Pedro da Silveira Carvalho Ripper

Study on a Variational Quantum Algorithm for molecule ground state estimation

RELATÓRIO DE PROJETO FINAL II

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DEPARTAMENTO DE INFORMÁTICA

Programa de graduação em Engenharia de Computação

Rio de Janeiro
06 de 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Pedro da Silveira Carvalho Ripper

Study on a Variational Quantum Algorithm for molecule ground state estimation

Relatório de Projeto Final II

Orientador : Prof. Guilherme Penello Temporão
Co-orientador: Dr. Gustavo Castro do Amaral

Rio de Janeiro
06 de 2023

Acknowledgments

First, I would like to thank my advisors, Guilherme Penello Temporão and Gustavo Castro do Amaral, for taking in a curious Computer Engineering student with no prior knowledge of Quantum Science. Since the beginning of my Undergraduate Research project in 2020, they have not only been excellent advisors, teaching me so much, but also great friends.

I am forever grateful to my family, specially my parents, Gláucia and Carlos, my sister Júlia and grandparents, Angela and Roberto, for the unconditional love and encouragement for everything that I wanted to pursue in these past few years, which were what sustained me during my journey through College.

I would also like to thank the friends that I have made during my studies at UFRJ, Álvaro d'Armada, Julia Togashi, Felipe Schreiber and Pedro Costa, and at PUC-Rio, Constance Duarte, Gabriel Manhães, Lucas Guilhon, Miguel Vuori, Pedro Costa (again) and Rachel Szenberg. Additionally, I am thankful for my friends from my exchange program at UCPH, Beatrice Kaukiainen, Sarah Clusiau and Susan Lin.

Besides I wanted to extend my gratitude to my colleagues at PSR, specially Carolina Monteiro, Guilherme Bodin, Iago Leal, Joaquim Dias Garcia, Pedro Maciel Xavier, Rafael Benchimol, Raphael Sampaio and Tiago Andrade.

Furthermore, I would like to acknowledge the help from the Qiskit community during the experimental part of my work.

Finally, I would like to thank the Pontifical Catholic University of Rio de Janeiro.

Summary

'Tá safo. Não pega nada.'

Vovô Roberto.

1
Introduction

Over the last few years, Quantum Computing(QC) has been gaining its place under the spotlight of emerging technologies, attracting investments from both public and private sectors. Although these financial ventures have been concentrated in more technologically advanced regions, such as United States, Canada, China and European Countries, as seen in Figure 1.1, the Brazilian government has recently started opening calls for funding for quantum technologies[1].

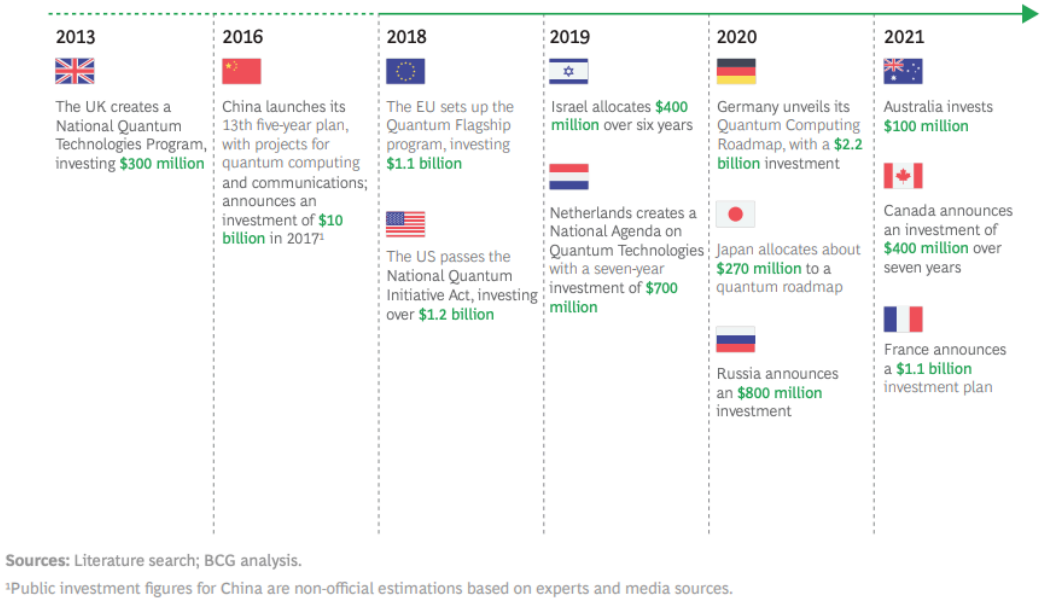

Figure 1.1: Investment plans in quantum technologies over different countries Source: BCG Report - *"Can Europe Catch Up with the US (and China) in Quantum Computing?"*, August 2022

This interest comes from the expected benefits from using quantum computers [1], which will bring what is known as *Quantum Advantage* for some applications in comparison with non-quantum (classical) computers. However, as of today, there is no proof of such phenomenon in the real world, being only shown in some very specific and tailored scenarios, exclusive to laboratory experiments [2; 3].

Quantum computers are yet to be useful outside the research bubble due to the fact that they are susceptible to unreliable operations and decoherence, which limit what a quantum computer can do nowadays. That being said, QC is in a embryonic stage known as the Noisy Intermediate Scale Quantum (NISQ)

[1]https://embrapii.org.br/futuro-da-industria-iniciativa-oferece-r-60-milhoes-para-desenvolver-computacao-quantica/

Era [4], where its machines lack the necessary technology to perform fault-tolerant operations.

There are already some quantum algorithms which harness the unique capabilities of a quantum computer[5; 6], but, unfortunately, they are still years away from being reliably executable on real hardware. Under this perspective, researchers have been developing algorithms that take into account the shortcomings of current machines, trying to perform useful computations based on what quantum systems can do for now. A great example for that is the class of Variational Quantum Algorithms (VQA) [7], composed of hybrid classical-quantum algorithms, which spares a quantum computer from executing all operations.

The focus of this project is to study a VQA known as the Variational Quantum Eigensolver(VQE) [8], designed to estimate the quantum state associated with the lowest energy of a Hamiltonian. The work is presented as follows. In Chapter 2, it is introduced some basic concepts of QC, followed by an overview of its current stage in Chapter 3. Then, in Chapter 4, the class of VQAs has its workflow analysed in parts. Subsequently, the Chapter 5 introduces the VQE algorithm in more depth. In Chapter 6, the environment where the experiments were performed is explained. Following, Chapter 7 presents all experiments using VQE, with a discussion of the results. Finally, Chapter 8 wraps-up this work, reviewing what was presented and what is expected in the near-future for QC, VQAs and, more specifically, the VQE.

2
Quantum Computing concepts

2.1
Quantum states and the Hilbert Space

There are different interpretations for the Postulates of Quantum Mechanics, that don't differ in the overall meaning, but most in the number of postulates. The postulates that will be mentioned in this work were based on Nielsen and Chuang's *Quantum Computation and Quantum Information* [9].

According to the first postulate, every state for a given quantum system can be represented by a vector $\mathbf{x}$ in a Hilbert Space. Conversely, every vector $\mathbf{x}$ in a Hilbert Space relates to a possible state of a quantum system [9].

Moreover, following the third postulate [9], to extract information from a quantum system in a Hilbert space $\mathcal{H}$, one needs to perform measurements, where their expected values use the inner product between two vectors $\mathbf{x}, \mathbf{y} \in \mathcal{H}$. According to Riesz representation theorem, this inner product can be represented by the product between $\mathbf{y}^* \in \mathcal{H}^*$ and $\mathbf{x}$, with $\mathbf{y}^*$ being a row vector and $\mathbf{x}$ a column vector.

2.2
Dirac or Bra-ket notation

The Dirac or Bra-ket notation [10] is widely used by physicists in quantum mechanics to represent vectors and operations. A column vector $\psi(x)$ is represented by a "ket" $|x\rangle$ and its transpose conjugate $\psi(x)^\dagger$ by a row vector "bra" $\langle x|$.

The bra-ket notation provides a simple way for representing operations with vectors. Table 2.1 lists some of them.

Operation	**Representation**
Inner Product	$\langle y\|x\rangle$
Outer Product	$\|x\rangle \langle y\|$
Tensor Product	$\|x\rangle \otimes \|y\rangle$ or $\|x\rangle \|y\rangle$ or $\|xy\rangle$
Matrix Multiplication	$A \|x\rangle$

Table 2.1: Bra-ket notation for vector operations

2.3
Qubits

Although quantum computers were first developed not too long ago, the idea of a computer that could simulate physics has been a topic of discussion

for some time already by renowned scientists, such as Richard Feynman [11]. Moreover, in the year 2000, David DiVincenzo outlined the requirements for the implementation of a quantum computer [12], where he highlighted the vast variety of realizations of qubits, the basic unit of Quantum Information.

A qubit is a two-level quantum system, like the vertical and horizontal polarization for a photon. Its state can be represented in many ways, using different orthonormal vector basis. The most used one is the computational basis in $\mathbb{C}^2$, where its basis vectors are presented in Equation 2-1. Using this notation, a qubit $|\psi\rangle$ can be represented as follows in Equation 2-2.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2-1}$$

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2-2}$$

Where $\alpha = \cos\frac{\theta}{2}$, $\beta = e^{i\phi}\sin\frac{\theta}{2}$ and $|\alpha|^2 + |\beta|^2 = 1$.

From the values of α and β from Equation 2-2, it follows that a qubit is in a two-dimensional complex vector space ($\mathbb{C}^2$) known as the Hilbert Space. Moreover, a qubit can be represented as a unitary vector ($|||\vec{\psi}\rangle|| = 1$) in what is known as Bloch Sphere, as presented in Figure 2.1, where we can see the variables θ and ϕ from Equation 2-2. A qubit's bloch vector has three coordinates, which are defined as follows in Equation 2-3.

$$\begin{aligned} \vec{\psi} &= (x, y, z), \\ \text{where } x &= cos\phi\ sin\theta, \\ y &= sin\phi\ sin\theta, \\ z &= \cos\theta \end{aligned} \tag{2-3}$$

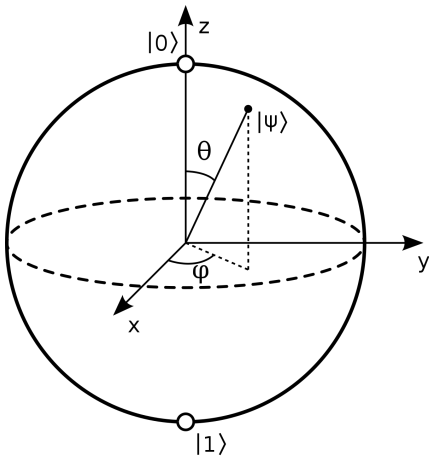

Figure 2.1: Bloch Sphere
Source:Wikipédia

Another important form of representation of a quantum state is the Density Matrix, also known as Density Operator. One can find the Density Operator ρ for a given state with Equation 2-4, where the variables x, y and z are the coordinates for the bloch vector, from Equation 2-3. The density matrix correspondent to a qubit $|\psi\rangle$ can also be calculated according to Equation 2-5.

$$\rho = \frac{1}{2}\begin{pmatrix} 1+z & x-iy \\ x+iy & 1-z \end{pmatrix} \tag{2-4}$$

$$\rho = |\psi\rangle\langle\psi| \tag{2-5}$$

Density Matrices have some properties:

1. $Tr(\rho) = 1$
2. $\langle\phi|\rho|\phi\rangle \geq 0, \forall\, |\phi\rangle \in \mathcal{H}$
3. $Tr(\rho^2) = 1 \iff \rho$ is a pure state
4. $Tr(\rho^2) < 1 \iff \rho$ is a mixed state
5. If an operator U is applied to a density matrix ρ, the resulting state is $\rho' = U\rho U^\dagger$
6. $\langle O\rangle_\rho = Tr(\rho O)$

where $Tr(A)$ is the trace of a matrix A.

Mixed states cannot be represented as in Equation 2-2. Moreover, the bloch vector associated to a mixed state is not unitary. Therefore is is possible to check if a state is pure by the norm of its bloch vector or by $Tr(\rho^2)$.

Furthermore, it is worth mentioning that the Bloch vector related to a pure state lies on the surface of the sphere, while the one that represents a

mixed state, is in the interior of the sphere. This concept will be later discussed, with a visual representation in Figure 2.7.

2.4 Composite Systems

In order to perform useful computations, it is necessary to use more than one qubit, which would make up a composite system. According to fourth postulate of Quantum Mechanics [9], a joint state of $1 \dots n$ qubits is represented by their tensor product $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$ [9]. Thus, one could represent a general joint state as follows in Equation 2-6.

$$\begin{aligned} |\psi_1\psi_2\rangle &= (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) \\ &= \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_2\beta_2 |11\rangle \end{aligned} \tag{2-6}$$

Having a two-qubit joint state $|\psi\rangle$, one would be able to find two one-qubit states $|\psi_A\rangle$ and $|\psi_B\rangle$ such that $|\psi_A\psi_B\rangle = |\psi\rangle$. This follows for any joint state with an arbitrary number of qubits, but not for every composite state, as will be discussed. For instance, consider the state defined in Equation 2-7

$$|\psi\rangle = \frac{1}{2} |0\rangle |0\rangle + \frac{1}{2} |0\rangle |1\rangle + \frac{1}{2} |1\rangle |0\rangle + \frac{1}{2} |1\rangle |1\rangle \tag{2-7}$$

Checking the coefficients of this joint state, we know that:

$$\alpha_A * \alpha_B = \frac{1}{2} \ , \ \alpha_A * \beta_B = \frac{1}{2} \ , \ \beta_A * \alpha_B = \frac{1}{2} \ , \ \beta_A * \beta_B = \frac{1}{2}$$

Thus, one could set each of the coefficients to $\frac{1}{\sqrt{2}}$ to find the following $|\psi_A\rangle$ and $|\psi_B\rangle$ states:

$$|\psi_A\rangle = \alpha_A |0\rangle + \beta_A |1\rangle , \quad |\psi_B\rangle = \alpha_B |0\rangle + \beta_B |1\rangle$$

However, not every composite state can be dismantled into the qubits that comprise it. For instance, consider the quantum state defined in Equation 2-8.

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2-8}$$

It is impossible to find four coefficients α_A, β_A, α_B and β_B, to later define the two qubits that form this state. As proof, lets consider the general two-qubit joint state formula in Equation 2-6. In order to have a joint state as a sum of $\frac{|00\rangle}{\sqrt{2}}$ with $\frac{|11\rangle}{\sqrt{2}}$, we have the following constraints:

$$\alpha_1 * \beta_2 = 0, \quad \beta_1 * \alpha_2 = 0$$

Moreover, if we want to have the tensor products $|00\rangle$ and $|11\rangle$, α_1 and α_2 need to be $\sqrt{\frac{1}{\sqrt{2}}}$, as well as β_1 and β_2. Yet, this is not feasible, as it would violate the restrictions that were just defined.

The state presented in Equation 2-8 is known as an entangled state where the two quantum systems are sharing energy.

2.5 The Hamiltonian

The central equation of Quantum Mechanics is the Schrödinger equation, which describes the evolution over time of a quantum system, as presented in Equation 2-9. The operator H is known as the Hamiltonian of the system, correspondent to the total energy of the system, and $\hbar$ is the Planck's constant, whose value needs to be experimentally determined.

$$i\hbar\frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle \tag{2-9}$$

On its time-independent form, it can be represented in a simplified fashion, as follows in Equation 2-10.

$$H\,|\psi\rangle = E\,|\psi\rangle \tag{2-10}$$

2.6 Operators

The second postulate of Quantum Mechanics states that the evolution of a closed quantum system is described by a unitary transformation, represented by a matrix operator [9]. Equation 2-11 portrays the relation between states $|\psi'\rangle$ and $|\psi\rangle$ by a unitary operator U.

$$|\psi'\rangle = U\,|\psi\rangle \tag{2-11}$$

Moreover, another representation of the evolution of a closed quantum system is by the Schrödinger equation, previously defined in Equation 2-9.

Considering that $|\psi\rangle$ and $|\psi'\rangle$ are the description of a quantum system at times t_1 and t_2, respectively, one can draw a connection between these two depictions. Reaching the solution of Schrödinger equation, as displayed in Equation 2-12, one can perceive that the unitary operator U is as follows in Equation 2-13.

$$|\psi'\rangle = \exp\left[\frac{-iH(t_2-t_1)}{\hbar}\right]|\psi\rangle = U\,|\psi\rangle \tag{2-12}$$

$$U(t_1,t_2) = \exp\left[\frac{-iH(t_2-t_1)}{\hbar}\right] \tag{2-13}$$

It is possible to draw a parallel between logic gates that perform operations on bits and operators that act on qubits. That being said, they are often called quantum gates. A procedure comprised of quantum gates being applied on qubits would be a Quantum Process.

Furthermore, taking into account that the Hamiltonian is an Hermitian operator [9], it has a spectral decomposition, where E is the eigenvalue correspondent to the normalized $|E\rangle$ eigenvector, as outlined in Equation 2-14. Thus, the Hamiltonian can be characterized by a set of quantum gates acting on a quantum system.

$$H = \sum E \, |E\rangle \, \langle E| \tag{2-14}$$

As a qubit is represented by a 2-dimensinal vector, a quantum gate corresponds to a 2x2 matrix. Applying an operator on a qubit would mean a multiplication between the matrix representing the operator and the qubit vector, as illustrated in Equation 2-15, where the qubit goes from the $|0\rangle$ to $|1\rangle$ state.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2-15}$$

The matrix chosen for the example is known as the Pauli X operator (or the X gate). When applied, it rotates a qubit on a 180^o angle over the X-axis in the Bloch Sphere, as displayed in Figure 2.2.

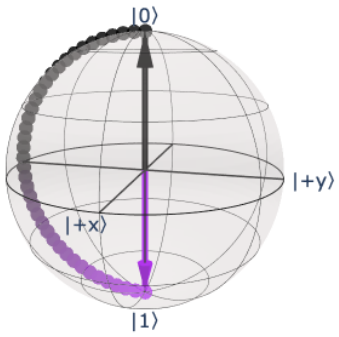

Figure 2.2: A qubit $|0\rangle$ state in the Bloch Sphere after applying an X gate.

When considering a composite system, one may need to perform operations on more than one qubit. As a joint or entangled state of n qubits is in a $\mathbb{C}^{2^n}$ space, a $2^n \times 2^n$ matrix is required. Multi-qubit quantum gates can affect each qubit independently or not.

2.7 The Quantum Circuit model

The most common way to depict a set of operations on a quantum system is the Quantum Circuit model [9]. In this framework, by convention, every qubit is initially set to $|0\rangle$. Additionally, the operations on a qubit are displayed just as logic gates. Having said that, it becomes easier to understand this structure making parallels with logic circuits.

Consider the quantum circuit presented in Figure 2.3. An important concept which will be later explored is the circuit depth, that is equal to the maximum number of operators that were applied to one of the qubits. Each kind of operation was indexed as follows.

1. Single-qubit operators: A Pauli X gate is applied to qubit q_0 and a Pauli Y to qubit q_1
2. Two-qubit operator: A Controlled-NOT (CNOT) gate is applied with qubit q_0 as control and qubit q_1 as target
3. Three-qubit operator: A Toffoli gate is applied with qubit q_0 and qubit q_1 as source and qubit q_2 as target
4. A measurement is performed on qubit q_0 and stored in a classical bit c_0

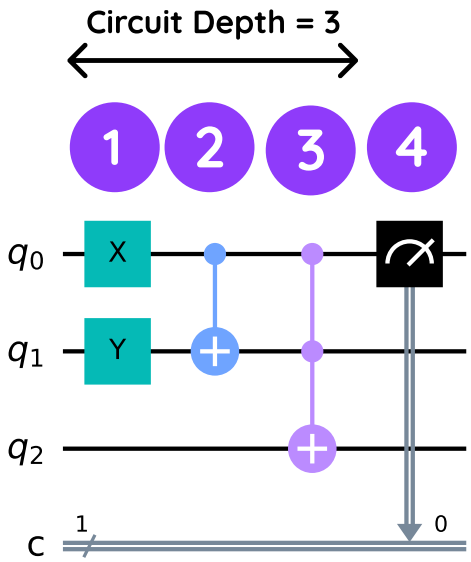

Figure 2.3: Example of a quantum circuit

2.8 Decoherence

One of the DiVincenzo's requirements for quantum computers is that the decoherence time for a quantum system should be much longer than the operation time. Decoherence can be defined as a qubit's process of "leaking" its information to the external world (the environment), thus sharing energy with it. Therefore, the magnitude of decoherence that a qubit is subjected to impacts in the quality of the results from a quantum computer.

To illustrate this phenomenon, consider a two-qubit quantum computer with a Hilbert Space $\mathcal{H} \in \mathbb{C}^4$ that can be described as $\mathcal{H}_1 \otimes \mathcal{H}_2$, where $\mathcal{H}_1$ and $\mathcal{H}_2 \in \mathbb{C}^2$. Now, let $|\psi\rangle = |0\rangle$ be a qubit in $\mathcal{H}_1$. It would be possible to apply any unitary operation and the qubit would preserve its coherence. If we apply an X gate to qubit $|\psi\rangle$ - Figure 2.4 - followed by a measurement, and run it several times in an ideal quantum computer, we would find that 100% of the outcomes are 1, as presented in Figure 2.5

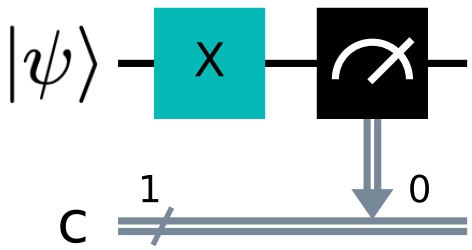

Figure 2.4: Circuit where it is applied an X gate to $|\psi\rangle$

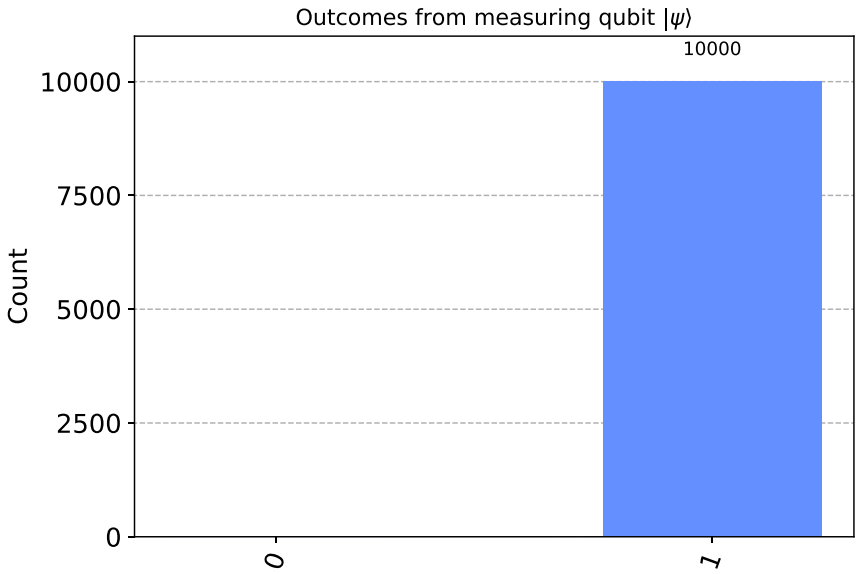

Figure 2.5: Outcomes from measuring qubit $|\psi\rangle$ after applying an X gate

Moreover, if, unintentionally, our qubit $|\psi\rangle$ becomes entangled with another qubit $|\phi\rangle$ from $\mathcal{H}_2$ without our knowledge, it would result in an entangled state $|\psi'\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$. As we are unaware of this event, performing a measurement would not return us the same results as before. This is due to the fact that, as we are perceiving only a subset of a larger Hilbert Space, to which the state $|\psi'\rangle$ belongs, the qubit $|\psi\rangle$ becomes a mixed-state. This unintended entanglement also happens to the environment, where the qubit, or the quantum system, would also lose its information.

There is a relation between the degree of entanglement between qubits, or the environment, and the degree of decoherence when we measure only one of them. Therefore, it is possible to mimic this leakage of quantum information in qubit $|\psi\rangle$ with the circuit in Figure 2.6. As the angle θ of the R_Y gate approaches $\frac{\pi}{2}$ radians, the qubits $|\psi\rangle$ and $|\phi\rangle$ become more entangled. At the same time, resulting state of qubit $|\psi\rangle$ shifts into a more mixed state.

It is worth mentioning that if we had access to qubit $|\phi\rangle$ from Figure 2.6, it would be possible to retrieve the "lost" information. Therefore, decoherence is associated to an unrecoverable partial information of the quantum state, which lies in a bigger Hilbert space $\mathcal{H}_1 \otimes \mathcal{H}_2$.

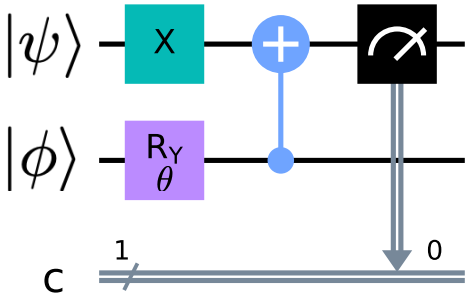

Figure 2.6: Circuit simulating qubit $|\psi\rangle$ suffering decoherence according to a rotation R_Y of θ radians

One way to notice this behavior is by analysing the length of the bloch vector that represents the state $|\psi\rangle$ and the purity of the state given by $Tr(\rho_\psi)$, where ρ_ψ is the Density Matrix that represents $|\psi\rangle$. The values of the length of the bloch vector and the purity of $|\psi\rangle$ given by the value of θ are presented in Figure 2.7.

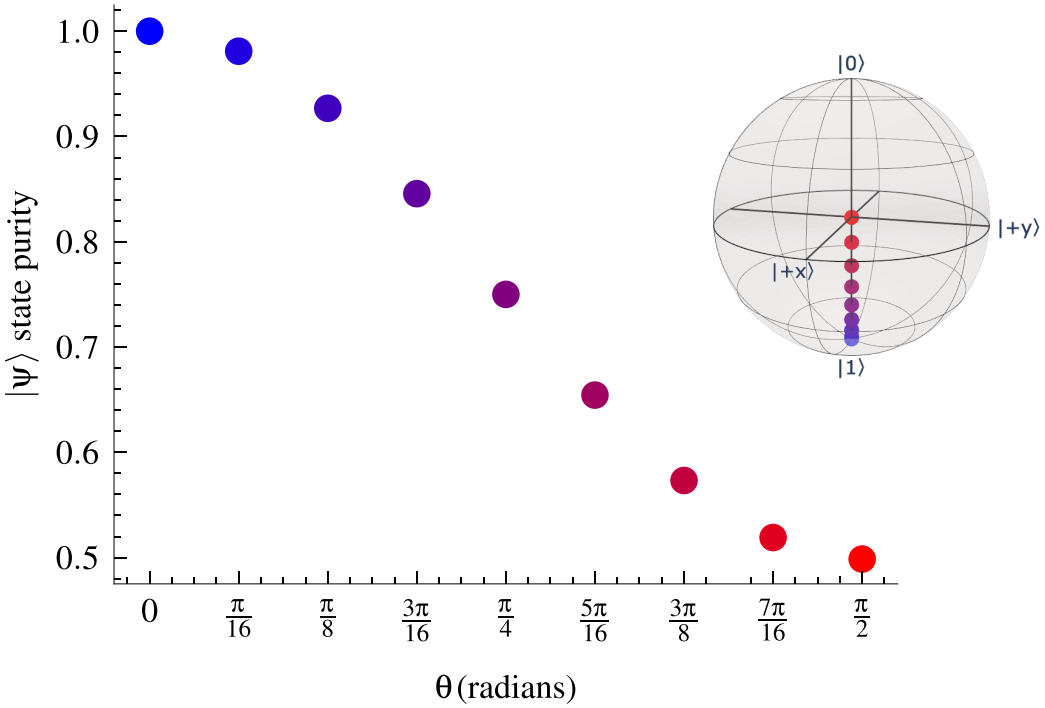

Figure 2.7: $|\psi\rangle$ bloch vector norm and state purity given by θ

Characterizing the decoherence process in current quantum computers can be a challenge as it is hard to distinguish results that were affected by this phenomenon [13].

3
The NISQ Era

In 2016, IBM released the first quantum computer available to anyone in the world through its cloud service, IBM Quantum Experience, for free. Such event helped to democratize the access to quantum hardware and, since then, other tech companies developed their own QC platforms, such as Amazon's AWS Braket and Microsoft's Azure Quantum. Moreover, in the last few years, many quantum technology startups have been founded, backed by large investments from both private and public sectors, such as Xanadu, Rigetti and IQM.

This great interest in QC was, and still is, caused by the prediction that quantum computers will provide better solutions for well-known hard problems. Unfortunately, this expectation has been misinterpreted in some cases, creating a scenario know as *Quantum Hype*. In this chapter, it will be introduced what is actually expected of a quantum computer, from a computational complexity perspective, the current state of quantum computers and what is possible to do with them today and in the near future.

3.1
Noise

As portrayed by Preskill [4], the present stage of QC is known as the Noisy Intermediate Scale Quantum (NISQ) Era, where quantum computers are limited by hardware constraints which allow the introduction of errors during computations. This noise can be originated from different sources.

First, the quantum computer unit of information, a qubit, is susceptible to decoherence, which can come from interactions with the environment or other qubits [14] inside the system. A qubit suffers decoherence over time, so circuits that take a longer time to be performed have lower fidelity results at the end of execution. Time can be perceived in a quantum computer by the amount of quantum gates that comprise it. As each gate has an operation time, stacking different operators increases the duration of a circuit.

Secondly, an operation on a qubit has an error rate, which increases as more quantum gates are added to the circuit. As both issues are caused by the number of operators in a circuit, it is important for NISQ algorithms to have small depths.

The depth of a quantum circuit is a metric that depends on the amount of quantum gates added to a process. Researchers are constantly taking this unit of measurement to benchmark their algorithms, as error-prone quantum computers depend on shallow circuits.

Moreover, a quantum computer usually only performs a small set of one-qubit operations and a CNOT gate. When a circuit is sent to a quantum computer, if an operation is not a basic gate, it is needed to be decomposed into other gates.

In addition, for a two-qubit gate, it is possible that the envisioned qubits for this operation are not connected. The manufacturing of a quantum computer chip draws the possible connections that can be made between qubits. As an example, Figure 3.1 is the connection topology of an actual quantum computer from IBM (`ibm_lagos`). Therefore, just as gate decomposition, some operators might need to be translated into a set of new gates.

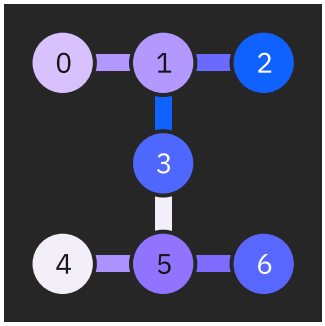

Figure 3.1: `ibm_lagos` qubit connectivity

Due to this connectivity problem and the basic gates restriction, when a circuit is sent to a quantum computer, it needs to go through a pre-processing phase, where it is checked whether any gate violates the hardware's restrictions. If any impossible operation is found, the circuit needs to be translated according to the hardware layout. Furthermore, this translation ends up increasing the circuit depth. That being said, this process of assignment of qubits and gates is known as the qubit routing problem and researchers have been trying to enhance it [15], in order to mitigate errors.

As an example, considering the hardware architecture in Figure 3.1, if we try to apply a Toffoli gate (Figure 3.2), it would have to be translated into the circuit presented in Figure 3.3. Using the circuit depth metric, the first circuit has a depth of 1 and the second of 19.

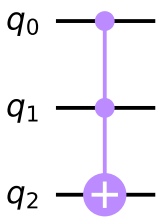

Figure 3.2: Circuit with Toffoli Gate - depth 1

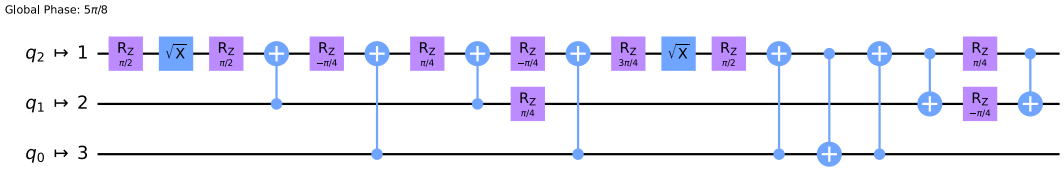

Figure 3.3: Circuit with Toffoli Gate translation according to *ibm_lagos* hardware - depth 19

There are methods to mitigate errors in quantum computers, known as the Quantum Error Correction (QEC) algorithms [16]. However, it is unfortunate that these protocols require a large amount of qubits, another constraint in current hardware.

3.2 Complexity Classes

Complexity Theory is one of the main topics in Theoretical Computer Science, where problems are analysed with respect to their time and space complexities. There are several Complexity Classes, which are not restricted to classical computation. Moreover, understanding some of the QC Complexity Classes is important, as there are many misconceptions about what a (fault-tolerant) quantum computer will be able to do.

One of the main misunderstandings in QC, caused by the so-called *Quantum-Hype*, is that quantum computers will be able to solve NP-complete problems in polynomial time. The NP Class stands for the family of problems that can be solved in polynomial time by a nondeterministic Turing Machine, or their answer can be checked by a deterministic polynomial time verifier [17]. Moreover, a NP problem B is said to be NP-complete if

1. B is in NP;
2. every other problem A in NP is polynomial time reducible to B [17];

which means that not every problem in NP is NP-complete. Additionally, if a NP-complete problem such as the Traveling Salesperson Problem (TSP) was to be solved in deterministic polynomial time, as every other NP problem is reducible to it, this would mean that P = NP.

Without diving into the specifics, there are two important Complexity Classes for quantum computers. The first, Bounded-error Quantum Polynomial (BQP), is the family of problems that are solved in polynomial time by a quantum computer [16; 18]. It is believed that there is an intersection between NP and BQP [16], where, for example, the integer factorization falls, as shown by Shor's algorithm [5].

Not every problem can be solved in polynomial time by a quantum computer, which leads us to the second class: Quantum Merlin Arthur (QMA). Also known as Bounded-error Quantum Nondeterministic Polynomial (BQNP), according to Bharti *et al* [16], it is the quantum analog for the NP class. Figure 3.4, from Bharti *et al* [16] depicts how these classes are believed to be arranged. Although it is not expected for quantum computers to provide exponential speedup for NP-complete problems, they could achieve faster solutions in some cases [16].

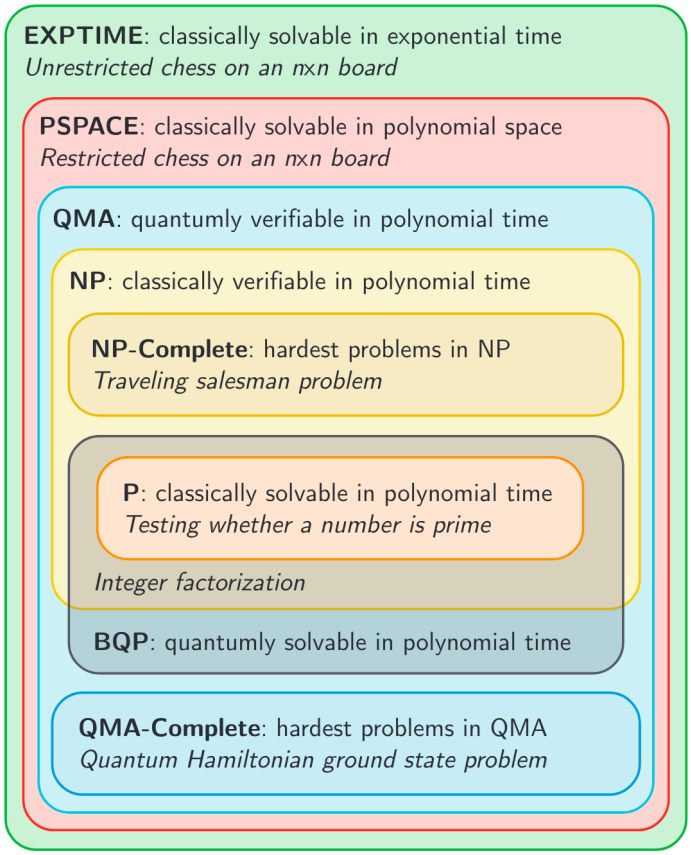

Figure 3.4: Arrangement of some complexity classes
Source: *Noisy Intermediate-scale quantum (NISQ) algorithms*[16]

3.3 Quantum Computer Architectures

Scott Aaronson, when comparing QC architectures, once said:

"is sort of like visiting a nursery school to decide which of the toddlers will become basketball stars"[1]

Envisioning the technological advances that would come with performing a BQP algorithm in a fault-tolerant or noise-resilient setting, Quantum Computing companies are pursuing different quantum computer architectures, that differ mainly on the qubit implementation. Google and IBM, for instance, work on machines with superconducting qubits. On the other hand, IonQ and Honeywell are developing trapped-ion hardware. Figure 3.5 gives a brief overview of the current scenario.

Figure 3.5: Variety of quantum computer architectures

This variety of hardware implementations is due to the fact that it is yet unknown which qubit implementation is the best one. Each architecture come with its pros and cons, considering the limitations of NISQ computers. In order to compare the advancements of different hardware, IBM designed a metric known as Quantum Volume (QV) [19].

QV is a measure of the largest square circuit (n qubits by n gates) of random two-qubit gates that a processor can run successfully, considering a confidence interval. For example, if a quantum computer can use 10 qubits to execute 10 gates successfully, it would achieve a QV of 1024 (2 to the power of n qubits). Figure 3.6 displays the evolution of the QV on different iterations of IBM systems.

[1] "How to evaluate computers that don't quite exist" - Science (link).

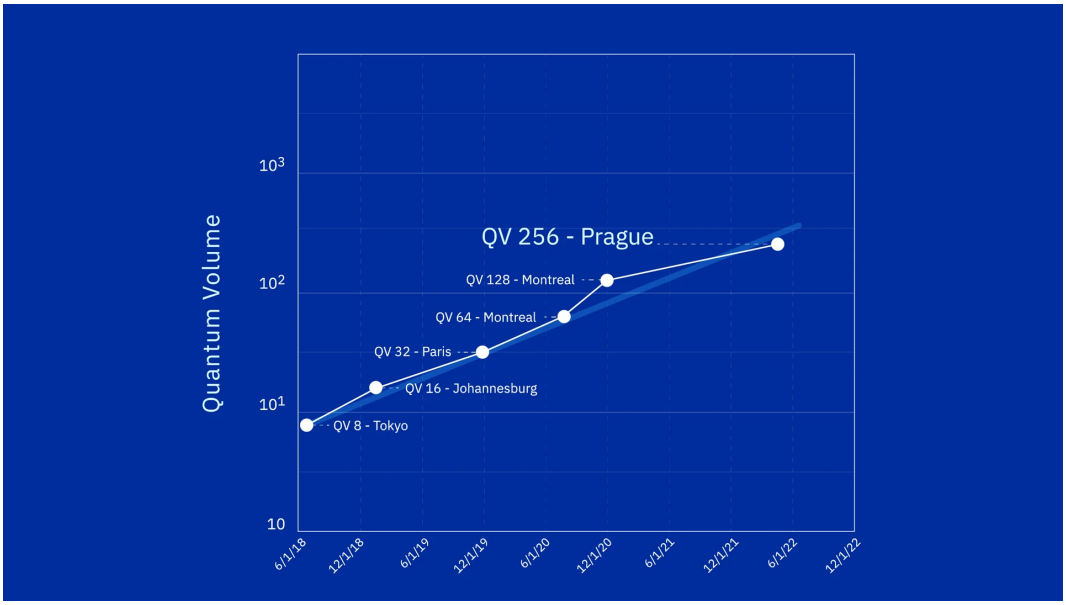

Figure 3.6: Quantum Volume evolution of IBM processors over time
Source:IBM Research Blog

3.4 The State of Quantum Software

Just as hardware architectures, there is a variety of software packages for working with QC. Most of them are developed in Python, such as Qiskit [20] from IBM and Cirq [21] from Google. Microsoft, however, has developed its own language for programming quantum computers, known as Q#.

Moreover, there are some Julia packages for working with quantum computing that are being released, giving shape to Julia's quantum ecossystem. Amazon and QuEra, for example, have respectively released Braket.jl and Bloqade.jl for interfacing with their computers. There is also a famous package in the Julia community named Yao.jl[22], created for working with quantum algorithms. Additionally, as Quantum Optimization is one of the most important applications for QC, some new software is being developed for this purpose in Julia. The company PSR, for instance, has released a software stack (ToQUBO.jl[23], QUBODrivers.jl[24] and QUBOTools.jl[25]) for translating general optimization problems into a quantum computer-compatible format and interfacing with different hardware architectures to solve them.

As not every company in the QC environment invests on building hardware, an important service that allows access to different quantum computers are cloud-based platforms, that work as interface between users and quantum devices. Some corporations that make quantum computers, such as IBM and Google, provide their own computing service. However, the majority rely on connecting their hardware on third-party platforms, such as Amazon's AWS

Braket and Microsoft's Azure Quantum. In Figure 3.7 it is presented the pricing for different quantum computers available in AWS Braket.

Quantum Computers

Hardware Provider	QPU family	Per-task price	Per-shot price
IonQ	IonQ device	$0.30000	$0.01000
OQC	Lucy	$0.30000	$0.00035
Quera	Aquila	$0.30000	$0.01000
Rigetti	Aspen-M	$0.30000	$0.00035
Xanadu	Borealis	$0.30000	$0.0002

Figure 3.7: AWS Braket pricing for different quantum computers
Source: AWS Braket

Another interesting business model is software solutions that use quantum or quantum-inspired hardware. Strangeworks[2] and Zapata Computing[3] have been developing all-in-one software for writing code, communicating with different cloud-platforms and presenting data, beyond other features.

3.5 Near-term algorithms

From the disclosed constraints of current quantum devices, there is still a long road ahead until fault-tolerant hardware. Although some might think that quantum computers will be unuseful for more than a decade, researchers have been developing tailored algorithms for error-prone machines, instigated by the possible speedups from the BQP complexity class.

These NISQ algorithms account for the limited number of qubits, their connectivity and errors during execution. That being said, they are mainly based on shallow circuits that avoid the introduction of errors. In addition, the majority of NISQ algorithms rely on a hybrid classical-quantum setting, where some hard parts for a quantum computer are executed in a classical machine. A notable group of algorithms that fulfill the requirements for NISQ hardware is the class of Variational Quantum Algorithms (VQA) [7], which will be the next topic of discussion.

[2]Strangeworks platform: (link).
[3]Zapata Computing platform: (link).

4
Variational Quantum Algorithms

VQAs consist of parameterized circuits which relie on classical optimization to update the value of their set of parameters (one or more) θ. Outsourcing the parameter update to a classical computer spares the quantum computer from executing lengthy circuits, mitigating errors.

There are three main building blocks for a VQA: the Cost Function, the Ansatz and the Classical Optimizer. When designing a VQA to solve a problem, one must perform the following steps: (i) define a cost function $C(\theta)$; (ii) design an ansatz (where the parameterized operations will take place); (iii) set an optimizer to iteratively find the θ which minimizes the cost function $C(\theta)$ - Equation 4-1.

$$\theta^* = \min_{\theta} C(\theta) \tag{4-1}$$

Figure 4.1 illustrates the schematics of a general VQA. In this chapter, each important component of VQAs will be briefly explained in more detail.

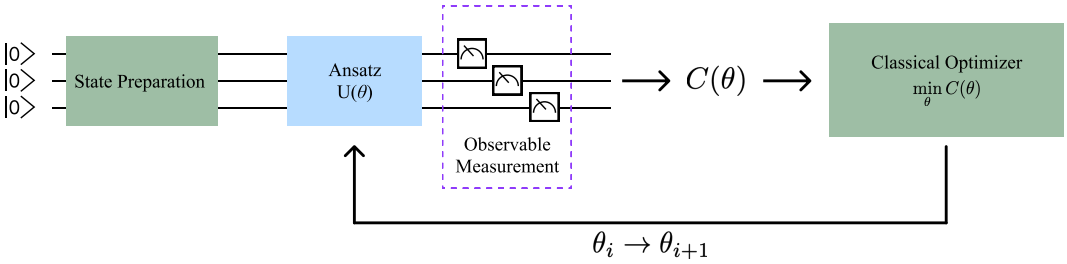

Figure 4.1: General workflow of a VQA

4.1
Cost Function

The cost function encodes the problem, mapping the parameters θ to a real number. According to Cerezo *et al* [7], a cost function should meet the following requirements:

1. Its minimum - $C(\theta^*)$ - should correspond to the minimum of the problem at hand;

2. A quantum computer should be able to calculate $C(\theta)$ efficiently, with the support of a classical computer;

3. It should be "operationally meaningful", so that smaller values indicate better solutions;

4. One must be able to efficiently optimize θ, which means that the cost function needs to be "trainable";

Considering the second item, VQAs are currently a highly discussed topic due to the expectation that the problems that they address cannot be efficiently computable on classical computers. With that said, although VQAs have some challenges to overcome, which will be discussed in this work, it is expected that they will be the key to near-term *Quantum Advantage*.

There is more than one way of defining $C(\theta)$, as presented in Equation 4-2, which will depend on the task. In the equation presented below, ρ_k is the input state, O_k an observable and $U(\theta)$ the Ansatz. Moreover, the trace of the product between O_k, $U(\theta)$, ρ_k and $U(\theta)^\dagger$ is the expectation value for the observable O_k.

$$C(\theta) = \sum_k f_k \Big(Tr[O_k U(\theta) \rho_k U^\dagger(\theta)] \Big) \tag{4-2}$$

4.2 Ansatz

The ansatz is the parameterized section of a VQA's circuit. Its design impacts on how "trainable" the θ parameters can be. These parameters are usually encoded in an unitary $U(\theta)$, which can be comprised of several other unitaries, each with a different parameter, as presented in Equation 4-3. These unitaries can be perceived as gates in a quantum circuit, as discussed during Chapter 2, in Figure 2.3. Additionally, the resulting operator $U(\theta)$ can be decomposed into a set of feasible quantum gates on a quantum computer, as it was described for the case of the Hamiltonian, in Equation 2-14.

$$U(\theta) = U_n(\theta_n) \dots U_2(\theta_2) U_1(\theta_1) \tag{4-3}$$

An ansatz can be constructed considering the problem definition or not, resulting in two classes: problem-inspired ansatze and hardware-efficient ansatze. Hardware-efficient ansatze are built considering the set of possible gates in a certain quantum computer and the connectivity between its qubits (see Figure 3.1). Such approach yields in a smaller circuit-depth, and thus, less errors, considering NISQ hardware. On the other hand, problem-inspired ansatze, by not regarding a device's restrictions, generates larger circuits, leading to more inaccuracy.

There are some metrics used to evaluate an ansatz. An important one is the *Expressibilty*, related to how uniformly one can access the possible states. A second one is the *Treinability* [26; 27].

4.3 Evaluating the cost function

As mentioned, VQAs delegates the task of updating the θ parameters to a classical optimizer. However, before doing so, one must be able to evaluate the cost function in order to feed information to the optimizer. A potencial approach is by finding the cost function gradient, which can, fortunately, be analytically calculated by the VQA. One way of finding the gradient of the cost function is by the parameter-shift rule [7].

4.4 Optimizer

As discussed, the problems that VQAs are envisioned to address cannot be efficiently calculated by a classical computer, as they are, in general, NP-Hard. This arises from the fact that their cost functions involve many local-minima [7]. Despite this predicted gain, VQAs are yet to prove experimental advantage, as there are some challenges to overcome.

Some of these issues are related to the optimizer's task to update the θ parameter. As expected, when working with NISQ hardware, the outcomes from evaluating the cost function are affected due to noise and a limited amount of measurements that can be performed. Additionally, there are the so-called barren plateaus, which are large regions where the cost function gradient is nearly zero [28]. Such issues lead to a stochastic environment, that can be troublesome for some optimizing methods.

That being said, when working with VQAs, one has to use a "quantum-aware" optimizer [7], that takes into account the current limitations of NISQ devices. One of the most prominent methods is the Simultaneous Perturbation Stochastic Approximation (SPSA) [29] optimizer, which will be explored in the experimental part of this work.

5
Variational Quantum Eigensolver

One of the biggest promises of QC is its application in Quantum Simulation, which will bring advances in many areas of research, such as Quantum Chemistry. Being able to extract information from molecules and atoms is of great importance, as it allows the discovery of new drugs and materials, for example. Simulating quantum systems in classical computers is a great challenge, as the dimension of the problem grows exponentially with respect to the size of the system, becoming infeasible for molecules with a large number of atoms [4; 28].

Within this context, some algorithms to tackle Quantum Chemistry problems have already been presented [28], such as the Quantum Phase Estimation (QPE) algorithm [9; 28]. The QPE algorithm allows one to calculate the phase or eigenvalue for a given eigenvector from a quantum system's Hamiltonian, which is a hard problem in the classical setting. Unfortunately, the QPE method is far from being useful for quantum computers prone to noise, as it requires full coherent execution.

Having said that, the Variational Quantum Eigensolver (VQE) [8] is presented as a VQA NISQ-alternative for the QPE algorithm. Being developed under the NISQ paradigm, the VQE is more robust to errors, using shallow circuits and aid from a classical optimizer. Similarly to the QPE, this variational algorithm has the purpose of finding the eigenvalue associated to the ground state of a quantum system.

One of the greatest assets of the VQE is its flexibility architecture-wise. A vast number of VQE experiments has been performed on different quantum hardware, such as photonic [8], trapped-ion [30] and superconducting [31]. Moreover, its inherent modularity from being a VQA allowed researchers to propose enhancements[32; 33; 34], aiming to a more reliable and efficient algorithm.

5.1
Algorithm Overview

As the VQE is applicable on distinct quantum computers, there is a variety of implementations that differ mainly on the ansatz formulation. In this section it will be presented a very general overview of the algorithm functionality, making a parallel with what was presented in Chapter 4.

Let H be the Hamiltonian associated with a molecule. For a given state $|\psi\rangle$, one can find the eigenvalue associated to this state (when it is an eigenstate), which will always be greater or equal to the minimum eigenvalue

of H (λ_{min}). Additionally, as we are considering the lowest eigenvalue of a molecule, it follows that λ_{min} is the lowest energy of the molecule (E_{min}). This is illustrated in Equation 5-1.

$$\langle\psi| H |\psi\rangle = \lambda_{|\psi\rangle} \geq \lambda_{min} = E_{min} \tag{5-1}$$

Now, if we consider a parameterized state $|\psi(\theta)\rangle$, we will end up with the following:

$$\langle\psi(\theta)| H |\psi(\theta)\rangle = \lambda_\theta \tag{5-2}$$

Therefore, one can formulate the optimization problem to find E_{min}, as presented in Equation 5-3. Once E_{min} is found, so is the eigenstate (ground state) associated to it.

$$\min_\theta \langle\psi(\theta)| H |\psi(\theta)\rangle \tag{5-3}$$

The parameterized state $|\psi(\theta)\rangle$ is formulated by an ansatz, as presented in Chapter 4. The choice of the ansatz is very important, as it can improve the optimization of θ. Just as most of VQAs, it is also possible to choose between problem-inspired and hardware-efficient ansatze.

In summary, the VQE has a cost function, which is the system's energy for a given θ (Equation 5-4), a chosen ansatz which yields $|\psi(\theta)\rangle$ and an optimizer that updates θ in order to minimize $E(\theta)$, according to Equation 5-3. After reaching a convergence requirement, the VQE algorithm finishes.

$$C(\theta) = E(\theta) = \langle\psi(\theta)| H |\psi(\theta)\rangle \tag{5-4}$$

5.2 Notes on the molecular Hamiltonian

Working with the molecular Hamiltonian poses a challenge, as the number of electrons and nuclei in the molecule increases the complexity of the calculations. That being said, the problem can be simplified with the Born-Oppenheimer Approximation (BOA) [35], where the nuclei are treated as stationary particles. This method yields in the following electronic Hamiltonian:

$$H_{el} = T_e(\mathbf{r}) + V_{eN}(\mathbf{r}, \mathbf{R}) + V_{NN}(\mathbf{R}) + V_{ee}(\mathbf{r}), \tag{5-5}$$

where:

- $\mathbf{R}$ represents all cartesian coordinates ($\mathbb{R}^3$) of the nuclei
- $\mathbf{r}$ represents all cartesian coordinates ($\mathbb{R}^3$) of the electrons

- $\hat{T}_e(\mathbf{r})$ is the kinetic energy of the electrons
- $V_{eN}(\mathbf{r}, \mathbf{R})$ is the potential energy between the electrons and the nuclei
- $V_{NN}(\mathbf{R})$ is the nuclear-nuclear interaction energy
- $V_{ee}(\mathbf{r})$ is the electron-electron interaction energy

In order to use QC for Chemistry, the molecule Hamiltonian is usually represented in the second quantization [35], where the fundamental particles are perceived as excitations of a field. The number of excitations in this configuration can be altered with the notion of electron density and the fermionic operators of creation and annihilation, $\{a^\dagger, a\}$. These operators can change the electron density at a given position.

That being said, the second quantization of the Hamiltonian is as presented in Equation 5-6, where the first summation depicts the nuclei-electrons interactions, while the second portrays the eletron-electron interactions. It is worth mentioning that both coefficients, h_{pq} and h_{pqrs}, can be classically calculated.

$$H = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s \tag{5-6}$$

5.3 Encoding the Molecular Hamiltonian

One of the greatest challenges when it comes to simulating molecules on a quantum computer, is to map their fermionic system, as portrayed in Equation 5-6, into qubits [36]. That being said, this encoding is a mapping between the fermionic Fock space to the Hilbert space of qubits, where each fermionic state can be portrayed by a qubit state.

Besides having noisy devices, the NISQ era is also characterized by quantum computers with a limited number of qubits. That being said, when encoding a molecule Hamiltonian into qubits, one should take into account the number of qubits required for such operation.

One of the most simple fermionic to qubit mappings is the Jordan-Wigner transformation [37], where the second-quantized Hamiltonian, from Equation 5-6, is mapped into a sum of Pauli matrices products, as presented in Equation 5-7.

$$H = \sum_j h_j \prod_i \sigma_i^j = \sum_j h_j P_j, \tag{5-7}$$

where:

- h_j is a scalar coefficient

- σ_i^j is a Pauli operator (I, X, Y, Z)
- P_j is a Pauli string, correspondent to the tensor product of Pauli matrices (e.g. $X \otimes Y \otimes X$)
- i is the qubit that the operator is acting on
- j is the term of the Hamiltonian

It is worth noting that the Pauli operator basis spans the space of operators in $\mathcal{H}^2$, i.e., every one-qubit gate can be represented by a linear combination of Pauli matrices. Furthermore, in the case of composite systems, as presented in Section 2.4, it is possible to show that an operator acting on a multi-qubit state corresponds to the tensor product of one-qubit operators. Thus, Equation 5-7 demonstrates that the Hamiltonian can be decomposed into a linear combination of tensor products of one-qubit Pauli operators.

The software package that was used to conduct the experiments in Chapter 7, Qiskit, has a set of built-in functions for qubit mapping, based on well known methods, such as the Jordan-Wiegner [37] and the Parity [38] mapping.

The following code snippet shows how Qiskit can map the Hamiltonian of the H_2 molecule, which will be later analysed, into a set of gates.

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import JordanWignerMapper

driver = PySCFDriver(
        atom="H .0 .0 .0; H .0 .0 0.7414", # cartesian coordinates
        unit=DistanceUnit.ANGSTROM,
        basis='sto3g'
    )

problem = driver.run()
fermionic_operator = problem.hamiltonian.second_q_op()
mapper = JordanWignerMapper()
operator = mapper.map(fermionic_operator)
```

Listing 1: Qiskit code to map a molecule Hamiltonian into a qubit operator

If we were to print this operator, a set of quantum gates will be returned with some coefficients, as presented below. It is possible to see that the Jordan-Wigner mapping method yielded in four qubits.

```
SparsePauliOp(['IIII', 'IIIZ', 'IIZI', 'IIZZ', 'IZII', 'IZIZ', 'YYYY',
    'XXYY', 'YYXX', 'XXXX', 'ZIII', 'ZIIZ', 'IZZI', 'ZIZI', 'ZZII'],
coeffs=[-0.81261796+0.j,  0.17119775+0.j, -0.22278593+0.j,  0.12054482+0.j,
  0.17119775+0.j,  0.16862219+0.j,  0.0453222 +0.j,  0.0453222 +0.j,
  0.0453222 +0.j,  0.0453222 +0.j, -0.22278593+0.j,  0.16586702+0.j,
  0.16586702+0.j,  0.17434844+0.j,  0.12054482+0.j])
```

Listing 2: Resulting Hamiltonian from Jordan-Wigner encoding in Qiskit

Rearranging the outcome, one can see that the resulting Hamiltonian is as follows in Equation 5-8. Note that the resulting Hamiltonian is a 16x16 matrix, that can be represented in a quantum circuit by a sequence of Pauli gates.

$$\begin{aligned} H_{JW} = &- 0.813 I_3 \otimes I_2 \otimes I_1 \otimes I_0 + 0.171 I_3 \otimes I_2 \otimes I_1 \otimes Z_0 \\ &- 0.223 I_3 \otimes I_2 \otimes Z_1 \otimes I_0 + 0.1713 I_3 \otimes Z_2 \otimes I_1 \otimes I_0 \\ &- 0.223 Z_3 \otimes I_2 \otimes I_1 \otimes I_0 + 0.121 I_3 \otimes I_2 \otimes Z_1 \otimes Z_0 \\ &+ 0.169 I_3 \otimes Z_2 \otimes I_1 \otimes Z_0 + 0.045 Y_3 \otimes Y_2 \otimes Y_1 \otimes Y_0 \\ &+ 0.045 X_3 \otimes X_2 \otimes Y_1 \otimes Y_0 + 0.045 Y_3 \otimes Y_2 \otimes X_1 \otimes X_0 \\ &+ 0.045 X_3 \otimes X_2 \otimes X_1 \otimes X_0 + 0.166 Z_3 \otimes I_2 \otimes I_1 \otimes Z_0 \\ &+ 0.166 I_3 \otimes Z_2 \otimes Z_1 \otimes I_0 + 0.174 Z_3 \otimes I_2 \otimes Z_1 \otimes I_0 \\ &+ 0.121 Z_3 \otimes Z_2 \otimes I_1 \otimes I_0 \end{aligned} \tag{5-8}$$

5.4 Evaluating the Cost Function

As previously mentioned, the ansatz for a VQE instance can be either hardware-efficient or problem-inspired. That being said, as the latter uses information about the model to formulate the trial state, for the case of the VQE algorithm with a molecule, it uses information from its Hamiltonian. Additionally, in this case, an approximated initial guess of the wave-function, which uses the Hartree-Fock method [28].

After preparing the Hamiltonian and choosing an ansatz, one can calculate a state $|\theta\rangle$ and estimate the energy of the system with the quantum computer. The energy can be found with Equation 5-9, where each term $\prod_i \sigma_i^j$ need to be measured enough times, according to a desired precision. This technique was suggested by Peruzzo *et al* [8] and is known as Hamiltonian Averaging.

$$E(\theta) = \sum_j h_j \langle\psi(\theta)| \prod_i \sigma_i^j |\psi(\theta)\rangle \tag{5-9}$$

6
Introduction to IBMQ and Qiskit

In the next chapter, it will be presented some experiments that were made with the VQE algorithm, with simulators and real quantum computers. In this chapter it will be briefly introduced the IBM Quantum platform, through which the quantum computers can be accessed, and the Qiskit framework.

6.1
IBM Quantum platform

Since 2016, IBM provides a cloud service (IBMQ) where anyone can send quantum circuits to be executed in real quantum hardware. Although most computers have limited access, requiring a subscription or some kind of research license, there are some real quantum systems available for free at IBMQ.

Despite the fact that the open access hardware have smaller number of qubits and Quantum Volume, compared to the more advanced ones, they are enough to run small VQE instances.

When a circuit is submmited to IBMQ to run on a device, it enters an execution queue as a *job*. The time one can wait to get their results back varies, from instants to hours, depending on how long the queue is. Another factor for waiting time is that researchers and some companies with more privileged access can reserve a computer for a period of time, pausing requests from others. In Figure 6.1 it is possible to see that `ibm_perth` is reserved in the *Status* column.

Name	Qubits ↓	QV	CLOPS	Status	Total pending jobs	Processor type	Plan	Features
ibm_perth	7	32	2.9K	Online - Reserved	390	Falcon r5.11H	open	OpenQASM 3
ibm_lagos	7	32	2.7K	Online	66	Falcon r5.11H	open	OpenQASM 3
ibm_nairobi	7	32	2.6K	Online	336	Falcon r5.11H	open	OpenQASM 3
ibmq_jakarta	7	16	2.4K	Online	41	Falcon r5.11H	open	OpenQASM 3
ibmq_manila	5	32	2.8K	Online	41	Falcon r5.11L	open	OpenQASM 3
ibmq_quito	5	16	2.5K	Online	13	Falcon r4T	open	
ibmq_belem	5	16	2.5K	Online	2	Falcon r4T	open	
ibmq_lima	5	8	2.7K	Online	15	Falcon r4T	open	

Figure 6.1: IBMQ free-access quantum computers

6.2
Qiskit

To submmit a *job* to IBMQ one has to use Qiskit, IBM's framework for programming on quantum computers and simulators. Qiskit is a very extensive

package, having several modules, available open-source, in its GitHub page[1].

Qiskit democratizes the access to Quantum Computing by providing built-in functions that provide an abstraction layer to what is actually happening in the quantum hardware. Furthermore, it also has some more lower-level methods such as Qiskit *pulse*[39], for more control of the hardware.

6.2.1 Qiskit Runtime

As previously mentioned, the IBMQ platform runs quantum circuits uploaded by users. However, as seen in Figure 4.1, VQAs need that quantum and classical hardware communicate to update the parameters of the ansatz.

As it would be a very time-consuming task to keep uploading locally-updated quantum circuits, Qiskit provides a wrapper called Qiskit Runtime, that does the whole work for the user. So for a VQE execution, instead of uploading a single *job*, one can send a Qiskit Runtime instance as a *job*, that will keep getting the results from quantum hardware execution and sending it to classical optimizers, to later re-send the updated quantum circuit. It is important to say that the new quantum circuits will not be treated as a new *job* that needs to get to the last position in line. It goes instantly to the first position of the queue.

In other words, Qiskit Runtime is a container that has all dependencies to run the VQE (or even other quantum programs) and creates a path between quantum and classical hardware. Figure 6.2, from Qiskit's blog [2], provides a graphical representation of Qiskit's workflow to send a Qiskit Runtime instance to the cloud. Additionally, Qiskit has some *Primitive* programs that simplifie the experiment output according to the algorithm being executed.

For instance, after the VQE optimization problem converges, the *Estimator* primitive runs a post-processing routine to return the minimum eigenvalue, the θ parameters from the ansatz that yielded in this result, among with other useful data.

[1] https://github.com/Qiskit

[2] https://medium.com/qiskit/so-what-is-qiskit-runtime-anyway-c78aecf3742

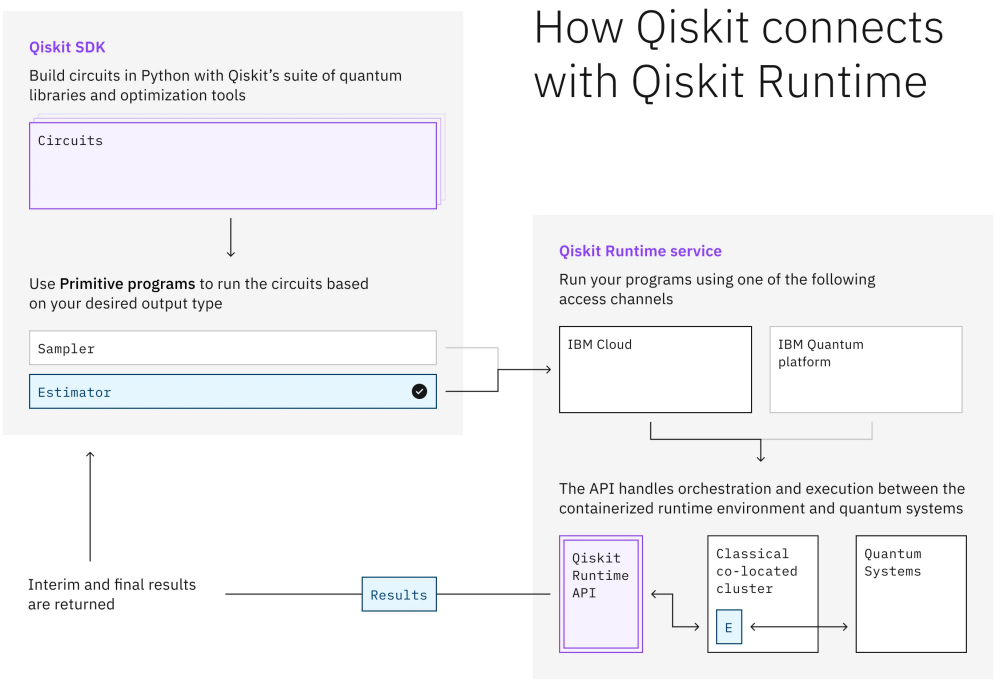

Figure 6.2: Qiskit Runtime's workflow
Source: Qiskit Blog

7
Experiments

For better understanding the VQE algorithm, this chapter presents experiments performed for the H_2 and LiH molecules. These molecules were chosen because they are simple enough to run VQE instances on classical simulators. As it is not the purpose of this project, some Chemistry concepts will be only briefly introduced to give more meaning to the results of the experiments.

Furthermore, as discussed in previous chapters, being a VQA, the VQE can be seen as a "modular" algorithm. That being said, some of the chosen components for this method will be concisely discussed.

7.1
Chosen components

For both simulation and real experiments, it was used Qiskit's `ParityMapper` to encode the fermionic Hamiltonian into qubits. Yet, for the LiH molecule, another approximation was required, as it is a more complex molecule. Besides `ParityMapper`, it was employed Qiskit's `FreezeCoreTransformer`, that removes some electrons from the representation.

Considering that the experiments are focused on small molecules, it was possible to use a problem-inspired ansatz known in Qiskit as `UCCSD`, which is a variation of the UCC ansatz, for the simulation part. When sending VQE instances to real quantum computers, however, hardware-inspired ansatz were required, to mitigate decoherence and gate errors. This ansatz is known in Qiskit as `EfficientSU2`.

Finally, for all experiments, it was used the SPSA optimizer, previously mentioned in Chapter 4, that is implemented in Qiskit.

7.2
Finding the optimal interatomic distance

The potential energy between two atoms is associated with their interatomic distance (bond length). The lowest potential energy of a molecule occurs at its ground state. At this point, the interatomic distance is optimal, meaning that the attractive and repulsive forces between the atoms make up to the greatest attractive force.

Using VQE, we can find the lowest energy between two atoms for a given interatomic distance. In this first experiment, the VQE was performed for different interatomic distances for the H_2 and LiH molecules in order to find their

ground state energy and, consequently, their optimal interatomic length. Thus, for each distance, it was created an iteration of the VQE algorithm, where the value of θ was optimized for the given molecule configuration. Gathering all VQE iterations results gives us the ground-state energy dissociation curve of the molecule [28], where it is possible to find the overall ground state energy.

To compare the results, it was used experimental results available at NIST's Computational Chemistry Comparison and Benchmark DataBase (CCCBDB)[1].

The first step to execute a VQE instance on Qiskit is to map the molecule to a circuit operator. For that, Qiskit provides a wrapper for a Python Computational Chemistry package called PySCF [40], where you can instantiate a molecule and get its corresponding fermionic operator with the following code.

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import ParityMapper

driver = PySCFDriver(
    atom="H 0.0 0.0 0.0;H 0.0 0.0 0.7", # cartesian coordinates of the atoms
    basis="sto3g", unit=DistanceUnit.ANGSTROM,
)
problem = driver.run()
operator = problem.hamiltonian.second_q_op()
mapper = ParityMapper(num_particles = problem.num_particles)

qubit_op = mapper.map(second_q_op)
```

Listing 3: Qiskit code to instantiate a molecule and map the fermionic operator

This fermionic operator is not yet ready to be applied on qubits, as previously seen. That being said, it was used the `ParityMapper` from Qiskit.

After mapping the Hamiltonian, it is already possible to instantiate the ansatz that will be sent to the VQE algorithm. The `UCCSD` ansatz requires a first guess about the state, which can be found classically with the `HartreeFock` method, as seen in the following code.

[1] https://cccbdb.nist.gov/introx.asp

```
from qiskit_nature.second_q.circuit.library import HartreeFock

init_state = HartreeFock(
    problem.num_spatial_orbitals,
    problem.num_particles,
)
```

Listing 4: Qiskit code to instantiate a `QubitConverter` and an initial Hartree-Fock state

After that, it is possible to set an ansatz to represent the problem. For the first experiment the VQE was simulated on a classical computer. Having said that, as the molecules being studied are not too complex, it is possible to use a problem-inspired ansatz, such as the Unitay-Coupled-Cluster ansatz, which can give us more exact results.

```
from qiskit_nature.second_q.circuit.library import UCCSD
ansatz = UCCSD(
    qubit_mapper = mapper,
    num_particles= problem.num_particles,
    num_spatial_orbitals = problem.num_spatial_orbitals,
    initial_state = init_state
)
```

Listing 5: Qiskit code to instantiate an ansatz

Finally, the last step to run a VQE instance is to set a Qiskit `Estimator` to estimate the expectation values from the circuit and an optimizer to minimize the potencial energy of the molecule. After that, everything is ready to perform the VQE algorithm, as presented in the following code.

```
from qiskit.algorithms.optimizers import  SPSA
from qiskit.primitives import Estimator
from qiskit_nature.second_q.algorithms import VQEUCCFactory
from qiskit_nature.second_q.algorithms import GroundStateEigensolver

estimator = Estimator()
optimizer = SPSA(maxiter=800)

vqe = VQEUCCFactory(estimator,ansatz, optimizer)
calc = GroundStateEigensolver(converter, vqe)
vqe_run = calc.solve(problem)
```

Listing 6: Qiskit code to instantiate an `Estimator`, an optimizer and run VQE

The results for the experiments are presented in Figures 7.1 and 7.2. Comparing the calculated ground state energies with CCCBDB results, we can find the following error percentage in Table 7.1.

Molecule	CCCBDB(Hartrees)	VQE Simulation(Hartrees)	Error %
H_2	-1.137306	-1.137306	0
LiH	-7.882134	-7.882751	0.486328

Table 7.1: VQE Simulation results comparison with CCCBDB

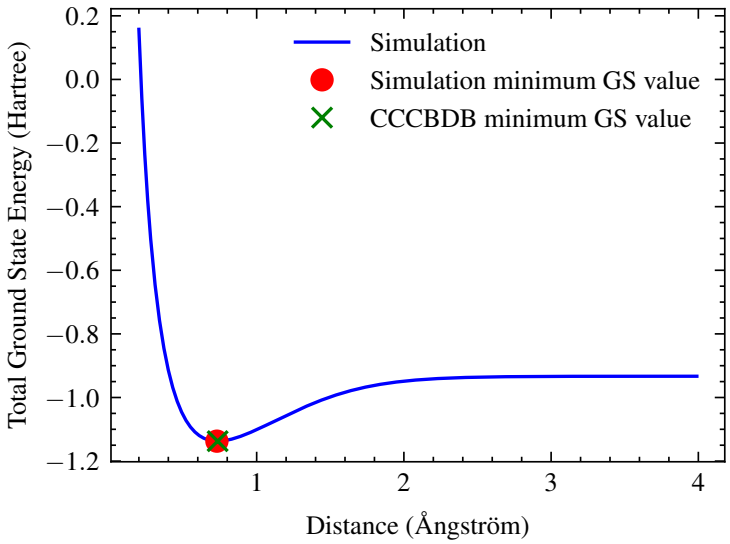

Figure 7.1: Total Ground State Energy (Hartree) of the H_2 molecule according to its interatomic distance (Å)

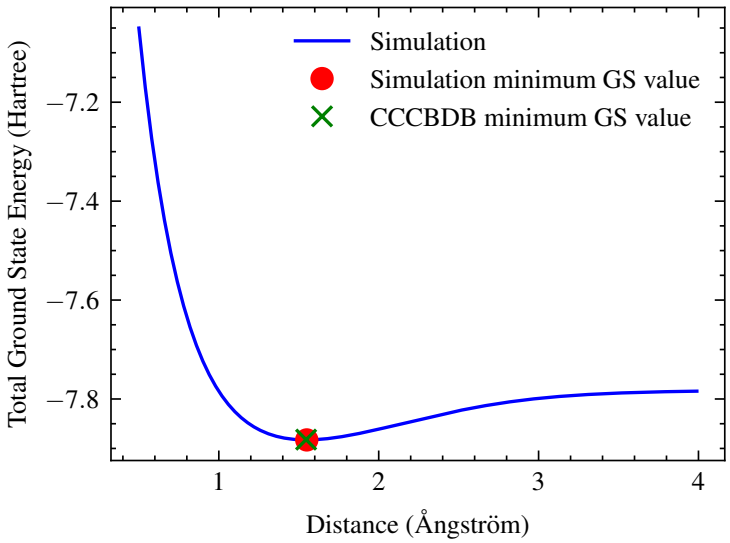

Figure 7.2: Total Ground State Energy (Hartree) of the LiH molecule according to its interatomic distance (Å)

7.3 Finding the optimal interatomic distance with real quantum computers

Thus far, we have executed VQE simulations on a classical computer. For the purpose of comparison, the same experiment was also conducted on real quantum hardware. Using Qiskit, we are able to send jobs to some real quantum computers that IBM gives access for free, as seen in Chapter 6. To avoid any misunderstanding, the results from this analysis do not indicate the current state of quantum hardware, as the free systems that IBM provides are not the most recent ones, which are only available in the premium plan, as also seen in Chapter 6. Although the results were very satisfactory, they could have been more accurate with more advanced devices.

The following quantum computers were chosen to run the experiments: `ibm_perth`, `ibm_nairobi` and `ibmq_lima`. Their specifications are presented in Table 7.2

Name	Qubits	Quantum Volume
`ibm_perth`	7	32
`ibm_nairobi`	7	32
`ibmq_lima`	5	8

Table 7.2: Technical details for the selected quantum computers

The code presented in Listings 3, 4, 5 and 6 had to be modified as Qiskit's methods to send VQE instances to the cloud require different parameters. Moreover, we are not able to use a problem-inspired ansatz, as it would result in a large circuit depth. That being said, the ansatz used had to be a hardware-inspired one.

Qiskit Runtime, presented in Chapter 6, was used to simplify the experiment workflow and get the results faster. Moreover, as the VQE was going to be executed on real devices, the qubit-encoding method had to be different to actually fit within the hardware's constraints. In the case of the LiH molecule, that will be presented later, it was necessary to abdicate of some information about the molecule, to get a smaller number of qubits to represent it.

7.3.1 H_2 molecule

Before sending a Qiskit Runtime instance, it is necessary to load your IBMQ account. Then, you can set the desired quantum computer that you want to use and instantiate a Qiskit Estimator with this backend, as presented in the following code.

```python
from qiskit_ibm_runtime import QiskitRuntimeService

service = QiskitRuntimeService()
backend = service.backend('ibmq_lima')
estimator = Estimator(session=backend)
```

Listing 7: Qiskit code connect to IBMQ, set your backend and Estimator

The next step is to set the molecule and get its operator, as seen in Listing 3. In this experiment, instead of using the `JordanWignerMapper` as in Listing 4, it was used the `ParityMapper` that yieled in fewer qubits.

```python
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import ParityMapper

driver = PySCFDriver(
    atom="H 0.0 0.0 0.0;H 0.0 0.7 {}",
    basis="sto3g",
    unit=DistanceUnit.ANGSTROM,
)

problem = driver.run()
second_q_op = problem.hamiltonian.second_q_op()
mapper = ParityMapper(num_particles=problem.num_particles)
```

Listing 8: Qiskit code to map the molecule to qubit-representation

When instantiating the ansatz, as mentioned, it was necessary to choose a hardware-efficient ansatz.

```python
from qiskit.circuit.library import EfficientSU2
ansatz = EfficientSU2(
    num_qubits=qubit_op.num_qubits,
    reps=1,
    entanglement="linear"
)
```

Listing 9: Qiskit code to instantiate a hardware-efficient ansatz

Finally, the last steps are to set an optimizer, an initial point and the VQE instance. After that, everything is ready to use Qiskit Runtime to execute the VQE algorithm.

```
from qiskit.algorithms.minimum_eigensolvers import VQE
from qiskit.algorithms.optimizers import SPSA
optimizer = SPSA(maxiter=50)
initial_point = np.random.random(ansatz.num_parameters)
vqe = VQE(
        estimator=estimator,
        ansatz=ansatz,
        optimizer=optimizer,
        initial_point=initial_point,
        callback=store_intermediate_result
    )
vqe_run = vqe.compute_minimum_eigenvalue(qubit_op)
```

Listing 10: Qiskit code set an optimizer and send a VQE instance to IBMQ

7.3.1.1 Results discussion

For all VQE executions, the optimizer was set to perform 50 iterations, as seen in Listing 9. Comparing the experimental results with the simulation ones in Figure 7.3 and Table 7.3, it is possible to see that all three computers got results very close to the expected ones, for the majority of iterations.

One important observation to be made is that although `ibm_perth` has more qubits and Quantum Volume than `ibmq_lima`, its results were not as good as the latter. It is hard to point out the reason for that, as there are many sources of errors. Among others, the hardware could not have been well calibrated or the qubit mapping to fit the hardware qubit-connectivity constraints could not have been as optimal. Moreover, for the interatomic distance 0.35Å, the result from `ibm_perth` is an outlier.

In order to perceive how the number of iterations for the optimizer impacts on the final result, the same experiment was run in `ibmq_lima` for 25 and 150 iterations. From the presented outcomes in Figure 7.4 and Table 7.3, as expected, 25 iterations yielded in more noticeable errors, while 150 and 50 iterations approximated the experimental results from the simulation ones.

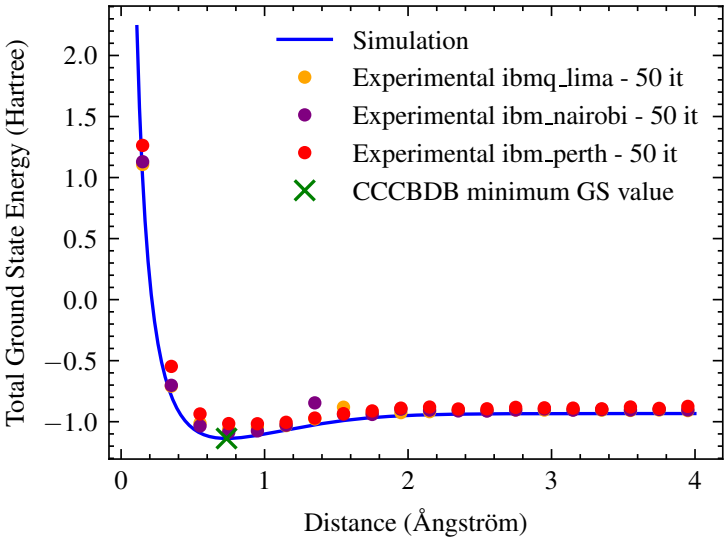

Figure 7.3: Total Ground State Energy (Hartree) of the H_2 molecule according to its interatomic distance (Å) in real quantum computers

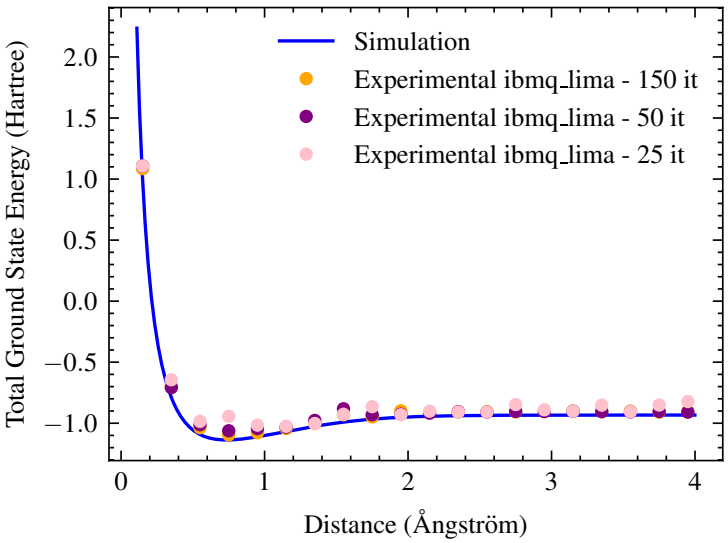

Figure 7.4: Total Ground State Energy (Hartree) of the H_2 molecule according to its interatomic distance (Å) with different optimization steps in `ibmq_lima`

Distance (Å)	`ibm_nairobi` (50 it)	`ibm_perth` (50 it)	`ibmq_lima` (50 it)	`ibmq_lima` (25 it)	`ibmq_lima` (200 it)
0.15	0.12605	0.21849	0.10838	0.10817	0.09036
0.35	0.12461	0.4433	0.11372	0.22616	0.13922
0.55	0.05601	0.16688	0.07967	0.10945	0.05354
0.75	0.04643	0.11989	0.07096	0.20538	0.03787
0.95	0.03355	0.09307	0.06433	0.09449	0.03092
1.15	0.03841	0.06294	0.03487	0.04149	0.02317
1.35	0.21212	0.05669	0.04925	0.02263	0.02235
1.55	0.054	0.06026	0.12396	0.0691	0.06354
1.75	0.02754	0.05999	0.03794	0.11777	0.01918
1.95	0.06031	0.07026	0.02782	0.02222	0.05955
2.15	0.04144	0.07126	0.02719	0.04588	0.03803
2.35	0.02891	0.04717	0.03621	0.03347	0.03332
2.55	0.02511	0.04636	0.02913	0.03004	0.03432
2.75	0.03357	0.05879	0.03405	0.10297	0.03047
2.95	0.04205	0.05355	0.03211	0.04908	0.03455
3.15	0.03103	0.05034	0.03434	0.03524	0.03783
3.35	0.03667	0.04364	0.03053	0.09532	0.03668
3.55	0.03254	0.06114	0.02794	0.0303	0.03646
3.95	0.0327	0.06708	0.02657	0.13522	0.027
3.75	0.03755	0.04761	0.03224	0.09656	0.0317
Average	0.05603	0.09494	0.05106	0.083547	0.0440

Table 7.3: Relative error for the ground state energy - H_2 molecule

7.3.2 LiH molecule

When running VQE instances for the LiH molecule, an extra step was necessary when encoding the fermionic operator into qubits. That being said, it was used Qiskit's `FreezeCoreTransformer` to lower the number of required qubits to perform the computation, as presented in Listing 11. After that, the same code from Listings 9 and 10 was used to run the algorithm.

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import ParityMapper
from qiskit_nature.second_q.transformers import FreezeCoreTransformer

driver = PySCFDriver(
        atom="Li .0 .0 .0; H .0 .0 1.5",
        unit=DistanceUnit.ANGSTROM,
        basis='sto3g'
)
problem = driver.run()

# Map to qubit operators
transformer = FreezeCoreTransformer(freeze_core=True, remove_orbitals=[3, 4])
problem = transformer.transform(problem)
mapper = ParityMapper(num_particles=problem.num_particles)
fermionic_op = problem.hamiltonian.second_q_op()
qubit_op = mapper.map(fermionic_op)
```

Listing 11: Qiskit code to map the molecule to qubit-representation

7.3.2.1 Results discussion

The same experiment conducted for the H_2 molecule was reproduced for the LiH. In order to be able to notice the ground state energy in the plot, it was necessary change the range of the plot, although the same interatomic distances were used. With that said, although the results in Figure 7.2 and 7.6 appear to be very distant from the expected result, they have about the same quality as the ones from the H_2 experiment.

As the LiH molecule is more complex than H_2, it was expected that its results would not be as good as the previous ones. However, as seen from the average relative error in Tables 7.3 and 7.4, the outcomes from the second experiments were, actually, better.

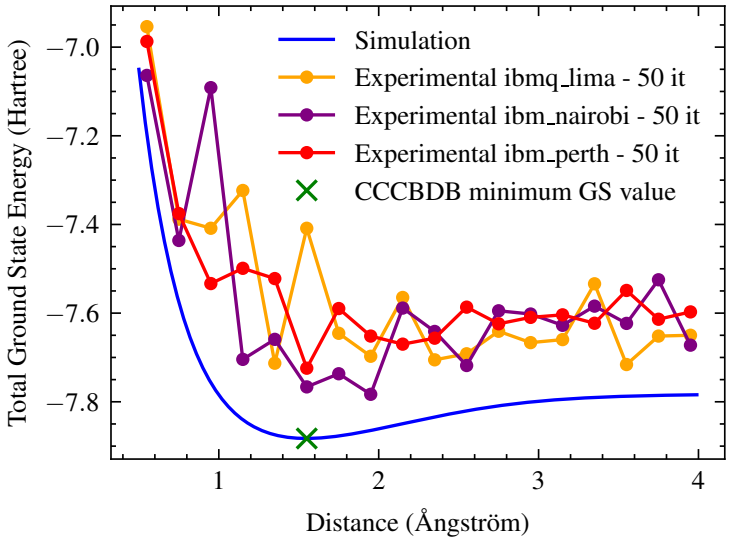

Figure 7.5: Total Ground State Energy (Hartree) of the LiH molecule according to its interatomic distance (Å) in real quantum computers

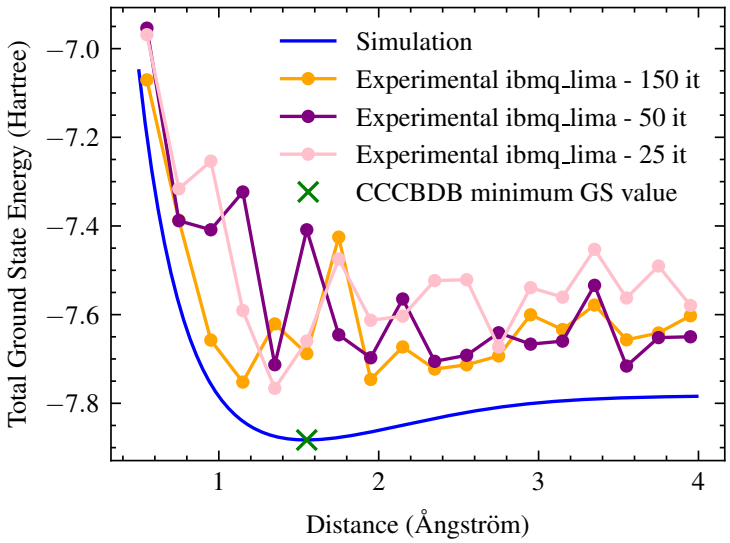

Figure 7.6: Total Ground State Energy (Hartree) of the LiH molecule according to its interatomic distance (Å) with different optimization steps in `ibmq_lima`

Distance (Å)	`ibm_nairobi` (50 it)	`ibm_perth` (50 it)	`ibmq_lima` (50 it)	`ibmq_lima` (25 it)	`ibmq_lima` (150 it)
0.15	0.10763	0.21321	0.23790	0.13221	0.21448
0.35	0.10303	0.03993	0.07406	0.03445	0.07068
0.55	0.01893	0.03010	0.03504	0.03285	0.03447
0.75	0.01875	0.02711	0.02536	0.03543	0.02537
0.95	0.09384	0.02967	0.04702	0.06935	0.04549
1.15	0.01767	0.04555	0.07061	0.03289	0.0667
1.35	0.02812	0.04687	0.02095	0.01396	0.02121
1.55	0.01498	0.02052	0.06400	0.02913	0.06167
1.75	0.01815	0.03788	0.03030	0.05386	0.0312
1.95	0.01049	0.02783	0.02175	0.03312	0.02161
2.15	0.03438	0.02341	0.03766	0.03237	0.03712
2.35	0.02524	0.02321	0.01672	0.04134	0.01669
2.55	0.01325	0.03083	0.01673	0.03978	0.01668
2.75	0.02822	0.02427	0.02201	0.01776	0.02186
2.95	0.02611	0.02509	0.01747	0.03468	0.01762
3.15	0.02188	0.02505	0.01758	0.03093	0.01764
3.35	0.02715	0.02195	0.03403	0.04528	0.03383
3.55	0.02154	0.03160	0.00926	0.02975	0.00933
3.75	0.03466	0.02256	0.01749	0.03942	0.01751
3.95	0.01461	0.02463	0.01757	0.0270	0.01768
Average	0.0339315	0.03856	0.0417	0.0403	0.0399

Table 7.4: Relative error for the ground state energy - LiH molecule

8
Conclusion

VQAs were presented as the answer to the current hardware restrictions of quantum computers during the NISQ Era. With that said, this area of research currently is currently one of the most significant and prominent topics in QC. They are being experimented for several applications, such as Computational Chemistry (as seen with the VQE), Mathematical Programming [41; 42], Optimization [43; 44], Quantum Information [45] and Machine Learning [46].

Moreover, new iterations for the VQE, as mentioned in Chapter 5, have been presented, such as the ADAPT-VQE [32] and TETRIS-ADAPT-VQE [33]. In brief, they try to reduce the depth of a problem-inspired ansatz, filtering excitations that do not contribute significantly to the final result. This ends up reducing the number of required qubits, making it feasible to run probem-inspired ansatz in NISQ hardware.

Beyond the scope of variational models, QC has been a thriving field of research. Whilst this thesis was being written over the past six months, for example, some milestones have been achieved.

For instance, IBM released a 400-qubit quantum computer[1], which is already available on IBMQ (not for open access). It was a big step, considering that IBM's biggest device had 127 qubits. Additionally, IBM has set the bold goal of unveiling, by 2033, a 100000-qubit system, where they will pair quantum computers with supercomputers[2].

Furthermore, Topological QC has presented important breakthroughs, where two *'firsts'* demonstrations of Non-Abelian braiding were realized by Quantinuum [47] and Google [48]. Such achievement is a big step towards the fault-tolerant QC era, as topological qubits are more resilent to noise, although still requiring Quantum Error Correction.

Overall, Quantum Computing is probably more than a decade away from reaching fault-tolerant quantum computers that are large enough to perform more costly algorithms such as Shor's Integer Factorization [5] and Quantum Phase Estimation (QPE) [9]. However, Variational Quantum Algorithms are presented as a good medium-term alternative, that could make quantum computers able to tackle some real-world problems and reach some computational speedup in the meantime.

[1] IBM Newsroom
[2] IBM Research Blog

Bibliography

[1] DESHPANDE, A.. **Assessing the quantum-computing landscape**, Sept. 2022.

[2] ARUTE, F.; ET AL. **Quantum supremacy using a programmable superconducting processor**, 2019.

[3] MADSEN, L. S.; ET AL. **Quantum computational advantage with a programmable photonic processor**, 2022.

[4] PRESKILL, J.. **Quantum computing in the NISQ era and beyond**. Quantum, 2:79, aug 2018.

[5] SHOR, P. W.. **Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer**, oct 1997.

[6] GROVER, L. K.. **A fast quantum mechanical algorithm for database search**, 1996.

[7] 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.. **Variational Quantum Algorithms**, Sept. 2021. arXiv:2012.09265 [quant-ph, stat].

[8] PERUZZO, A.; MCCLEAN, J.; SHADBOLT, P.; YUNG, M.-H.; ZHOU, X.-Q.; LOVE, P. J.; ASPURU-GUZIK, A. ; O'BRIEN, J. L.. **A variational eigenvalue solver on a photonic quantum processor**, July 2014. Number: 1 Publisher: Nature Publishing Group.

[9] NIELSEN, M. A.; CHUANG, I. L.. **Quantum computation and quantum information: 10th anniversary edition**, 2010.

[10] GRIFFITHS, D. J.; SCHROETER, D. F.. **Introduction to quantum mechanics**, 2018.

[11] FEYNMAN, R. P.. **Simulating physics with computers**, June 1982.

[12] DIVINCENZO, D. P.. **The Physical Implementation of Quantum Computation**, 2000.

[13] RIPPER, P.; AMARAL, G. ; TEMPORÃO, G.. **Swap test-based characterization of decoherence in universal quantum computers**. Quantum Information Processing, 22(5), May 2023.

[14] MURALI, P.; MCKAY, D. C.; MARTONOSI, M. ; JAVADI-ABHARI, A.. **Software mitigation of crosstalk on noisy intermediate-scale quantum computers**. In: PROCEEDINGS OF THE TWENTY-FIFTH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, ASPLOS '20, p. 1001–1016, New York, NY, USA, 2020. Association for Computing Machinery.

[15] NANNICINI, G.; BISHOP, L. S.; GÜNLÜK, O. ; JURCEVIC, P.. **Optimal qubit assignment and routing via integer programming**. ACM Transactions on Quantum Computing, 4(1), oct 2022.

[16] BHARTI, K.; CERVERA-LIERTA, A.; KYAW, T. H.; HAUG, T.; ALPERIN-LEA, S.; ANAND, A.; DEGROOTE, M.; HEIMONEN, H.; KOTTMANN, J. S.; MENKE, T.; MOK, W.-K.; SIM, S.; KWEK, L.-C. ; ASPURU-GUZIK, A.. **Noisy intermediate-scale quantum algorithms**. Reviews of Modern Physics, 94(1), feb 2022.

[17] SIPSER, M.. **Introduction to the Theory of Computation**. Course Technology, Boston, MA, third edition, 2013.

[18] AARONSON, S.. **Quantum Computing since Democritus**. Cambridge University Press, USA, 2013.

[19] CROSS, A. W.; BISHOP, L. S.; SHELDON, S.; NATION, P. D. ; GAMBETTA, J. M.. **Validating quantum computers using randomized model circuits**. Phys. Rev. A, 100:032328, Sep 2019.

[20] TREINISH, M.; GAMBETTA, J.; NATION, P.; QISKIT BOT; KASSEBAUM, P.; RODRÍGUEZ, D. M.; DE LA PUENTE GONZÁLEZ, S.; LISHMAN, J.; HU, S.; KRSULICH, K.; GARRISON, J.; BELLO, L.; YU, J.; MARQUES, M.; GACON, J.; MCKAY, D.; GOMEZ, J.; CAPELLUTO, L.; TRAVIS-S-IBM;

MITCHELL, A.; PANIGRAHI, A.; LERONGIL; RAHMAN, R. I.; WOOD, S.; ITOKO, T.; POZAS-KERSTJENS, A.; WOOD, C. J.; SINGH, D.; RISINGER, D. ; ARBEL, E.. **Qiskit/qiskit: Qiskit 0.39.4**, Dec. 2022.

[21] DEVELOPERS, C.. **Cirq**, Dec. 2022. See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors.

[22] LUO, X.-Z.; LIU, J.-G.; PAN, Z. ; WANG, L.. **Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design**, Oct. 2020.

[23] XAVIER, P. M.; RIPPER, P.; ANDRADE, T.; GARCIA, J. D. ; NEIRA, D. E. B.. **ToQUBO.jl**, feb 2023.

[24] XAVIER, P. M.; RIPPER, P.; ANDRADE, T.; GARCIA, J. D. ; NEIRA, D. E. B.. **psrenergy/qubodrivers.jl: v0.2.0**, Apr. 2023.

[25] XAVIER, P. M.; RIPPER, P.; ANDRADE, T.; GARCIA, J. D. ; NEIRA, D. E. B.. **psrenergy/qubotools.jl: v0.8.0**, Apr. 2023.

[26] SIM, S.; JOHNSON, P. D. ; ASPURU-GUZIK, A.. **Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms**, oct 2019.

[27] HOLMES, Z.; SHARMA, K.; CEREZO, M. ; COLES, P. J.. **Connecting ansatz expressibility to gradient magnitudes and barren plateaus**, Jan 2022.

[28] CAO, Y.; ROMERO, J.; OLSON, J. P.; DEGROOTE, M.; JOHNSON, P. D.; KIEFEROVÁ, M.; KIVLICHAN, I. D.; MENKE, T.; PEROPADRE, B.; SAWAYA, N. P. D.; SIM, S.; VEIS, L. ; ASPURU-GUZIK, A.. **Quantum Chemistry in the Age of Quantum Computing**. Chemical Reviews, Aug. 2019. Publisher: American Chemical Society.

[29] SPALL, J. C.. **An overview of the simultaneous perturbation method for efficient optimization**, 1998.

[30] HEMPEL, C.; MAIER, C.; ROMERO, J.; MCCLEAN, J.; MONZ, T.; SHEN, H.; JURCEVIC, P.; LANYON, B. P.; LOVE, P.; BABBUSH, R.; ASPURU-GUZIK, A.; BLATT, R. ; ROOS, C. F.. **Quantum chemistry calculations**

on a trapped-ion quantum simulator. Physical Review X, 8(3), jul 2018.

[31] O'MALLEY, P. J. J.; BABBUSH, R.; KIVLICHAN, I. D.; ROMERO, J.; MCCLEAN, J. R.; BARENDS, R.; KELLY, J.; ROUSHAN, P.; TRANTER, A.; DING, N.; CAMPBELL, B.; CHEN, Y.; CHEN, Z.; CHIARO, B.; DUNSWORTH, A.; FOWLER, A. G.; JEFFREY, E.; LUCERO, E.; MEGRANT, A.; MUTUS, J. Y.; NEELEY, M.; NEILL, C.; QUINTANA, C.; SANK, D.; VAINSENCHER, A.; WENNER, J.; WHITE, T. C.; COVENEY, P. V.; LOVE, P. J.; NEVEN, H.; ASPURU-GUZIK, A. ; MARTINIS, J. M.. **Scalable quantum simulation of molecular energies**. Phys. Rev. X, 6:031007, Jul 2016.

[32] GRIMSLEY, H. R.; ECONOMOU, S. E.; BARNES, E. ; MAYHALL, N. J.. **An adaptive variational algorithm for exact molecular simulations on a quantum computer**. Nature Communications, 10(1), jul 2019.

[33] ANASTASIOU, P. G.; CHEN, Y.; MAYHALL, N. J.; BARNES, E. ; ECONOMOU, S. E.. **Tetris-adapt-vqe: An adaptive algorithm that yields shallower, denser circuit ansätze**, 2022.

[34] ZHANG, Y.; CINCIO, L.; NEGRE, C. F. A.; CZARNIK, P.; COLES, P. J.; ANISIMOV, P. M.; MNISZEWSKI, S. M.; TRETIAK, S. ; DUB, P. A.. **Variational quantum eigensolver with reduced circuit complexity**. npj Quantum Information, 8(1), aug 2022.

[35] HELGAKER, T.; JØRGENSEN, P. ; OLSEN, J.. **Molecular electronic structure theory**, 2000.

[36] SHEE, Y.; TSAI, P.-K.; HONG, C.-L.; CHENG, H.-C. ; GOAN, H.-S.. **Qubit-efficient encoding scheme for quantum simulations of electronic structure**, May 2022.

[37] JORDAN, P.; WIGNER, E.. **Über das paulische äquivalenzverbot**, 1928.

[38] SEELEY, J. T.; RICHARD, M. J. ; LOVE, P. J.. **The Bravyi-Kitaev transformation for quantum computation of electronic structure**, 12 2012. 224109.

[39] ALEXANDER, T.; KANAZAWA, N.; EGGER, D. J.; CAPELLUTO, L.; WOOD, C. J.; JAVADI-ABHARI, A. ; MCKAY, D. C.. **Qiskit pulse: programming quantum computers through the cloud with pulses**, aug 2020.

[40] SUN, Q.; BERKELBACH, T. C.; BLUNT, N. S.; BOOTH, G. H.; GUO, S.; LI, Z.; LIU, J.; MCCLAIN, J. D.; SAYFUTYAROVA, E. R.; SHARMA, S.; WOUTERS, S. ; CHAN, G. K.-L.. **Pyscf: the python-based simulations of chemistry framework**. WIREs Computational Molecular Science, 8(1):e1340, 2018.

[41] JENNINGS, D.; LOSTAGLIO, M.; PALLISTER, S.; SORNBORGER, A. T. ; SUBAŞI, Y.. **Efficient quantum linear solver algorithm with detailed running costs**, 2023.

[42] BRAVO-PRIETO, C.; LAROSE, R.; CEREZO, M.; SUBASI, Y.; CINCIO, L. ; COLES, P. J.. **Variational quantum linear solver**, 2020.

[43] SÆVARSSON, B.; CHATZIVASILEIADIS, S.; JÓHANNSSON, H. ; ØSTERGAARD, J.. **Quantum computing for power flow algorithms: Testing on real quantum computers**, 2022.

[44] FARHI, E.; GOLDSTONE, J. ; GUTMANN, S.. **A quantum approximate optimization algorithm**, 2014.

[45] ABBAS, A.; KING, R.; HUANG, H.-Y.; HUGGINS, W. J.; MOVASSAGH, R.; GILBOA, D. ; MCCLEAN, J. R.. **On quantum backpropagation, information reuse, and cheating measurement collapse**, 2023.

[46] SCHULD, M.; BOCHAROV, A.; SVORE, K. M. ; WIEBE, N.. **Circuit-centric quantum classifiers**, mar 2020.

[47] IQBAL, M.; TANTIVASADAKARN, N.; VERRESEN, R.; CAMPBELL, S. L.; DREILING, J. M.; FIGGATT, C.; GAEBLER, J. P.; JOHANSEN, J.; MILLS,

M.; MOSES, S. A.; PINO, J. M.; RANSFORD, A.; ROWE, M.; SIEGFRIED, P.; STUTZ, R. P.; FOSS-FEIG, M.; VISHWANATH, A. ; DREYER, H.. **Creation of non-abelian topological order and anyons on a trapped-ion processor**, 2023.

[48] ANDERSEN, T. I.; LENSKY, Y. D.; KECHEDZHI, K.; DROZDOV, I. K.; BENGTSSON, A.; HONG, S.; MORVAN, A.; MI, X.; OPREMCAK, A.; ACHARYA, R.; ALLEN, R.; ANSMANN, M.; ARUTE, F.; ARYA, K.; ASFAW, A.; ATALAYA, J.; BABBUSH, R.; BACON, D.; AI, G. Q. ; COLLABORATORS. **Non-abelian braiding of graph vertices in a superconducting processor**. Nature, 2023.