of angle diversity.

7.4.2 Geometric Limitations

Lattice Contraints and Restrictions

The geometric constraints inherent in the tetrahedral lattice model present notable implications for the accuracy and representation of protein structures. The model's fixed bond angles and discrete movement directions fundamentally limit the range of possible conformations the protein can assume, consequently impacting the model's ability to accurately simulate real protein behavior.

Tetrahedral Geometry and Its Effects in Simulation

Within the tetrahedral lattice model, each point, representing an amino acid, connects to four neighboring points, forming a tetrahedral shape with bond angles at 109.5°. Moreover, this configuration inherently restricts the possible spatial orientations, particularly the planes formed by sequential triads of amino acids. As a result, the angles between these planes are confined primarily to 0° and 60°, which directly stems from the fixed bond angles in tetrahedral geometry. This primary constraint results in the inability of the simulation to capture the continuous range of angles.

7.5 Anticipated Outcomes and Expectations

Despite attempts to enhance the model by alternating lattice sets P and Q, which aimed to prevent spatial overlaps and broaden the conformational space, the expected improvement in the variety of movement pathways and folding patterns has not been realized effectively. The following outlines the key areas where the model falls short:

- **Limited Conformational Diversity:** The alternation between lattice sets P and Q, while theoretically increasing flexibility, still does not offer sufficient variety to accurately reflect the dynamic nature of protein folding. The rigid geometric constraints imposed by the tetrahedral angles significantly inhibit the model's adaptability, failing to represent the nuanced angular variations observed in actual protein molecules.
- **Model Realism:** The model struggles to mirror realistic folding patterns, often resulting in repetitive or physically unrealistic conformations that do not correspond well with empirical protein structures. While the model may achieve computational efficiency, the trade-off is notably evident in the accuracy and realism of the simulation outcomes, which remain inadequate.

The model's limitations in capturing the complex, continuous nature of protein folding suggest the need for further refinement or alternative modeling strategies to achieve more realistic and accurate simulations.

7.6 Potential for Future Refinement

From what I observed, the predicted output maintains with high accuracy the sign of the concavity. This constraint could be applied to train machine learning model to predict feasible conformations for each amino acid sequence. The hybrid model will help identify folding patterns that are consistent with the constraints imposed by the tetrahedral geometry and maintain the correct sign of concavity, thereby excluding unfeasible conformations.

Additionally, the thing that VQE and QAOA have similar expressibility and entanglement capability but QAOA converges faster, suggests that the QAOA ansatz is better suited for the specific problem. This is because the QAOA ansatz is designed to naturally explore the solution space of combinatorial optimization problems. Thus, the QAOA seems to be better choice for the prediction of the 3D structure of the proteins.

Chapter 8

QAOA (p=5) - Simulators vs QPUs

The research presented in this chapter focuses on solving the 3D folding structure of proteins using the QAOA and the tetrahedral encoding with MJ interaction model. In contrast to leveraging pre-built functions from the Qiskit framework, I have taken a distinctive approach for educational purposes, and because the previous used libraries have been deleted: crafting all essential components from scratch. This methodology allowed me to gain a profound and nuanced understanding of the underlying quantum mechanics and computational processes involved.

To achieve this, I manually developed the Hamiltonian necessary for the optimization problem using `pyomo`. Subsequently, I constructed the quantum circuits for the QAOA implementation as are needed by the Qiskit framework. Furthermore, I progressed to the stage of creating and dispatching computational jobs to execute these circuits on real IBM's quantum devices.

8.1 The Code

Based on the general outline for hybrid algorithms (as we saw in Sec. 6.2) the code that implements these, using tetrahedral lattice, turn encoding and MJ interaction model (as discussed in 5.5.1) is:

```python
import numpy as np
import pyomo.core as pyo

# Example interaction energy dictionary and protein sequence
interaction_energy = {
    'A': {'A': -2.72, 'P': 0.13, 'G': 0.18, 'R': 0.11},
    'P': {'A': 0.13, 'P': -1.75, 'G': 0.20, 'R': 0.23},
    'G': {'A': 0.18, 'P': 0.20, 'G': -2.24, 'R': 0.30},
    'R': {'A': 0.11, 'P': 0.23, 'G': 0.30, 'R': -1.55}
}

def folding_hamiltonian(main_chain: str) -> pyo.ConcreteModel:
    model = pyo.ConcreteModel("protein_folding")
    N = len(main_chain)

    Ninteraction = int((N - 5) * (N - 4) / 2) if N > 5 else 0

    model.f = pyo.Var(range(2 * (N - 3)), domain=pyo.Binary) # 2 qubits to
encode each amino acid
    model.interaction = pyo.Var(range(Ninteraction), domain=pyo.Binary)
    f_array = np.array(list(model.f.values()))
    interaction_array = np.array(list(model.interaction.values()))

    a = np.array([1, 0, 0, 1])
    full_f_array = np.append(a, f_array)

    T = lambda i, j: (1 - (full_f_array[2* i] - full_f_array[2 * j]) ** 2) * (
        1 - (full_f_array[2 * i + 1] - full_f_array[2 * j + 1]) ** 2
    )
    L = 50
    model.Hgc = sum(L * T(i, i + 1) for i in range(N - 2)) # Hydrophobicity term, summing over all pairs of adjacent amino acids in the chain and calculating the hydrophobicity term

    # Convert {0,1}^2 to 4 functions, each giving 1 for one vector and 0 for the others:
    fun0 = lambda i, j: (1 - full_f_array[i]) * (1 - full_f_array[j])
    fun1 = lambda i, j: full_f_array[i] * (1 - full_f_array[j])
    fun2 = lambda i, j: full_f_array[j] * (1 - full_f_array[i])
    fun3 = lambda i, j: full_f_array[i] * full_f_array[j]

    # Calculate distance between i, j amino acids:
    d_units_0 = lambda i, j: sum(
        [((-1) ** k) * fun0(2 * k, 2 * k + 1) for k in range(i, j)]
    )
    d_units_1 = lambda i, j: sum(
        [((-1) ** k) * fun1(2 * k, 2 * k + 1) for k in range(i, j)]
    )
    d_units_2 = lambda i, j: sum(
        [((-1) ** k) * fun2(2 * k, 2 * k + 1) for k in range(i, j)]
    )
    d_units_3 = lambda i, j: sum(
        [((-1) ** k) * fun3(2 * k, 2 * k + 1) for k in range(i, j)]
    )
    d = lambda i, j: (
        (d_units_0(i, j)) ** 2
        + (d_units_1(i, j)) ** 2
        + (d_units_2(i, j)) ** 2
        + (d_units_3(i, j)) ** 2
    )

    def h(i, j):
        L1, L2 = 5, 3  # Penalty factors
        penalty = L1*(d(i, j) - 1) ** 2 + L2*(
            (2 - d(j - 1, i)) ** 2
            + (2 - d(j + 1, i)) ** 2
            + (2 - d(i - 1, j)) ** 2
            + (2 - d(i + 1, j)) ** 2
        )
        amino_i = main_chain[i]
        amino_j = main_chain[j]
        interaction_strength = interaction_energy.get((amino_i, amino_j), interaction_energy.get((amino_j, amino_i), 0))
        interaction_idx = sum([N - 5 - k for k in range(0, i + 1)]) - (N - j)
        interaction_term = interaction_array[interaction_idx] * interaction_strength +penalty # Use interaction strength directly
        return interaction_term

    model.Hint = sum(h(i, j) for i in range(N - 5) for j in range(i + 5, N))

    # Objective function combining Hgc and Hint
    model.obj = pyo.Objective(expr=model.Hgc + model.Hint, sense=pyo.minimize)

    return model

# Define your protein sequence
my_protein = "APGPR"
protein_model = folding_hamiltonian(my_protein)

protein_model.pprint()

# Extract and print the objective function
objective_func = protein_model.obj.expr
print("\nObjective Function Expression:\n")
print(objective_func)

import numpy as np

# Define the energy computation as per the objective function
def compute_energy(f):
    term1 = 50 * (1 - (-f[0])**2) * (1 - (1 - f[1])**2)
    term2 = 50 * (1 - (f[0] - f[2])**2) * (1 - (f[1] - f[3])**2)
    return term1 + term2

# Convert a binary string (measured state) to a list of integers
def bitstring_to_list(bitstring):
    return [int(bit) for bit in bitstring]

# Compute the average energy based on the measurement counts
def average_energy(measurement_counts):
    total_energy = 0
    total_counts = sum(measurement_counts.values())

    for bitstring, count in measurement_counts.items():
        # Convert the bitstring to f variables
        f = bitstring_to_list(bitstring)

        # Compute energy for this configuration
        energy = compute_energy(f)

        # Weight the energy by the count
        total_energy += energy * count

    # Compute the average energy
    average_energy = total_energy / total_counts
    return average_energy

def avg(quasi):
    total_energy = 0.0
    num_quasi_distributions = len(quasi[0])
    quasi_dist = quasi[0]

    for key, value in quasi_dist.items():
            print(f"Key: {key}, Value: {value}")

            # Convert integer key to a binary string representation
            # The format specifier `#0{n}b` ensures that we get the binary
    string representation with leading zeros.
            # The length of the bitstring is chosen by considering the maximum
    key value's bit length
```

```python
            #max_bits = max(len(bin(k)) - 2 for k in quasi_dist.keys())  # Determine the maximum bits required
            bitstring = format(key, f'04b')
            f = bitstring_to_list(bitstring)
            # Calculate the parity contribution: +1 if even number of '1's, -1 if odd number of '1's
            energy = compute_energy(f)
            # Contribution to the total energy
            total_energy += energy*value

    # Average energy calculation
    average_energy = total_energy if num_quasi_distributions > 0 else 0.0
    print(f"Average Energy: {average_energy}")
    return average_energy

import re
import sympy as sp
from qiskit.quantum_info import Pauli, SparsePauliOp

def binary_to_pauli_z(index):
    """Converts a binary variable f to a Pauli-Z operator term."""
    return (1 - sp.Symbol(f'Z{index}')) / 2

def classical_to_quantum_hamiltonian(objective_function):
    terms = re.split(r'\s*\+\s*', objective_function.strip())
    pauli_list = []
    coeff_list = []

    for term in terms:
        coeff_term_match = re.match(r'(\d+)\s*\*\s*\((.*)\)', term)
        if coeff_term_match:
            coeff, expr = coeff_term_match.groups()
            coeff = float(coeff)

            # Replace f[i] with corresponding Pauli-Z expressions using formatted strings
            for i in range(4):  # Assuming binary variables f[0] to f[3]
                expr = expr.replace(f'f[{i}]', f'({binary_to_pauli_z(i)})')

            try:
                # Convert expression using sympy with proper formatting
                expression = sp.sympify(expr, evaluate=False)
                pauli_coeff = sympy_to_pauli_op(expression, coeff)
                pauli_list.extend(pauli_coeff["paulis"])
                coeff_list.extend(pauli_coeff["coeffs"])
            except Exception as e:
                print(f"Failed to parse expression correctly: {expr}")
        else:
            print(f"Failed to parse term correctly: {term}")

    hamiltonian = SparsePauliOp(pauli_list, coeff_list)
    return hamiltonian


def sympy_to_pauli_op(expression, coeff):
    """Convert a sympy expression to a Qiskit SparsePauliOp."""
    symbols = sorted(expression.free_symbols, key=lambda x: str(x))
    pauli_labels = []

    for symbol in symbols:
        if 'Z' in str(symbol):
            z_index = int(re.findall(r'\d+', str(symbol))[0])
            pauli_label = f'{"I"*z_index}Z{"I"*(3-z_index)}'  # Assuming 4
    qubits total
            pauli_labels.append(pauli_label)

    pauli_coeff = {"paulis": [], "coeffs": []}

    for label in pauli_labels:
        pauli = Pauli(label)
        pauli_coeff["paulis"].append(pauli)
        pauli_coeff["coeffs"].append(coeff)

    return pauli_coeff

# objective function
objective_function = "50*((1 - (- f[0])**2)*(1 - (1 - f[1])**2)) + 50*((1 - (f
    [0] - f[2])**2)*(1 - (f[1] - f[3])**2))" # as retrieved from the Pyomo model
cost_function = classical_to_quantum_hamiltonian(objective_function)
print(cost_function)

from qiskit_ibm_runtime import QiskitRuntimeService, Sampler

QiskitRuntimeService.save_account(channel="ibm_quantum", token="your_token_here"
    ,
                                   instance="ibm-q/open/main",overwrite=True)

service = QiskitRuntimeService()
print("Service account saved.")

job_name = "APGPR"

# Holds job references for each backend
jobs = []

p = 5
num_qubits = 4
mixer_hamiltonian = sum(SparsePauliOp.from_list([(X, 1.0)]) for X in ['IIIX', '
    IIXI', 'IXII', 'XIII'])

# import the COBYLA optimizer
from qiskit_algorithms.optimizers import COBYLA
optimizer = COBYLA(maxiter=100)

from qiskit import QuantumCircuit
from qiskit.circuit import Parameter
from qiskit.quantum_info import SparsePauliOp
from qiskit.circuit.library import PauliEvolutionGate
import matplotlib.pyplot as plt

def apply_pauli_terms(qc, term, angle):
    evolved_op = PauliEvolutionGate(term, time= angle)
    qc.append(evolved_op, qc.qubits)


def qaoa_ansatz(num_qubits, p, cost_hamiltonian, mixer_hamiltonian):
    # Define parameters for gamma (cost) and beta (mixer) angles
    gammas = [Parameter(f'_{i}') for i in range(p)]
    betas = [Parameter(f' _{i}') for i in range(p)]

    # Initialize quantum circuit with all qubits in superposition state
    qc = QuantumCircuit(num_qubits)
    qc.h(range(num_qubits))

    # Apply p layers of cost and mixer Hamiltonians
    for i in range(p):
        # Apply cost Hamiltonian layer
        apply_pauli_terms(qc, cost_hamiltonian, gammas[i])

        # Apply mixer Hamiltonian layer
        apply_pauli_terms(qc, mixer_hamiltonian, betas[i])
    qc.measure_all()
    return qc, gammas, betas

from qiskit_ibm_runtime import Session
from qiskit.circuit import Parameter
from qiskit import transpile

qaoa, gamma_params, beta_params = qaoa_ansatz(num_qubits, p, cost_function,
    mixer_hamiltonian)
gamma_values = [np.random.uniform(0, np.pi) for _ in range(p)]
beta_values = [np.random.uniform(0, np.pi) for _ in range(p)]
i_params = np.array(gamma_values + beta_values)

def objective_f(params):
        #backend = service.get_backend('ibm_brisbane')
        backend = service.least_busy(operational=True, simulator=False)
        # create a new circuit and initialize the qubits in superposition
        qc = QuantumCircuit(num_qubits)
        qc.h(range(num_qubits))

        bound_qaoa_circuit = qaoa.assign_parameters(params)
        qc.compose(bound_qaoa_circuit, inplace=True)
        # draw the circuit
        qc.draw('mpl')
        # print the circuit
        print(qc)
        transpiled_qc = transpile(qc, backend=backend)
        inputs = {
            'circuits': transpiled_qc,
        }
        program_id = 'circuit-runner'

        with Session(service= service, backend = backend) as session:
            job = session.run(program_id = program_id, inputs = inputs)
            print(f"Wainting for job to finish...")
            job.wait_for_final_state()
            result = job.result()
            counts = result.get_counts()
            print(f"Counts:", counts)
            energy = average_energy(counts)
            print(f"Real device energy: {energy}")

        return energy
            # optimize the QAOA circuit with simulator

# optimize the QAOA circuit with Cobyla
```

```python
optimizer = COBYLA(maxiter=10)
# use minimize method to find the optimal parameters
result = optimizer.minimize(fun=objective_f , x0=i_params)
print(result)
```

Code Snippet 8.1: The Implementation of QAOA using tetrahedral lattice, turn encoding and MJ interaction model

Folding Hamiltonian Function

The primary function, `folding_hamiltonian(main_chain)`, initializes the Pyomo model to represent the protein folding problem.

- `model.f`: Defines binary variables representing qubits encoding directions for each amino acid (two qubits per amino acid, excluding the first three acids which are arbitrarily set).
- `model.interaction`: Defines binary variables for non-trivial amino acid interactions.

$$\texttt{Ninteraction} = \frac{(N-5)*(N-4)}{2} \quad \text{if } N > 5 \text{ else } 0$$

- `model.Hgc`: Calculates the geometrical constraints term by summing the penalties for adjacent amino acids folding back.
- `model.Hint`: Computes the interaction terms for the Hamiltonian, integrating penalty factors for distances between non-trivial interacting amino acids.
- `model.obj`: Defines the objective function, combining `Hgc` and `Hint`, set to minimize the folding energy.

The function returns a Pyomo concrete model representing the folding problem:

$$\texttt{model.Hgc} = \sum L \cdot T(i, i+1) \quad \text{for } i \text{ in range}(N-2) \tag{8.1.1}$$

$$\texttt{model.obj} = \texttt{minimize}(model.Hgc + model.Hint) \tag{8.1.2}$$

Evaluation of Energy Function

The evaluation functions compute the energy based on the objective function. Auxiliary functions convert measured binary states into energy values.

- `compute_energy(f)`: Evaluates the energy given a binary state representation.
- `average_energy(measurement_counts)`: Computes average energy from measurement counts on real devices.
- `avg(quasi)`: Determines the average energy based on quasi-distributions from simulation results.

Classical to Quantum Hamiltonian Conversion

The script includes functions to convert classical Hamiltonian representation into quantum Pauli operators.

- `binary_to_pauli_z(index)`: Converts a binary variable to a Pauli-Z operator term.
- `classical_to_quantum_hamiltonian(objective_function)`: Parses the objective function and generates a quantum Hamiltonian representation.

$$\text{Hamiltonian} = \sum \text{Pauli Terms} + \sum \text{Coefficients} \tag{8.1.3}$$

QAOA Ansatz and Circuits

The QAOA ansatz is constructed to solve the optimization problem using a parametrized quantum circuit.

- `apply_pauli_terms`: Applies Pauli terms to the quantum circuit.
- `qaoa_ansatz`: Builds the QAOA ansatz by applying the cost and mixer Hamiltonians in layered rounds.

$$\text{QAOA Ansatz} = U(\gamma, \beta)\,|+\rangle^{\otimes n} \tag{8.1.4}$$

QAOA Loop and Optimization

The script uses the Qiskit Runtime and IBM Quantum backends to execute and optimize the QAOA ansatz.

- `optimizer.minimize`: Runs the QAOA optimization loop over multiple iterations, adjusting parameters to minimize the objective function.
- `objective_f`: Objective function for optimizing QAOA parameters using the COBYLA optimizer.

$$\text{Optimal Parameters} = \underset{\gamma,\beta}{\text{argmin}}\langle\psi(\gamma, \beta)|\mathscr{H}|\psi(\gamma, \beta)\rangle \tag{8.1.5}$$

The script can evaluate the Hamiltonian using both real quantum devices and simulated environments, optimizing the QAOA parameters to achieve the minimum energy configuration.

8.2 IBM Devices Setup

The implementation employs quantum computing resources provided by IBMQ. Specifically, three backends are utilized to ensure efficient computation and result verification.

Backend Name	EPLG	Median Redout Error	Median ECR Error	Median SX Error	T1 (µs)	T2 (µs)
ibm_kyoto	3.9%	1.660e-2	9.892e-3	3.215e-4	209.51	104.58
ibm_osaka	3.3%	2.100e-2	9.31e-3	2.85e-4	279.17	163.45
ibm_brisbane	2.2%	1.340e-2	7.42e-3	2.51e-4	231.12	145.97

Table 8.2.1: IBM devices setup for executing the quantum protein folding simulation.

EPLG (Error Per Layered Gate): A measure of the overall error rate for a quantum computation run on the device. It represents the average error incurred per layered gate operation. Lower EPLG values indicate higher fidelity computations.

Median Readout Error: This refers to the median error rate during the measurement or ”readout” of qubit states. Qubits, after computation, are measured to determine their final state (0 or 1), and this metric reflects the typical error in that measurement process. A lower median readout error means more accurate qubit measurements.

Median ECR Error: It is the average gate fidelity, which is the average over all possible input states of the fidelity between the state produced by the actual operation and the state produced by the ideal operation.

Median SX Error: The "SX" refers to a square-root of X gate. It represents the median error rate when implementing this particular type of gate on the device. duration of a measurement in nanoseconds (ns). One of the primary constraints on existing quantum technology is that these periods are frequently much longer than the T2.

T1 (µs): This is the "relaxation time" or "amplitude damping time" of a qubit, measured in microseconds (µs). It quantifies how long a qubit can maintain its excited state (representing a '1') before decaying back to its ground state (representing a '0') due to energy loss to the environment. Longer T1 times are desirable for preserving qubit coherence and performing longer computations.

T2 (µs): Known as the "coherence time" or "dephasing time," T2 (also in µs) measures how long a qubit can maintain a specific superposition state (a combination of 0 and 1) before losing its phase information due to interactions with the environment. Like T1, a longer T2 time is crucial for maintaining the delicate quantum states needed for computations.

8.3 A Look Into the Quantum Circuit

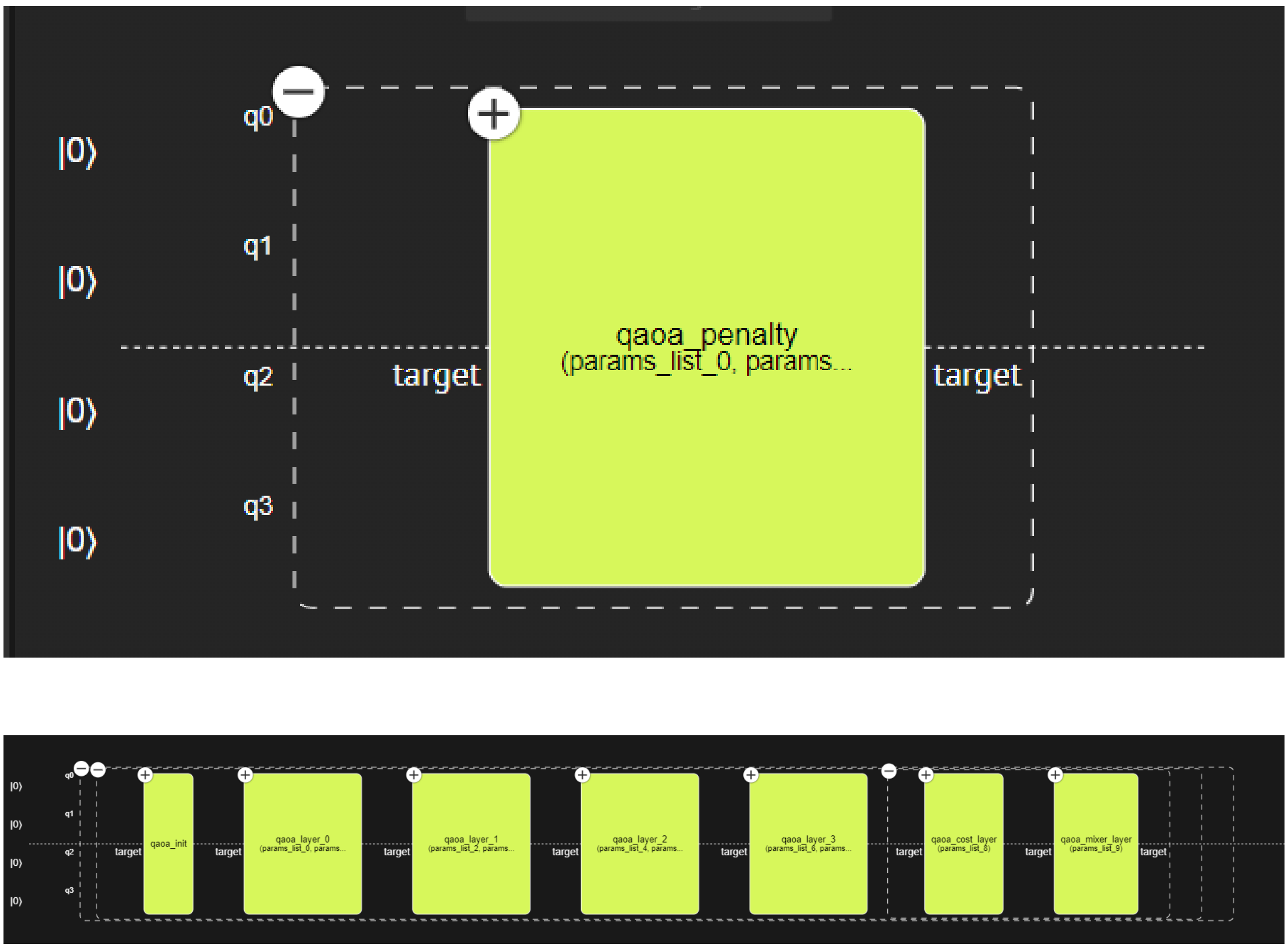


Figure 8.1: The quantum part of the hybrid machine learning model

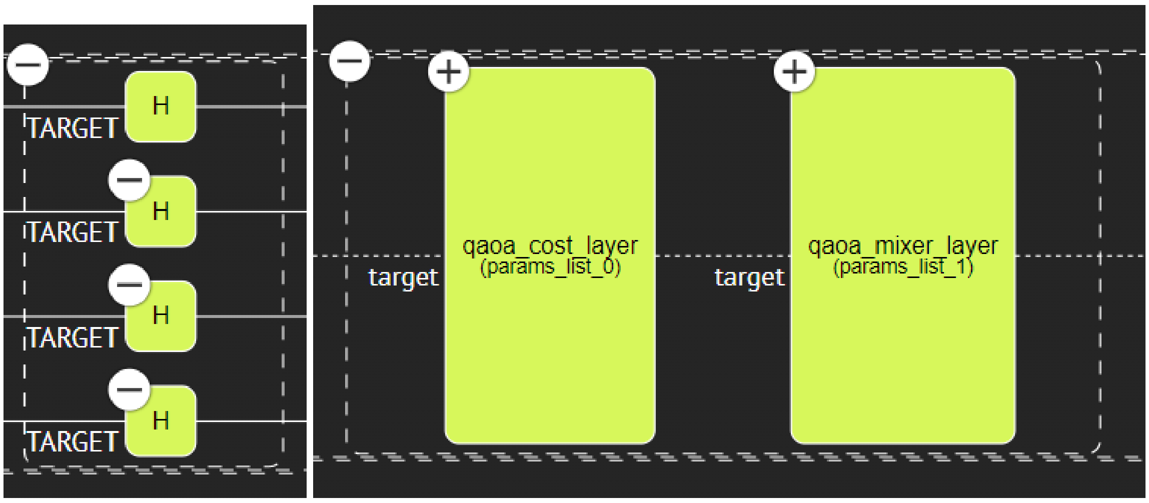


Figure 8.2: The initialization (left) and each of the repetitive layers of the qaoa-ansatz (right)

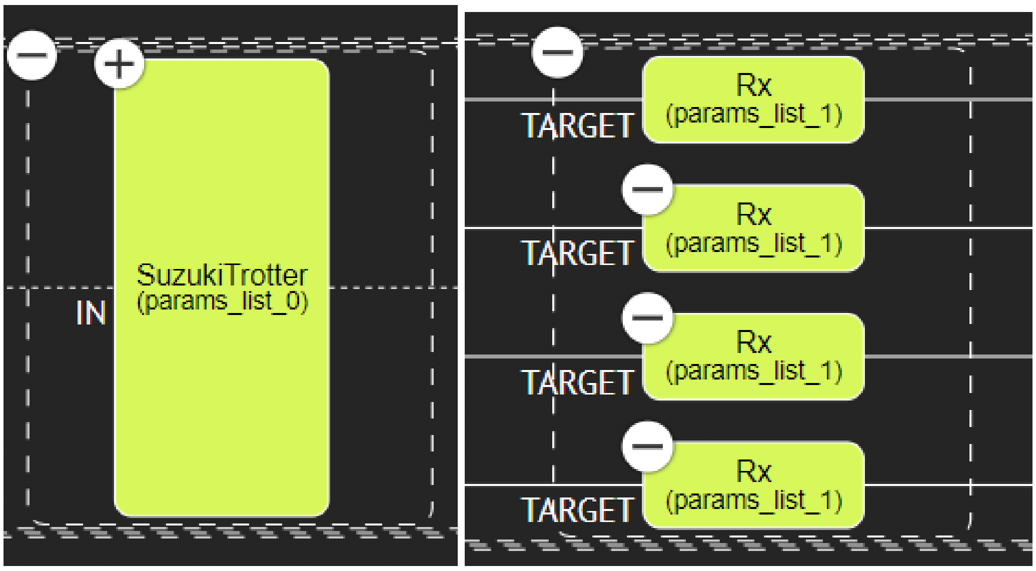


Figure 8.3: The cost-layer (left) using the Trotter Suzuki Formula, and each of the mixed-layer (right)

8.4 Peptide YGGFM

We implement now the previous mentioned hybrid quantum-classical machine learning model, for the YGGFM peptide. We remind that its NMR solution is given in Fig. 8.4. The table that our code uses to calculate the Hamiltonian according to the MJ model for the interactions between amino acids, looks like the Fig. 8.5 for this peptide. However, because the peptide YGGFM has small number of amino acids, this table is not used here as we stated at 8.1, but we provide it the sake of completeness. Implemetation of such algorithm in larger peptides requires access to bigger quantum computers.

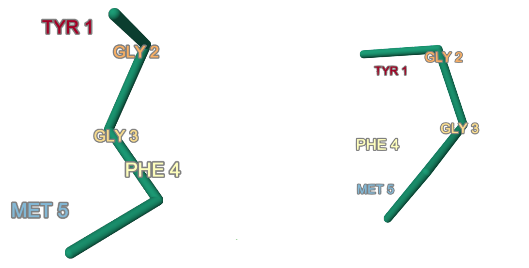


Figure 8.4: NMR solution structure of peptide YGGFM

	Y	G	F	M
Y	-4.17	0.20	0.05	-0.10
G	0.20	-2.24	0.62	0.46
F	0.05	0.62	-7.26	-0.20
M	-0.10	0.46	-0.20	-5.46

Figure 8.5: Interaction energies between amino acids according to the MJ model for YGGFM peptide.

- **Simulation** Using QAOA Resource-Efficient Algorithm we got the structures:

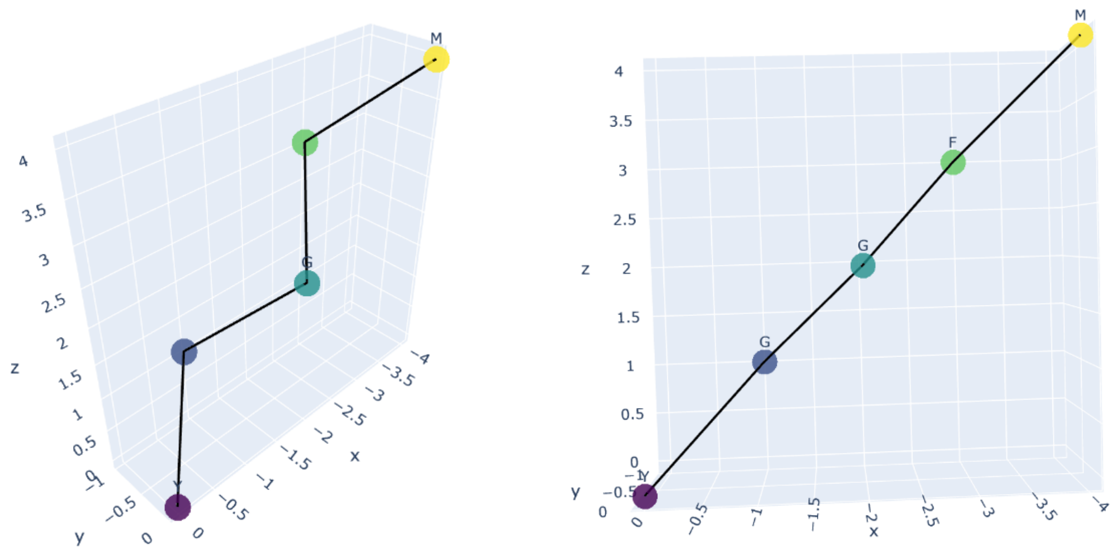


Figure 8.6: 3D structure of peptide YGGFM using simulator

- **IBM's Real Devices**

`ibm_kyoto`

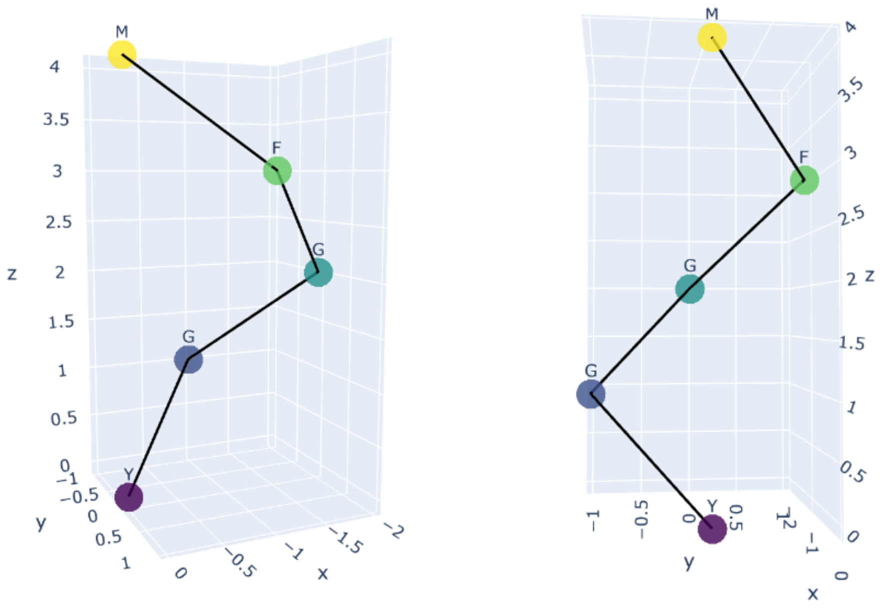


Figure 8.7: 3D structure of peptide YGGFM using `ibm_kyoto`

`ibm_osaka`

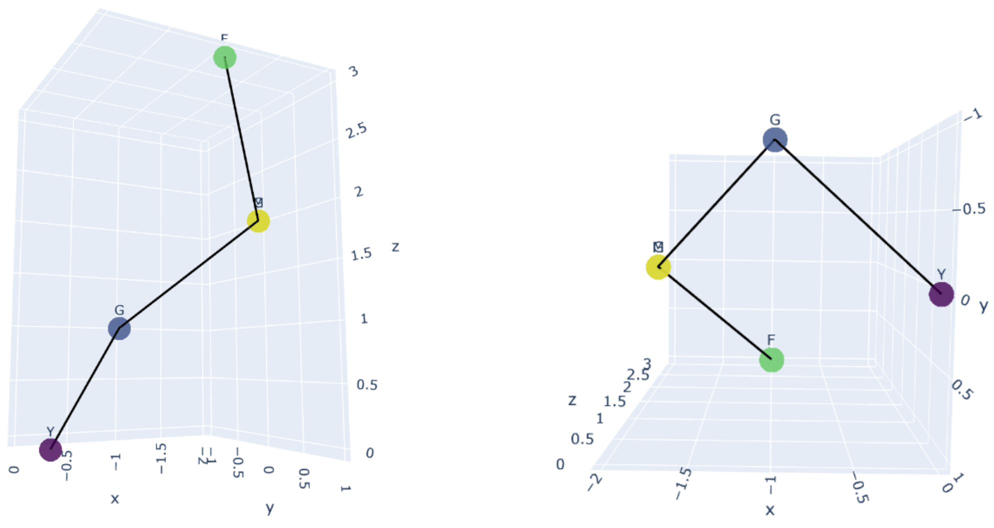


Figure 8.8: 3D structure of peptide YGGFM using `ibm_osaka`

ibm_brisbane

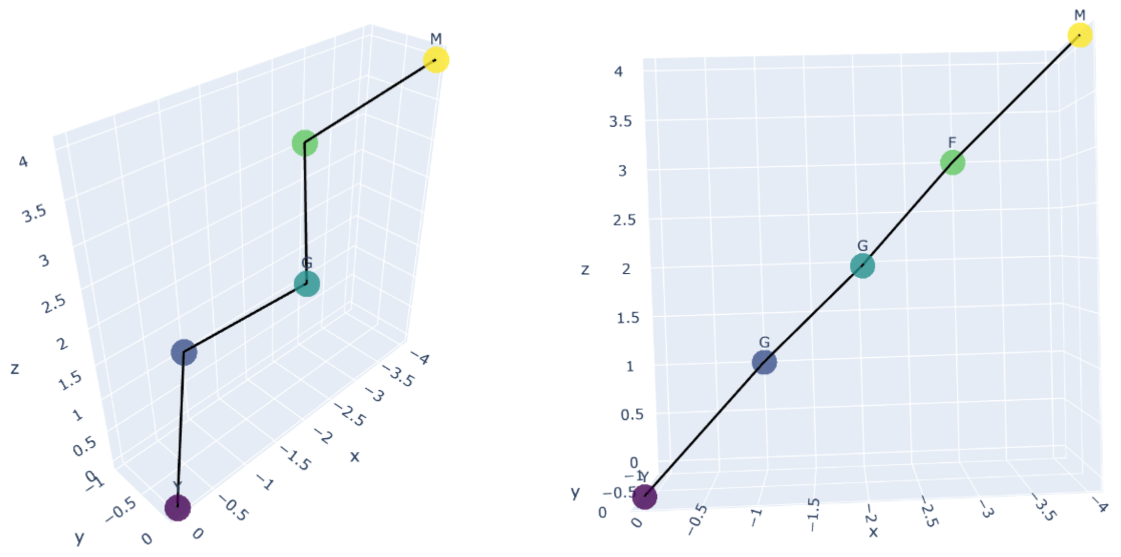
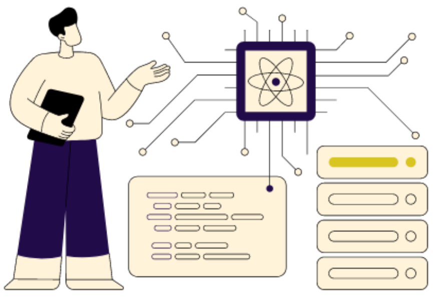


Figure 8.9: 3D structure of peptide YGGFM using ibm_brisbane

It's clear that just the structure of ibm_brisbane perfectly replicate the simulator's result. This highlights the impact of noise and errors present in real quantum devices.



Chapter 9

Conclusion

In order to tackle the complex problem of protein folding, my study investigated the fascinating field where quantum and classical computing collide. This thesis shows that quantum computers can navigate the large conformational space of proteins by mapping the problem onto a simpler lattice model and using variational quantum algorithms, such as QAOA and VQE.

Although there are still difficulties due to the noise in modern quantum devices and the lattice model's intrinsic constraints, this research has provided insightful information. The effectiveness of QAOA in examining the energy landscape and the findings of differences between the outcomes of simulated and real-device experiments, demonstrate the potential and difficulties of this emerging discipline.

Building upon these findings, this work proposes a hybrid quantum-classical machine learning model as a pathway for future research: but this time using both a hybrid model and an additional classical model. This approach leverages the strengths of both computational paradigms: QAOA's speed in traversing vast search spaces to identify promising low-energy regions, followed by a classical ML model's (such as LLMs) ability to refine these conformations to high accuracy.

By training the classical models on the extensive AlphaFold DB and feeding the QAOA output on the sign of the concavity as constraint, this hybrid model has the potential to achieve not only faster convergence but also higher accuracy compared to existing methods. Rigorous evaluation using metrics such as amino acid distances, backbone angles, and bending angles, will be essential to benchmark its performance against established techniques.

The implications of this research extend beyond the realm of structural biology. The successful development of a robust and efficient hybrid model for protein folding could revolutionize drug discovery, materials science, and any field where understanding and predicting 3D structures is paramount. This work represents a significant step towards unlocking the full potential of quantum computation for tackling complex scientific challenges, paving the way for a future where the boundaries between classical and quantum blur, and scientific discovery accelerates at an unprecedented pace.

Bibliography

[1] Pranav Chandarana, Narendra N. Hegade, Iraitz Montalban, Enrique Solano, Xi Chen (2022). arXiv preprint arXiv:2212.13511. https://arxiv.org/abs/2212.13511

[2] Cerezo M., Arrasmith A., Babbush R., Benjamin S. C., Endo S., Fujii K., McClean J. R., Mitarai K., Yuan X., Cincio L., & Coles P. J. (2021). Variational quantum algorithms. *Nature Reviews Physics*, *3*(9), 625–644. https://doi.org/10.1038/s42254-021-00348-9

[3] Benedetti M., Lloyd E., Sack S., & Fiorentini M. (2019). Parameterized quantum circuits as machine learning ls. *Quantum Science and Technology*, *4*(4), 043001. https://doi.org/10.1088/2058-9565/ab4eb5

[4] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, M. D. Lukin (2018). arXiv preprint arXiv:1812.01041. http://arxiv.org/abs/1812.01041

[5] RCSB Protein Data Bank. (n.d.). RCSB PDB: https://www.rcsb.org/.

[6] Li H., Tang C., & Wingreen N. S. (1997). Nature of driving force for protein folding: A result from analyzing the statistical potential., Physical Review Letters, 79(4), 765–768. https://doi.org/10.1103/physrevlett.79.765

[7] Pellow-Jarman A., McFarthing S., Sinayskiy I., Pillay A., & Petruccione F. (2023). QAOA performance in noisy devices: The effect of classical optimizers and ansatz depth. In arXiv [quant-ph]. http://arxiv.org/abs/2307.10149

[8] Jaie Christina Woodard, Monte Carlo simulation approaches to protein stability and aggregation prediction. Harvard.edu.

[9] Sinha S., Tam B., & Wang, S. M. (2022). Applications of molecular Dynamics simulation in protein study. Membranes, 12(9), 844. https://doi.org/10.3390/membranes12090844

[10] Qu X., Swanson R., Day R., & Tsai J. (2009). A guide to template based structure prediction. Current Protein & Peptide Science, 10(3), 270–285. https://doi.org/10.2174/138920309788452182

[11] Robert A., Barkoutsos P. K., Woerner S., & Tavernelli I. (2021). Resource-efficient quantum algorithm for protein folding. Npj Quantum Information, 7(1), 1–5. https://doi.org/10.1038/s41534-021-00368-4

[12] Weiss R., Karimijafarbigloo S., Roggenbuck D., & Rödiger S. (2022). Applications of neural networks in biomedical data analysis. Biomedicines, 10(7), 1469. https://doi.org/10.3390/biomedicines10071469

[13] Swenson N., Krishnapriyan A. S., Buluc A., Morozov D., & Yelick, K. (2020). PersGNN: Applying topological data analysis and geometric deep learning to structure-based protein function prediction. In arXiv [q-bio.BM]. http://arxiv.org/abs/2010.16027

[14] Pearce R., & Zhang Y. (2021). Deep learning techniques have significantly impacted protein structure prediction and protein design. Current Opinion in Structural Biology, 68, 194–207. https://doi.org/10.1016/j.sbi.2021.01.007

[15] Peruzzo A., McClean J., Shadbolt P., Yung M.-H., Zhou X.-Q., Love P. J., Aspuru-Guzik A., & O'Brien J. L. (2013). A variational eigenvalue solver on a quantum processor. http://arxiv.org/abs/1304.3061. In arXiv [quant-ph].

[16] Bharti K., Cervera-Lierta A., Kyaw T. H., et al. (2022). Noisy intermediate-scale quantum algorithms. Reviews of Modern Physics 94(1). https://doi.org/10.1103/revmodphys.94.015004

[17] Holm M. (2020). Using Deep Reinforcement Learning for Active Flow Control. Master's thesis, Faculty of Mathematics and Natural Sciences, University of Oslo. https://www.duo.uio.no/bitstream/handle/10852/79212/1/main.pdf

[18] W. Lavrijsen et al., "Evaluating Classical Optimizers on Noisy Intermediate-Scale Quantum Devices," 2021.

[19] Rios et al., "Derivative-Free Optimization for Noisy Objective Functions," 2016.

[20] Sabrina Herbst, Vincenzo De Maio, and Ivona Brandic. On optimizing hyperparameters for quantum neural networks. *ArXiv preprint arXiv:2403.18579*, 2024.

[21] Band Y. B., & Avishai Y. (2013). The formalism of quantum mechanics. In *Elsevier eBooks* (pp. 61–104). https://doi.org/10.1016/b978-0-444-53786-7.00002-2

[22] Hermann, R. (1983). The Theory of Spinors. by Elie Cartan. *The American Mathematical Monthly*, 90(10), 719–720. https://doi.org/10.1080/00029890.1983.11971324

[23] Blekos K., Brand D., Ceschini A., Chou C-H., Li R.-H., Pandya K., and Summer A. (2024). A review on Quantum Approximate Optimization Am and its variants. *Physics Reports*, *1068*, 1–66. https://doi.org/10.1016/j.physrep.2024.03.002

[24] Farhi E., Goldstone J., and Gutmann S. (2014, November 14). A Quantum Approximate Optimization algorithm. *arXiv.org*. https://arxiv.org/abs/1411.4028

[25] Somma, R. D. (2016). A Trotter-Suzuki approximation for Lie groups with applications to Hamiltonian simulation. *Journal of Mathematical Physics*, *57*(6). https://doi.org/10.1063/1.4952761

[26] Expressibility and entanglement capability of parameterized quantum circuits. (n.d.). https://obliviateandsurrender.github.io/blogs/expr.html

[27] Sim, S., et al. (2019). Expressibility and entangling capability of parameterized quantum circuits for hybrid Quantum‐Classical algorithms. *Advanced Quantum Technologies*, *2*(12). https://doi.org/10.1002/qute.201900070

[28] Wiedmann, M., Hölle, M., Periyasamy, M., Meyer, N., Ufrecht, C., Scherer, D. D., Plinge, A., and Mutschler, C. (2023). An Empirical Comparison of Optimizers for Quantum Machine Learning with SPSA-Based Gradients. https://doi.org/10.1109/qce57702.2023.00058

[29] Lockwood, O. (2022, February 3). An Empirical review of optimization techniques for quantum variational circuits. *arXiv.org*. https://arxiv.org/abs/2202.01389

[30] Using deep reinforcement learning for active flow control. (2020). https://www.duo.uio.no/handle/10852/79212?show=full

[31] Franken, L., Georgiev, B., Mücke, S., and Bauckhage, C. (2020). Gradient-free quantum optimization on NISQ devices. *ResearchGate*. https://www.researchgate.net/publication/347965271_Gradient-free_quantum_optimization_on_NISQ_devices

[32] Lavrijsen, W., Tudor, A., Müller, J., and De Jong, W. A. (n.d.). Classical Optimizers for Noisy Intermediate-Scale Quantum Devices. *ResearchGate*. https://www.researchgate.net/publication/340500054_Classical_Optimizers_for_Noisy_Intermediate-Scale_Quantum_Devices

[33] Protein folding dynamics and stability. (2023). https://doi.org/10.1007/978-981-99-2079-2

[34] Miyazawa, S., and Jernigan, R. L. (1996). Residue – Residue Potentials with a Favorable Contact Pair Term and an Unfavorable High Packing Density Term, for Simulation and Threading. *Journal of Molecular Biology/Journal of Molecular Biology*, *256*(3), 623–644. https://doi.org/10.1006/jmbi.1996.0114

[35] Singh, H., Majumder, S., and Mishra, S. (2023). Benchmarking of different optimizers in the variational quantum algorithms for applications in quantum chemistry. *Journal of Chemical Physics*, *159*(4). https://doi.org/10.1063/5.0161057

[36] Fang, Y. (2012, February 7). Protein folding: the Gibbs free energy. *arXiv.org*. https://arxiv.org/abs/1202.1358

[37] Sun, P. D., Foster, C. E., and Boyington, J. C. (2004). Overview of protein structural and functional folds. *Current Protocols in Protein Science*, *35*(1). https://doi.org/10.1002/0471140864.ps1701s35

[38] Pande, V. S., Grosberg, A. Y., Tanaka, T., and Rokhsar, D. S. (1998). Pathways for protein folding: is a n needed? *Current Opinion in Structural Biology*, *8*(1), 68–79. https://doi.org/10.1016/s0959-440x(98)80012-2

[39] Lazaridiridis, T., and Karplus, M. (1997). "New View" of Protein Folding Reconciled with the Old Through Multiple Unfolding Simulations. *Science*, *278*(5345), 1928–1931. https://doi.org/10.1126/science.278.5345.1928

[40] Classiq Technologies' documentation. https://docs.classiq.io/

[41] IBM Quantum documentation. https://docs.quantum.ibm.com/

Appendices

Appendix A

Standard Amino Acids' Notations

Table A.0.1: Amino Acids

Amino Acid	**3-Letter Code**	**1-Letter Code**
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartate	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamate	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V