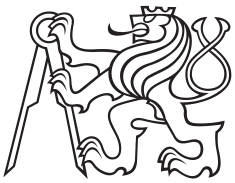
Bakalářská práce



České
vysoké
učení technické
v Praze

F4

Fakulta jaderná a fyzikálně inženýrská
Katedra fyziky

Programování kvantového počítače

Vlastimil Hudeček

Vedoucí: Aurél Gábor Gábris, Ph.D.
Obor: Matematické inženýrství, zaměření Matematická fyzika
Studijní program: Aplikace přírodních věd
Prosinec 2022

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA JADERNÁ A FYZIKÁLNĚ INŽENÝRSKÁ
PRAHA 1 - STARÉ MĚSTO, BŘEHOVÁ 7 - PSČ 115 19

Katedra: fyziky *Akademický rok:* 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Vlastimil Hudeček

Studijní program: Aplikace přírodních věd

Obor: Matematické inženýrství, zaměření Matematická fyzika

Název práce: (česky) Programování kvantového počítače

Název práce: (anglicky) Programming quantum computers

Pokyny pro vypracování:

1) Studovat kvantové optimalizační algoritmy: variační kvantový algoritmus hledání vlastních čísel (VQE) a kvantový aproximační optimalizační algoritmus (QAOA)

2) Seznámit se s implementacemi vybraných kvantových optimalizačních algoritmů v kvantovém SDK frameworku jako například qiskit nebo cirq

3) Najít fyzikálně relevantní optimalizační problém vhodný pro efektivní implementaci na kvantovém počítači

4) Implementovat kvantový počítačový kód k řešení kvantové optimalizační metody

5) Studovat vlastnosti jako jsou efektivita a stabilita implementovaných řešení

Doporučená literatura:

[1] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, 10th Anniversary Edition, Cambridge Univ. Press, 2010

[2] E. Farhi, J. Goldstone, Sa. Gutmann: A Quantum Approximate Optimization Algorithm. arXiv:1411.4028

[3] A. Peruzzo, J. McClean, P. Shadbolt, M.-H.Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien: A variational eigenvalue solver on a photonic quantum processor. Nature Commun. 5, 4213 (2014)

[4] J. R. McClean, Ma. P. Harrigan, M. Mohseni, N. C. Rubin, Zh. Jiang, S. Boixo, V. N. Smelyanskiy, R. Babbush, H. Neven: Low-Depth Mechanisms for Quantum Optimization. PRX Quantum 2, 030312 (2021)

Jméno a pracoviště vedoucího bakalářské práce:

Aurél Gábor Gábris, Ph.D.
Katedra fyziky, Fakulta jaderná a fyzikálně inženýrská ČVUT v Praze

Datum zadání bakalářské práce: 20.10.2021

Termín odevzdání bakalářské práce: 07.07.2022

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

.....
děkan

V Praze dne 20.10.2021

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská

Břehová 7
115 19 Praha 1

PROHLÁŠENÍ

Já, níže podepsaný

Jméno a příjmení studenta:	Vlastimil Hudeček
Osobní číslo:	494840
Název studijního programu (oboru):	Matematické inženýrství: Matematická fyzika

prohlašuji, že jsem bakalářskou práci s názvem:

Programování kvantového počítače (Programming quantum computers)

vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 30.12.2022 ..... Hudeček .....

podpis

Poděkování

Předně chci poděkovat Aurélu Gáboru Gábrisovi za veškerou odbornou pomoc při psaní této práce. Vždy se mi trpělivě věnoval, pomáhal mi svými radami a inspiroval mě k dalšímu prohlubování znalostí. Děkuji ČVUT, za veškerou poskytnutou podporu v průběhu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 22. prosince 2022

Abstrakt

Díky rychlému vývoji kvantových počítačů v nedávné době, narůstá zájem o kvantové algoritmy a jejich využití. V prvních kapitolách této práce je podán stručný úvod do tématu kvantové výpočetní techniky a algoritmů. Dále jsou v nich shrnuty principy, vlastnosti a použití dvou významných kvantových algoritmů, variační algoritmus hledání vlastních čísel (VQE) a kvantový aproximační optimalizační algoritmus (QAOA). V dalších kapitolách jsou nastíněny v současné době dostupné softwarové nástroje pro vývoj kvantových programů, důraz je zde kladen na SDK Qiskit od IBM. Poslední část práce se věnuje implementaci zkoumaných algoritmů.

Klíčová slova:. Kvantové počítání, kvantový algoritmus, QAOA, VQE, Qiskit

Vedoucí:. Aurél Gábor Gábris, Ph.D.
Katedra fyziky,
Fakulta jaderná a fyzikálně inženýrská
ČVUT v Praze

Abstract

Thanks to the rapid development of quantum computers in recent years, the interest in quantum algorithms and their applications is rising. The first chapters of this work gives a brief introduction to the topic of quantum computing and algorithms. Additionally, they summarize the principles, properties and applications of two major quantum algorithms, the variational quantum eigensolver and the quantum approximate optimization algorithm. The following chapters outline the currently available software tools for development of quantum programs with focus on the IBM Qiskit software development kit. Final part of work deals with implementation of the studied algorithms.

Keywords:. Quantum computing, quantum algorithm, QAOA, VQE, Qiskit

Title translation:. Programming quantum computers

Contents

Chapter 1

Introduction

At the end of 1970s the idea to use the extraordinary properties of quantum systems in computation was first proposed and soon led to development of theoretical foundations to study its potential. At the beginning of 1980s the concept was introduced to broader public by Richard Feynman[1]. The interest in the quantum effect used for the computational purposes was renewed when in the 1980s researches, such as David Deutsch, explored the theory of quantum information and quantum information processing in particular. Finally, in the 1990s emerged the first truly quantum algorithms which were proven to be superior over classical alternatives. The theoretical advantages of the new approach to computation motivated further research and in 1994 Peter Shor presented polynomial-time quantum algorithm for factoring integers. New improvements in this field of study were made throughout the rest of twentieth century. The new quantum algorithms introduced more efficient solutions not only to problems such as factorization, but also to search problems and quantum simulations. Promising results of research motivated the creation of first quantum computers. At the current moment such devices appear to have significant advantages over classical computers. Further progress depends not only on development of more advanced hardware but also on progress in quantum algorithms.[2, 3]

In this project I would like to give an introduction to the topic of quantum algorithms, and to summarize the key terms and concepts. The next goal is to present two important quantum algorithms, the Quantum approximate optimization algorithm and the Variational quantum eigensolver. The first one being concerned with solving combinatorial problems and the latter with finding eigenvalues. These algorithms are in the centre of interest for this project, because they are examples of significant building blocks in terms of algorithms that can be used to create more complex programs. Both algorithms are subject of current research as they were discovered in recent past. This project will also give on overview of currently publicly available quantum SDK frameworks as well as demonstrate some of their possibilities of simulating of discussed quantum circuits.

The main goal of project is to give detail and rigorous description of discussed topics that can be used for further research. Apart from description of algorithms the project aims to discuss advantages and pitfalls of quantum algorithms mentioned above.

1.1 Basic concepts

In order to discuss the quantum algorithms let us summarize the basic terminology first. The concepts explained below are necessary to be able to describe the algorithms that are studied in Chapters 2 and 3. It should be emphasized that the general formalism of quantum computing, which is used to work with the abstract qubits, is theoretically independent of the physical qubit realization. This is true as long as the realization is ideal. In real conditions the abstract models need to be adjusted in order to account for the properties of physical object.

1.1.1 Qubits and qubit registers

The fundamental concept in theory of quantum information is a qubit, which is a quantum unit of information, that unlike a classical bit can be in a linear combination of states. A qubit can be understood as a vector in Hilbert space $\mathcal{H}$. If we define an orthogonal computational basis $|0\rangle, |1\rangle$ of $\mathcal{H}$, then we can describe the state of qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ for $\alpha, \beta \in \mathbb{C}$. The meaning of α and β can be seen from the Born rule, which states that the probability of measuring a certain result, e.g. 0, is equal to $|\alpha|^2 = |\langle 0|\psi\rangle|^2$. This implies $|\alpha|^2 + |\beta|^2 = 1$. Another important property of qubit is the phase. The global phase can be seen, when we express the qubit state as $|\psi\rangle = e^{i\gamma}\left(\cos\frac{\phi}{2}|0\rangle + e^{i\varphi}\sin\frac{\phi}{2}|0\rangle\right)$.

The register refers to an array of multiple qubits that are being worked with during computations. It is important to note that in order to describe the state of a system composed of multiple qubits the tensor product operation is applied. The state of the composed system is an element of the tensor product of respective Hilbert spaces of qubits that are included. In this project we use the standard notation $|0\rangle \otimes |1\rangle = |01\rangle$. If we describe qubits in matrix notation we can write

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, |0\rangle \otimes |1\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1\cdot 0\\1\cdot 1\\0\cdot 0\\0\cdot 1\end{pmatrix} = |01\rangle . \qquad \text{Eq (1.1.1)}$$

Once we have described a qubit register we can discuss the convention that is often used to name and describe states of qubit register. Let $n \in \mathbb{N}$ and $N = 2^n \in \mathbb{N}$ be the number of basis states. Then the x-th state, where $x \in 0, \ldots, 2^n - 1$, can be naturally identified with the state $|x_1 x_2 \ldots x_n\rangle$, where $x = \sum_{i=1}^{n} x_i 2^{n-i}$.[4]

1.1.2 Quantum circuits and gates

In order to describe manipulations with classical bits on the level of individual Boolean operations the logic circuit formalism can be used. In quantum computing there is the same necessity to describe manipulation with qubits. A frequent solution of this necessity is the use of so-called quantum circuits. The quantum circuit notation is similar to classic circuits, because utilizes the concepts of wires that

carry units of information and gates that have various effects on these units. In this project we will use the circuit notation to keep the schematics compatible with the description used by the quantum SDKs.

The qubits can be understood as vectors on $\mathcal{H}$ and the quantum gates represent the allowed transformations that are described by unitary operators. Some of the most important gates, including both single qubit and multiqubit gates, in standard basis are listed below.

Single qubit gates

First we shall list the identity and Pauli matrices[1] and their effect of qubit in standard basis. Let $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad I\,|\psi\rangle = a\,|0\rangle + b\,|1\rangle \qquad \text{Eq (1.1.2)}$$

$$X = \sigma_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_X\,|\psi\rangle = b\,|0\rangle + a\,|1\rangle \qquad \text{Eq (1.1.3)}$$

$$Y = \sigma_Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix} \qquad \sigma_Y\,|\psi\rangle = -\mathrm{i}b\,|0\rangle + \mathrm{i}a\,|1\rangle \qquad \text{Eq (1.1.4)}$$

$$Z = \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \sigma_Z\,|\psi\rangle = a\,|0\rangle - b\,|1\rangle \qquad \text{Eq (1.1.5)}$$

These operators are sometimes marked $\sigma_0, \sigma_1, \sigma_2, \sigma_3$, because together the sigma operators form an basis of 2×2 Hermitean matrices. The Pauli gates are also linked to the following useful identities for outer product[5].

$$|0\rangle\langle 0| = \frac{1}{2}(I + Z) \qquad I = |0\rangle\langle 0| + |1\rangle\langle 1| \qquad \text{Eq (1.1.6)}$$

$$|0\rangle\langle 1| = \frac{1}{2}(X + \mathrm{i}Y) \qquad Z = |0\rangle\langle 0| - |1\rangle\langle 1| \qquad \text{Eq (1.1.7)}$$

$$|1\rangle\langle 0| = \frac{1}{2}(X - \mathrm{i}Y) \qquad Y = \mathrm{i}(|1\rangle\langle 0| - |0\rangle\langle 1|) \qquad \text{Eq (1.1.8)}$$

$$|1\rangle\langle 1| = \frac{1}{2}(I - Z) \qquad X = |0\rangle\langle 1| + |1\rangle\langle 0| \qquad \text{Eq (1.1.9)}$$

[1] The Pauli matrices bear the name of Wolfgang Pauli, Austrian physicist and a pioneer of quantum physics.

Next we shall list other important single qubit gates with their names leaving out the effect for the sake of brevity.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad \text{Hadamard gate} \qquad \text{Eq (1.1.10)}$$

$$S = \begin{pmatrix}1 & 0\\ 0 & \mathrm{i}\end{pmatrix} \qquad \text{Phase gate} \qquad \text{Eq (1.1.11)}$$

$$T = \begin{pmatrix}1 & 0\\ 0 & \exp\left\{\mathrm{i}\frac{\pi}{4}\right\}\end{pmatrix} \qquad \pi/8 \text{ gate} \qquad \text{Eq (1.1.12)}$$

$$R_x = \begin{pmatrix}\cos\frac{\phi}{2} & -\mathrm{i}\sin\frac{\phi}{2}\\ -\mathrm{i}\sin\frac{\phi}{2} & \cos\frac{\phi}{2}\end{pmatrix} \qquad \text{Rotation about } \hat{x} \text{ axis} \qquad \text{Eq (1.1.13)}$$

$$R_y = \begin{pmatrix}\cos\frac{\phi}{2} & -\sin\frac{\phi}{2}\\ \sin\frac{\phi}{2} & \cos\frac{\phi}{2}\end{pmatrix} \qquad \text{Rotation about } \hat{y} \text{ axis} \qquad \text{Eq (1.1.14)}$$

$$R_z = \begin{pmatrix}\exp\left(-\mathrm{i}\frac{\phi}{2}\right) & 0\\ 0 & \exp\left(\mathrm{i}\frac{\phi}{2}\right)\end{pmatrix} \qquad \text{Rotation about } \hat{z} \text{ axis} \qquad \text{Eq (1.1.15)}$$

The Hadamard gates have special importance, because they are able to change states $|0\rangle$ and $|1\rangle$ to so-called balanced states, i.e. states for which there is an equal probability of measuring 1 or 0 in standard basis.

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \qquad \text{Eq (1.1.16)}$$

The operation of applying Hadamard transformation to multiple qubits of a quantum register, equivalent to a tensor product of the necessary number of H gates, is called *Hadamard transformation*, marked $\mathcal{H} = H^{\otimes n}$.

Multiqubit gates

Most common example of multiqubit gates are the controlled unitary gates. For example a controlled X, which is usually denoted as CX, gate works similarly to a classical controlled `NOT` logical gate, i.e. `CNOT`, if we consider only $|0\rangle$ and $|1\rangle$ states. This means that X gate is applied to target qubit only if the control qubit is in $|1\rangle$ state. This implies how the matrix representation of such gate looks. Other

$$CX = \texttt{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix} =$$

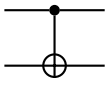


Table 1.1: The CX or `CNOT` gate matrix and the circuit symbol

controlled gates represented by Pauli matrices have similar form. It can be proved that an controlled arbitrary single qubit gate with a unitary matrix form can be created only using two `CNOT` gates, three single qubit gates and a phase adjustment. The next multiqubit gate we mention is the Toffoli

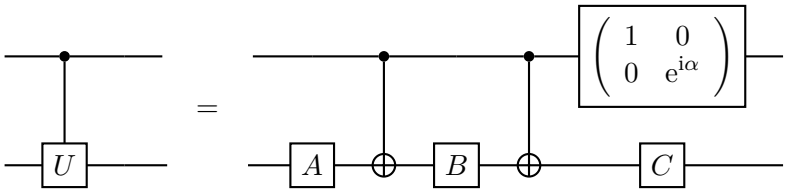


Figure 1.1: Circuit realizing controlled U gate using unitary and `CNOT` gates that satisfy $ABC = I$ and $U = \exp(\mathrm{i}\alpha)AXBXC$.

$$\begin{pmatrix} I_{6,6} & \vec{0} & \vec{0} \\ \vec{0}^T & 0 & 1 \\ \vec{0}^T & 1 & 0 \end{pmatrix} =$$

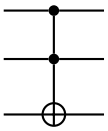


Table 1.2: The `CNOT` gate matrix and the circuit symbol

gate, a triple qubit gate. Its function can be simply described as an X controlled by two qubits.[4][2] Its importance lies in it being the foundation for reversible computation on quantum computer with classical Boolean logic. Hence, it and other equivalent gates are needed to prove theorems about general properties of quantum computers in comparison with classical computers.

Gates based on matrix functions

Many quantum algorithms use gates which are based of functions with matrix argument. More generally the *operator functions* are considered.

Definition 1.1 (Operator function)**.** Consider a normal operator (resp. matrix) A, which has a spectral decomposition $A = \sum_a a\,|a\rangle\langle a|$, then we define an *operator (resp. matrix) function* f its action

$$f(A) := \sum_a f(a)\,|a\rangle\langle a|\,. \qquad \text{Eq (1.1.17)}$$

In many practical situations an approximation of the function by a Taylor series is used, where the argument of function appears as a power of the argument to a positive integer, which can be applied to a matrix easily.

Example 1.2 (Basic operator functions)**.** In order to explain the term of operator functions, few examples are listed below.

- The first example could be defined even without the formalism of operator functions. Taking $f(x) := \sqrt{x}$ and arbitrary normal A, then the matrix $A^{1/2}$ can be computed for a small matrix using the fact that $A^{1/2} \cdot A^{1/2} = A$.
- Consider $f(x) := \mathrm{e}^x$ and $A := \theta \cdot Z$. It can be seen that $Z = (+1) \cdot |0\rangle\langle 0| + (-1)\,|1\rangle\langle 1|$. Thus

$$f(\theta Z) = \mathrm{e}^{\theta}\,|0\rangle\langle 0| + \mathrm{e}^{-\theta}\,|1\rangle\langle 1| = \begin{pmatrix} \mathrm{e}^{\theta} & 0 \\ 0 & \mathrm{e}^{-\theta} \end{pmatrix} \qquad \text{Eq (1.1.18)}$$

- Through the Euler identity it can be seen that for $f(x) := \mathrm{e}^x$ and $A := \mathrm{i}\theta\vec{\mathbf{v}} \cdot \vec{\boldsymbol{\sigma}}$, where $\vec{\mathbf{v}} \in \mathbb{R}^3$ and $\vec{\mathbf{v}} \cdot \vec{\boldsymbol{\sigma}} := \sum_{i=1}^{3} v_i \cdot \sigma_i$ we have

$$f(\mathrm{i}\theta\vec{\mathbf{v}} \cdot \vec{\boldsymbol{\sigma}}) = \cos(\theta)I + \mathrm{i}\sin(\theta)\,(\vec{\mathbf{v}} \cdot \vec{\boldsymbol{\sigma}}) \qquad \text{Eq (1.1.19)}$$

- The last identity can be further generalized, $f : \mathbb{C} \to \mathbb{C}$ and $A := \theta\vec{\mathbf{n}} \cdot \vec{\boldsymbol{\sigma}}$, where $\vec{\mathbf{n}}$ is a normalized real vector:

$$f(\theta\vec{\mathbf{n}} \cdot \vec{\boldsymbol{\sigma}}) = \frac{f(\theta) + f(-\theta)}{2} I + \frac{f(\theta) - f(-\theta)}{2} \vec{\mathbf{n}} \cdot \vec{\boldsymbol{\sigma}} \qquad \text{Eq (1.1.20)}$$

Analysis of quantum circuits

To understand and analyse complicated multiqubit circuits it is useful to define several concepts. The most trivial metric of the resources needed by an algorithm is the number of physical qubits needed for a realization. In some cases it is desirable to include the number of additional qubits, i.e. the ancillary qubits defined below 1.6, if their number is significant. To quantify the number of steps needed to execute the algorithm using an abstract quantum computer capable of performing arbitrary number of parallel operations we define the circuit depth.

Definition 1.3 (Circuit depth)**.** The depth of a quantum circuit denotes the number of distinct time steps at which a parallel set of gates is applied.

Apart from the quantum specific metrics, many conventional characteristics are used as well. For example for approximative algorithms it is important to quantify the dispersion of the result.

For many algorithms an important role is played by so-called weight characteristics. Examples of widely used weights are the Hamming weight and Pauli string weight.

Definition 1.4 (Hamming distance, Hamming weight)**.** [4] Let S_1 and S_2 be a binary strings composed of ones and zeros that have the same length, i.e. they contain the same number of symbols. For these strings we define the *Hamming distance* as the number of symbols in which S_1 and S_2 differ. The Hamming distance is denoted as $\mathrm{d}_H(\bullet,\ \bullet)$. For example $S_1 = 0011, S_2 = 0110$ we have $\mathrm{d}_H(S_1, ,\ S_2) = 2$.

Using the Hamming distance the *Hamming weight* for a string S can be defined as $\mathrm{d}_H(S,\ S_0)$, where S_0 represents a string consisting of zeros of the same length as S. The Hamming weight is equal to the number of non-zero symbols in a string.

Definition 1.5 (Pauli string weight)**.** [6] Consider a Pauli string, that is a tensor product of $n \in \mathbb{N}$ of Pauli gates. The *Pauli weight* represents the number of qubits Pauli string acts on non-trivially. Hence, it is equal to the number of X, Y, Z operators in the Pauli string. In this project the Pauli weight of a Pauli string P_a is denoted a w_a.

Ancilla qubits

Another important concept in quantum computation are the ancillary qubits. In order to perform some algorithms or even implement some gates, e.g. implement `CNOT` or a `NOT` gate using Toffoli gates, additional qubits need to be used.

Definition 1.6 (Ancilla qubit)**.** Consider a quantum circuit. A qubit in register which is initially in a known state and its value does not contain the result of the computation at the end of circuit is called an *ancillary qubit*[2]. The ancillary qubits are often not explicitly measured, i.e. their end "value" is discarded. In other cases they hold the result of a more complex operation on circuit, such as the Hadamard test[6], which is used to measure real an imaginary part of an amplitude of form $\langle\Psi|U|\Psi\rangle$.

The ancillary qubits are important parts of quantum circuits because the quantum computing is reversible, so the values of qubits cannot be set to arbitrary values without loss of information. The ancillary qubits often play role of a record of performed operations on work qubits. The number of ancillary qubits has to be taken into account of total number of used qubits in quantum circuit since the ancillary qubits have to be physically present in the register.

1.2 Quantum algorithms

The goal of an algorithm is to effectively solve the assigned problem using the given resources. The classical algorithms rely on the classical resources of information, communication and information processing. On the other hand the quantum algorithms apart from quantum counterparts of classical resources have physical resource such as superposition entanglement and simulation capabilities inaccessible to classical algorithms. These resources can be harnessed in order to create algorithms with quantum advantage[3].

Since the theoretical breakthroughs of the beginning of the 21st century the efforts to built quantum computers with hundreds or more qubits are increasing. The current quantum computers suffer many technological limitations, such as low number of available qubits and presence of noise which cannot be decreased by the means of error correction, because of the small size in terms of qubit quantity. This is the reason they are often called Noisy Intermediate-Scale Quantum (NISQ) devices[7]. Nevertheless, the NISQ devices have already proven to be capable of solving problems otherwise intractable on conventional computers[8, 9]. For example the Gaussian boson sampling computation model is a prime example of an approach which may lead to a practical advantage of physical quantum computers over the conventional devices[10]. The phenomenon of quantum devices outperforming classical counterparts is usually referred to as quantum supremacy[11]. These restrictions of NISQ devices mean additional requirements the algorithms have to meet. The algorithms should require minimal number of physical qubits, use the facilities of hybrid computers, i.e. perform parts of computation using classical computer, and be reasonably resistant to noise. The last feature is often called noise resilience[6].

[2] Sometimes referred to as *ancilla* for short.

[3] Some literature the "quantum advantage" and "quantum supremacy" are treated as synonyms. In this project the "quantum advantage" will be considered as a reference to an instance of "quantum supremacy" and a property of having an advantage over classical alternative in a tangible application. This approach is also used in [6]

As was stated in the introduction 1, the algorithms this works focuses on are QAOA and the VQE, both of these algorithms are considered to be low level optimization algorithms, which can be realized on NISQ devices and possibly have potential in reaching quantum supremacy in practically demanded problem. From the quantum technological point of view QAOA and more prominently VQE are variational quantum algorithms.

Definition 1.7 (Variational Quantum Algorithms)**.** The term *Variational Quantum Algorithms* (abbreviation *VQA*) refers to a group of hybrid quantum algorithms, which share a common structure. The problem is encoded into a parametrized cost function 2.2. This function defines a hypersurface, a cost landscape. The ansatz denotes the choice of cost function encoding and the choice of parameters. The solving procedure is based of a loop consisting of evaluation of the cost function by means of a quantum computer and the following optimization of the parameters using a conventional computer. The process of optimization is sometimes called training in this context. The variational principles are applied to prove the success of training.

Other common features of VQAs aim to use only low depth circuits and rely on precision of optimization methods rather than quantum error correction and fault-tolerance. Both features arise from demands of current NISQ devices.[12].

Chapter 2

Quantum approximate optimization algorithm

The combinatorial optimization problems in general are concerned with finding the maximum or the minimum of an objective function. Unfortunately in many cases, such as NP optimization problems, it may be difficult to compute the precise solution in a straightforward way. The common approach in such cases is to use an approximative algorithm that is generally capable of finding an approximate solution in polynomial time. To quantify the performance of an approximation algorithm an approximation ratio is defined.

Definition 2.1 (Approximation ratio)**.** [13] Let $A(f)$ denote value of objective function f at x as calculated by the algorithm and $\texttt{OPT}(f)$ denote the optimal value of the objective function for f. Then we define an *approximation ratio* R by formula

$$R := \frac{A(f)}{\texttt{OPT(f)}}. \qquad \text{Eq (2.0.1)}$$

This definition means that for a maximization problem we get $0 \leq R \leq 1$, where $R = 1$ is equivalent to the best possible result. It is worth nothing that this definition is significantly different from that used in classical discrete optimization and approximation methods. The approximation ratio in classical optimization is defined so that the success occurs for $R = 0$ and $R = 1$ is a failure[14] or as an upper bound for performance ratio[15].

The development of quantum computation theory motivated research of new quantum approaches to optimization problems that could offer performance improvements. Some of the most recent quantum algorithms in this field are the *Quantum Annealing* (QA) and *Quantum Approximate Optimization Algorithm* (QAOA). The latter is often seen as a promising algorithm that does not suffer with performance problems of QA and has a simple structure that makes it a candidate for implementation on the NISQ devices[13].

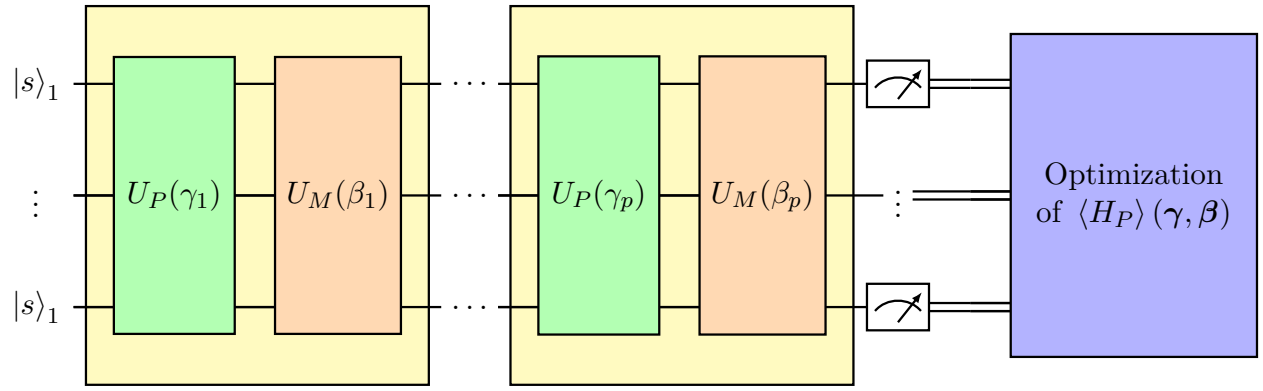


Figure 2.1: Schematic of a QAOA circuit, [16]

2.1 General QAOA algorithm

First let us define the problem to be solved. A combinatorial problem is specified by number of answer bits n and the number of clauses m. The clauses can individually be satisfied for certain assignments and be unsatisfied for others. This motivates the following definition of an objective function.

Definition 2.2 (Objective function)**.** The term *objective functions* denotes a map, which represents an optimization problem, by mapping the values of one or more variables onto a real number. The process finding the solution of the optimization problem is then reduced to finding a set of values which minimize or maximize the objective function. An objective function in case of minimization is called *cost function*[1]. For our case let $z = z_1 z_2 \dots z_n$ represent answer string and function $\forall \alpha \in 1, \dots m, C_\alpha$ represent the clause, where

$$C_\alpha(z) = \begin{cases} 1 \text{ if satisfied,} \\ 0 \text{ otherwise.} \end{cases}$$

Then we define the *objective function*

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z). \qquad \text{Eq (2.1.1)}$$

To implement the objective function using quantum formalism we first consider the Hilbert space. In our case we have 2^n dimensional $\mathcal{H}$ with computational basis vectors $|z\rangle$ that represent answer string. To encode the objective function we define the phase Hamiltonian H_P which acts diagonally on basis states $|z\rangle$

Definition 2.3 (Phase Hamiltonian, phase operator)**.** Let $|z\rangle$ denote a computational basis state. Then we define *phase Hamiltonian* H_P using computational basis as

$$H_P |z\rangle := C(z) |z\rangle , \qquad \text{Eq (2.1.2)}$$

where for $C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$ we have

$$C_\alpha |z\rangle = C_\alpha |z\rangle \qquad \text{Eq (2.1.3)}$$

[1] Is it worth mentioning that the objective function for maximalization is, by analogy to *cost function* for minimization, called a *profit function*.

Using the phase Hamiltonian we define the *phase operator* U_P with parameter γ as

$$U_P(\gamma) := \mathrm{e}^{-\mathrm{i}\gamma H_P} = \exp(-\mathrm{i}\gamma H_P) = \prod_{\alpha=1}^{m} \exp(-\mathrm{i}\gamma C_\alpha). \qquad \text{Eq (2.1.4)}$$

Here we notice that the terms of product Eq (2.1.4) commute and that γ can be restricted to interval $[0, 2\pi)$. The first is consequence of the terms being diagonal in computational basis and the second is implied by the H_P having only integer eigenvalues.

The second component needed for QAOA algorithm are the mixing operators.

Definition 2.4 (Mixing Hamiltonian, mixing operators)**.** Let n be the number of work qubits, equal to length of answer string and σ_j^X denote the Pauli X operator acting on j-th qubit in register. Then we define the *mixing Hamiltonian* H_M as follows:

$$H_M := \sum_{j=1}^{n} \sigma_j^X. \qquad \text{Eq (2.1.5)}$$

Now we define the *mixing operator* U_M:

$$U_M := \mathrm{e}^{-\mathrm{i}\beta H_M} = \exp(-\mathrm{i}\beta H_M) = \prod_{j=1} \exp\left(-\mathrm{i}\beta\sigma_j^X\right), \qquad \text{Eq (2.1.6)}$$

where $\beta \in [0, \pi)$.

Note that thanks to the identity

$$\exp(\mathrm{i}Ax) = \cos(x)I + \mathrm{i}\sin(x)A, \qquad \text{Eq (2.1.7)}$$

which holds for operators satisfying $A^2 = I$, we can interpret the U_M as $U_M = \prod_{j=1} \left(R_x(2\beta)\right)_j$.

The third component is the initial state, it is defied to make the following text and circuit diagrams easier to read. To represent all possible answer strings in the initial state we use the Hadamard transformation form Section 1.1.2.

Definition 2.5 (Initial state)**.** For a quantum circuit the state, to which is the quantum register set before any computation is referred to as *initial state*. The initial states may be set differently for various algorithms.

The QAOA uses the following initial state: Let $\mathcal{H}$ be a $2^n = N$ dimensional Hilbert space with computational basis $|z\rangle$. Then we define the *initial state*[2] as

$$|s\rangle := \frac{1}{\sqrt{N}} \sum_z |z\rangle\,. \qquad \text{Eq (2.1.8)}$$

Now we may proceed to define the p-level the QAOA state.

[2]The initial state of this form is the equal superpositon state, which can be prepared by applying hadamard transformation to a register initialised to $|0\rangle^{\otimes N}$ state.

Definition 2.6 (p-level QAOA state)**.** Let p be a positive integer and $\gamma_1, \ldots, \gamma_p \in [0, 2\pi)$ and $\beta_1, \ldots, \beta_p \in [0, \pi)$ be parameters. For short $\vec{\gamma} := \left(\gamma_1, \ldots, \gamma_p\right)$ and $\vec{\beta} := \left(\beta_1, \ldots, \beta_p\right)$. Then we define p-level QAOA state as

$$\left|\vec{\gamma}, \vec{\beta}\right\rangle := U_M(\beta_p)U(\gamma_p)\ldots U(\beta_1)U_p(\gamma_1)\left|s\right\rangle \qquad \text{Eq (2.1.9)}$$

The formula Eq (2.1.9) implies that to prepare p-level QAOA in general case the depth of circuit has to be $m \cdot p + p$, where m is the depth of one U_P implementation. The next step in QAOA is to measure the expectation value for H_p

$$\langle H_p \rangle = \langle H_p \rangle (\vec{\gamma}, \vec{\beta}) := \left\langle \vec{\gamma}, \vec{\beta} \middle| H_p \middle| \vec{\gamma}, \vec{\beta} \right\rangle = \langle C \rangle \qquad \text{Eq (2.1.10)}$$

and find the maximum of expectation value $\langle H_P \rangle$ over the angles

$$M_p := \max_{\vec{\gamma} \in [0, 2\pi), \vec{\beta} \in [0, \pi)} \langle H_p \rangle (\vec{\gamma}, \vec{\beta}). \qquad \text{Eq (2.1.11)}$$

Once the optimal values of $\vec{\gamma}, \vec{\beta}$ are found, all that is left is to find the optimal answer string z^* and the value of objective function $C(z^*)$. The former can be found using with repeated measurements in computational basis and the latter can be calculated using classical computer.

The remaining problem is that it is not clear, what angles $\vec{\gamma}, \vec{\beta}$ to pick. The process of finding the optimal $\vec{\gamma}, \vec{\beta}$ for the maximum can be implemented in various ways. Let p be a fixed constant independent of n. One possible approach is to perform the algorithm with angles chosen from a set of points $G := [0, 2\pi]^p \times [0, \pi]^p$. The points of the set form a grid, which is then gradually explored by the computer. Due to the searched set G being compact and the partial derivatives of $\langle H_p \rangle$ being bound by $\mathcal{O}(m^2 + m \cdot n)$, using the big $\mathcal{O}$ notation, this procedure produces an answer string z^*, such that $C(z^*)$ is close to M_p. Another approach is to use a classical computer and a classical optimization to find the optimal $\vec{\gamma}, \vec{\beta}$. The quantum computer is in this case used only to prepare $\left|\vec{\gamma}, \vec{\beta}\right\rangle$ state and perform the measurements to get the answer string z^*[16, 13].

2.2 The properties of the general QAOA

The QAOA has distinctive properties that characterize its potential application. As can be seen from the equation Eq (2.1.11) the value of M_{p-1} can be considered a case of constrained maximization at p, thus $M_{p-1} \leq M_p$.

The QAOA may be compared another algorithm the Quantum Adiabatic Algorithm (QAA) although their principles are different. While the QAOA is an approximative algorithm that is designed to find a good approximation of the solution, the QAA is specialized in finding the optimal solution exactly, of course up to an error. This difference can be used to prove that for QAOA

$$\lim_{p \to \infty} M_p = \max_{z \in [n]} C(z). \qquad \text{Eq (2.2.1)}$$

In order to show Eq (2.2.1) the same problem is solved using QAA. First the initial state $|s\rangle = |+\cdots+\rangle$ from Eq (2.1.8) is prepared. Then the objective function is used to define the time dependent

Hamiltonian $H(t) = (1 - \frac{t}{T})H_M + (\frac{t}{T})H_P$, where the Hamiltonians H_M, H_P are not the same Hamiltonians as in previous section. This definition allows usage of QAA because $|s\rangle$ is the highest energy state of H_M and the objective is to find the highest energy state of H_P. According to its definition the H_M has only non-negative elements outside of diagonal, therefore the Perron-Frobenius theorem[17, 18] for non-negative irreducible matrices can be applied. The theorem states that there is a non-zero gap between the top and one below energy states for $t < T$. This is a sufficient condition for the QAA to succeed if the run time T is large enough. The evolution described by Hamiltonian $H(t)$ an be approximated using a modification of the Trotter formula[4]

$$\lim_{n\to\infty}(\exp\left(\mathrm{i}A\frac{t}{n}\exp\left(\mathrm{i}B\frac{t}{n}\right)\right)^n = \exp(\mathrm{i}(A+B)t), \qquad \text{Eq (2.2.2)}$$

such as

$$\exp(\mathrm{i}(A+B)\delta t) = \exp(\mathrm{i}A\delta t)\exp(\mathrm{i}B\delta t) + \mathcal{O}(\delta t^2), \qquad \text{Eq (2.2.3)}$$

where A and B are Hermitian operators. The approximation consists of altering operators $U_P(\gamma)$ and $U_M(\beta)$. To reach good approximation the parameters γ, β are chosen to be small while the p should be large to provide longer time T.

The approximation ratio can be applied to illustrate the performance of the QAOA. Consider a combinatorial optimization problem defined as maximization of an objective function. The p-level QAOA is then applied and produces optimal parameters γ^* and β^*. From the formula Eq (2.0.1) follows that

$$r := \frac{\langle C\rangle(\gamma^*,\beta^*)}{C_{\max}} = \frac{\langle\gamma^*,\beta^*|H_P|\gamma^*,\beta^*\rangle}{C_{\max}}. \qquad \text{Eq (2.2.4)}$$

In publication [16] it was shown that for 1-level QAOA $r = 0.6924 \leq R$ for MaxCut on unweighed 3-regular graphs. This result means that even for $p = 1$ QAOA has good performance.

2.3 QAOA for fixed p

To further explore the algorithm, we can focus on the case of fixed p. This assumption is reasonable as for practical implementation of the algorithm the QAOA level parameter p would be set first. In section 2.2 it was shown that the QAOA implementation strongly depends on the objective function $C(z)$ of the examined problem. From this follows that many properties of QAOA apply only for certain classes of problems and associated objective functions. One such class are the problems with objective functions that can be expressed using Boolean operators. This class of problems has a significant subclass of problems with an objective function of form

$$C(z) = a + \sum_{j=1}^{n} c_j z_j + \sum_{j<k} d_{jk} z_j z_k \quad ; c_j, d_{jk} \in \mathbb{R} \qquad \text{Eq (2.3.1)}$$

This subclass is often referred to as Quadratic unconstrained binary optimization (QUBO) class and is implementable on D-Wave quantum computers using the quantum annealing algorithm[19].

In order to describe some properties of the QAOA in more comprehensive way, the MaxCut problem, which belongs to the class of problems with Boolean objective functions, will be used as an example to illustrate the methods of deriving some properties for other problems of this class. The shown methods are generally applicable to problems with the same constitutional elements of objective functions.

2.3.1 MaxCut for graphs with bound degree

The MaxCut problem was suggested as an example of a problem that can be solved with the help of the QAOA. First let us define the MaxCut problem.

Definition 2.7 (MaxCut problem)**.** [20] Given an undirected edge-weighted graph $G = (V, E)$, where V is the set of vertices and E the set of edges, the maximum cut problem (MaxCut) is to find the bipartition of vertices, i.e. a cut dividing the set V so that the resulting sets V_1 and V_2 are complementary, that maximizes the weight of the edges crossing the partition.

This problem is a problem with many applications, some of the most prominent are circuit layout design and models in statistical physics, e.g. Ising model[21]. Finding new more efficient ways to this problem is desirable, because the MaxCut problems belongs to NP-complete problems[22].

In this work we will be concerned only with unweighted graphs, this means we are only concerned with maximizing the number of edges crossed by the cut on a graph $G = (V, E)$, where $|V| = n$ and $|E| = m$[3].

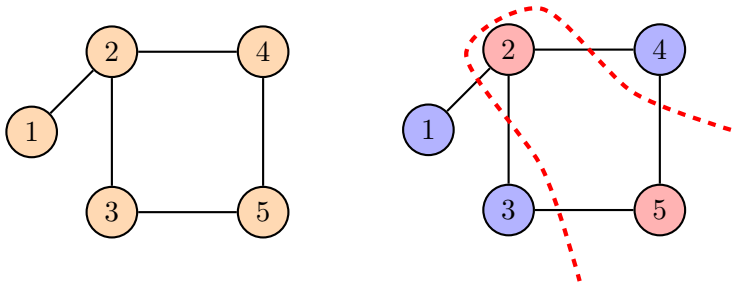


Figure 2.2: Illustration of a cut solving MaxCut problem for a simple graph

To define the configuration space of the problem we realize that the cut divides the vertices of the graph G into two disjoint subsets V_1 and V_2. The position of vertex can be marked by a binary value.

Definition 2.8 (Configuration space of MaxCut)**.** Let V_1 and V_2 be the complementary sets of vertices created by a cut. Then the *configuration space of MaxCut* is the set of n bit strings $x = x_1 x_2 \dots x_n$, where

$$x_i = \begin{cases} 1 & \text{if vertex } i \in V_1, \\ 0 & \text{if vertex } i \in V_2 \end{cases}$$

From the definition it can be seen that the requirement for the vertices of an edge to be on the opposite sides of the edge cut implies the following equation

$$\texttt{IF}(\text{Edge } \langle u, v \rangle \text{ is cut according to MaxCut definition}) = x_u \oplus x_v.$$

From this the form of objective function is deduced.

Definition 2.9 (Objective function of MaxCut)**.** Supposing the MaxCut problem is given in the form defined above. Then the objective function is defined by equation

$$C(x) = \sum_{\langle u,v \rangle} (x_u \oplus x_v), \qquad \text{Eq (2.3.2)}$$

[3] From the perpective of profit function, each edge cut gives the same profit.

where $u \in V_1$ and $v \in V_2$.

To derive phase Hamiltonian corresponding to the objective function Eq (2.3.2) the following theorem[19] is applied.[4]

Theorem 2.10 (Fourier transformation for Boolean functions). *A Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be represented by a unique Hamiltonian on n-qubits, which satisfies $H_f |x\rangle = f(x) |x\rangle$ for all computational basis states $|x\rangle$. This Hamiltonian is*

$$H_f = \sum_{S \subset [n]} \hat{f}(S) \prod_{j \in S} Z_j = \hat{f}(\emptyset) I + \sum_{j=1}^{n} \hat{f}(\{j\}) Z_j + \sum_{j<k} \hat{f}(\{j,k\}) Z_j Z_k + \dots \qquad \text{Eq (2.3.3)}$$

where the Fourier coefficients are

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)(-1)^{S \cdot x} = \sum_{j=1}^{m} \hat{f}_j(S) \in [-1,1].$$

The notation $S \cdot x := \sum_{j \in S} x_j$ is used. The Fourier coefficients satisfy the relation

$$\sum_{S \subset [n]} (\hat{f}(S))^2 = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) = \hat{f}(\emptyset).$$

Proof. Firstly it is useful to realize that the condition $(\forall x \in \{0,1\}^n)(H_f |x\rangle = f(x) |x\rangle)$ is equivalent to $H_f = \sum_{x \in \{0,1\}^n} f(x) |x\rangle\langle x| = \sum_{x : f(x)=1} |x\rangle\langle x|$. Secondly it can be seen that $Z |0\rangle = |0\rangle$ and $Z |1\rangle = -|1\rangle$, thus the product of Z_j acts as

$$\prod_{j \in S} Z_j |x\rangle = \xi_S(x) |x\rangle ;$$

where $\xi_S(x) : \{0,1\}^n \to \{-1,+1\}$ is the parity function $\xi_S(x) = (-1)^{S \cdot x}$. Important observation is that the set of parity functions on n-bits gives an orthonormal basis for real functions:

$$\langle f, g \rangle := \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) g(x).$$

Using this observation any Boolean function $f : \{0,1\}^n \to \{0,1\}$ may be written using Fourier expansion

$$f(x) = \sum_{S \subset [n]} \hat{f}(S) \xi_S(x), \qquad \text{Eq (2.3.4)}$$

with Fourier coefficients given as

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) \xi_S(x) = \langle f, \xi_S \rangle. \qquad \text{Eq (2.3.5)}$$

□

It is important to realize that the class of functions that can be represented using this method is broader than only the class of Boolean functions. This fact is illustrated in the following theorem.

[4] To improve legibility the Pauli gates will be noted in the following way for the rest of this section: $\sigma_i^x = X_i, \sigma_i^y = Y_i, \sigma_i^z = Z_i$.

Theorem 2.11 (Weighted Boolean function Fourier transformation). *For an n-bit function* $f : \{0,1\}^n \to \mathbb{R}$ *given as* $f(x) = \sum_{j=1}^{m} w_j f_j(x)$, *where* $(\forall j \in [m])(w_j \in \mathbb{R}$ *and* $f_j : \{0,1\}^n \to \{0,1\}$[5], *Hamiltonian on n-qubits, which satisfies* $H_f |x\rangle = f(x) |x\rangle$, *is given as*

$$H_f = \sum_{S \subset [n]} \hat{f}(S) \prod_{j \in S} Z_j = \sum_{j=1}^{m} w_j H_{f_j}, \qquad \text{Eq (2.3.6)}$$

where Fourier coefficients are

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)(-1)^{S \cdot x} = \sum_{j=1}^{m} \hat{f}_j(S) \in \mathbb{R}$$

and Hamiltonians H_{f_j} *are given by* Eq (2.3.4)

Proof. The proof is a direct consequence of application of the previous theorem and the linearity of the formula Eq (2.3.3) in this theorem. □

Although it is not necessary in this case, another useful tool in designing Hamiltonians for more complex problems are the *composition rules*.

Theorem 2.12 (Composition rules). *Let* $f, g : \{0,1\}^n \to \{0,1\}$ *be Boolean functions with Hamiltonian representations* H_f *and* H_g. *The Hamiltonian that represent the basic operations on* f *and* g *are derived by the following rules:*

$$H_{\neg f} = I - H_f \qquad H_{f \implies g} = I - H_f + H_f H_g \qquad \text{Eq (2.3.7)}$$

$$H_{f \wedge g} = H_f H_g \qquad H_{f \vee g} = H_f + H_g - H_f H_g \qquad \text{Eq (2.3.8)}$$

$$H_{f \oplus g} = H_f + H_g - 2 H_f H_g \qquad H_{af+bg} = a H_f + b H_g \quad a, b \in \mathbb{R} \qquad \text{Eq (2.3.9)}$$

Proof. First the Boolean operations are translated using logical identities into $(\cdot, +)$ formulas. Then the formulas modified to Hamiltonians using the fact that Fourier transform is linear and that the logical values 1 and 0 can be represented by the identity matrix (I) and zero matrix (O).

$$(\neg f) = (1 - f) \qquad (f \implies g) = (\neg f + fg) \qquad \text{Eq (2.3.10)}$$

$$(f \wedge g) = (fg) \qquad ((f \vee g) = (f + g - fg) \qquad \text{Eq (2.3.11)}$$

$$(f \oplus g) = (f + g - 2fg) \qquad (af + bg) = (af) + (bg) \qquad \text{Eq (2.3.12)}$$

□

The theorems can be used to derive the Hamiltonian representations of basic Boolean functions that can be then further combined using composition rules theorem.

The correct phase Hamiltonian definition for MaxCut can now derived by first observing that the objective function takes simple form

$$C(x) = \sum_{\langle u,v \rangle} (x_u \oplus x_v), \qquad \text{Eq (2.3.13)}$$

where each summand is 0 is vertices are in the same partition and 1 if the are not, and the information from the table 2.1.

[5] $(\forall k \in \mathbb{N})([k] := 1, 2, \ldots, k)$

$f(x)$	H_f	$f(x)$	H_f
x	$\frac{1}{2}I - \frac{1}{2}Z$	$\neg x$	$\frac{1}{2}I - \frac{1}{2}Z$
$x_1 \wedge x_2$	$\frac{1}{4}I - \frac{1}{4}(Z_1 + Z_2 - Z_1Z_2)$	$\bigwedge_{j=1}^{k} x_j$	$\frac{1}{2^k}\prod_j (I - Z_j)$
$x_1 \vee x_2$	$\frac{3}{4}I - \frac{1}{4}(Z_1 + Z_2 + Z_1Z_2)$	$\bigvee_{j=1}^{k} x_j$	$I - \frac{1}{2^k}\prod_j (I + Z_j)$
$x_1 \oplus x_2$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2$	$x_1 \implies x_2$	$\frac{3}{4}I + \frac{1}{4}(Z_1 - Z2 + Z1Z2)$

Table 2.1: Hamiltonian representation of Boolean functions

Definition 2.13 (Phase Hamiltonian of the MaxCut)**.** The objective function Eq (2.3.2) of MaxCut is represented according to formula $C(x) = \sum x_u \oplus x_v$ in the table 2.1 by a phase Hamiltonian

$$H_P = \sum_{\langle u,v\rangle} H_{p\langle u,v\rangle} = \sum_{\langle u,v\rangle} \frac{1}{2}(1 - Z_uZ_v). \qquad \text{Eq (2.3.14)}$$

It should be mentioned that this form of Hamiltonian expressed using the Pauli operators is often referd to as Ising Hamiltonian[23].

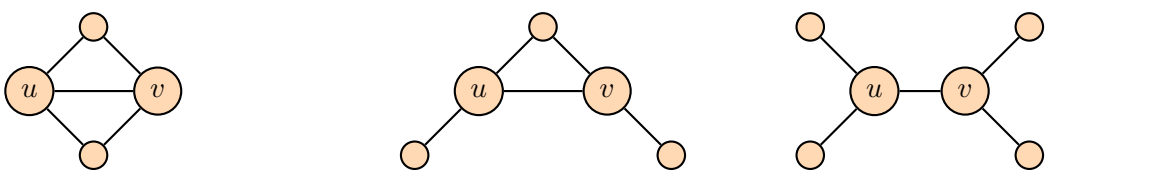
The expectation value of $\langle H_P\rangle$ is

$$\begin{aligned}\langle H_P\rangle &= \sum_{\langle u,v\rangle} \langle\gamma,\beta|H_{p\langle u,v\rangle}|\gamma,\beta\rangle \\ &= \sum_{\langle u,v\rangle} \langle s|U_P^\dagger(\gamma_1)\dots U_M(\beta_p)H_{p\langle u,v\rangle}U_M(\beta_p)\dots U_P(\gamma_1)|s\rangle\,.\end{aligned} \qquad \text{Eq (2.3.15)}$$

To perform the classical optimization in QAOA the formula (2.3.15) needs to be simplified. The approach to simplification can be illustrated for $p = 1$, in this case it can be seen that the mixing operators "partially commute" through the $H_{P\langle u,v\rangle}$ and cancel.

$$\begin{aligned}&U_P^\dagger(\gamma_1)U_M^\dagger(\beta_1)H_{P\langle u,v\rangle}U_M(\beta_1)U_P(\gamma_1) = \\ &= U_P^\dagger(\gamma_1)\mathrm{e}^{\mathrm{i}\beta_1(X_u+X_v)}H_{P\langle u,v\rangle}\mathrm{e}^{-\mathrm{i}\beta_1(X_u+X_v)}U_P(\gamma_1)\end{aligned} \qquad \text{Eq (2.3.16)}$$

This for arbitrary p means that the operator involves only some edges, specifically the edges at most p steps away from the inspected edge $\langle u,v\rangle$. This implies that only certain possible subgraphs, each containing number of qubits independent of n because the degree is bounded, need to be analysed to evaluate $\langle H_P\rangle$. This reduction can be shown on input graphs of fixed degree 3 and $p = 1$. In this case the possible subgraphs for edge $\langle u,v\rangle$ are in figure 2.3. For the analysis of a subgraph G we define



(a) : Subgraph type G_4 **(b) :** Subgraph type G_5 **(c) :** Subgraph type G_6, tree

Figure 2.3: Possible subgraphs for graph of fixed degree 3 and $p = 1$.[16]

$$
\begin{aligned}
C_G &:= \sum_{\langle u,v\rangle \in G} C_{\langle u,v\rangle} \\
H_{PG} &:= \sum_{\langle u,v\rangle \in G} H_{P\langle u,v\rangle} \qquad\qquad U_{PG}(\gamma) := \exp\{-\mathrm{i}\gamma C_G\} \\
H_{MG} &:= \sum_{j\in G} X_j \qquad\qquad U_{MG}(\beta) := \exp(-\mathrm{i}\beta H_{MG}) \\
|s_G\rangle &:= \prod_{j\in G} |+\rangle \,.
\end{aligned}
$$

Contribution to $\langle H_P\rangle$ of an edge $\langle u,v\rangle$ considering the subgraph $G_{(u,v)}$ associated with it is

$$
\begin{aligned}
f_g(\vec{\gamma},\vec{\beta}) = \\
\left\langle s_{G_{(u,v)}} \middle| U^\dagger_{PG_{(u,v)}}(\gamma_1)\ldots U^\dagger_{MG_{(u,v)}}(\beta_p) H_{P\langle u,v\rangle} U_{MG_{(u,v)}}(\beta_p)\ldots U_{PG_{(u,v)}}(\gamma_1) \middle| s_{G_{(u,v)}} \right\rangle . \quad \text{Eq (2.3.17)}
\end{aligned}
$$

Since all isomorphic subgraphs produce the same contributions, the number of occurrences of different subgraph types can be taken as a weight w_g in sum over subgraph types g.

$$
\langle H_P\rangle (\vec{\gamma},\vec{\beta}) = \sum_g w_g f_g(\vec{\gamma},\vec{\beta}), \qquad \text{Eq (2.3.18)}
$$

where the weight w_g denotes number of subgraphs of type g. The benefit of this method for the classical computation of $\langle H_P\rangle$ is that the function f_g are independent of m and n. It is useful to note that the upper bound for the number of qubits involved in subgraph contribution Eq (2.3.17) is the number of qubits in tree type subgraph. In case of a graph with maximum degree d this number is

$$
N_{\text{tree}} = 2\cdot\frac{(d-1)^{p+1}-1}{(d-1)-1}, \qquad \text{Eq (2.3.19)}
$$

showed in [16]. Moreover, the number of types of subgraphs is finite for each p. The values of the m and n parameters affect only the w_g can be found from the complete graph using little resources. From this follows that the evaluation of $\langle H_P\rangle$ using Eq (2.3.18) does not require resources growing with n.

To evaluate the performance of the QAOA it is useful to focus of the "concentration of results" of algorithm. In order to make the proofs easier to read, the objective function C will be considered to be an abstract equivalent of H_P, So these two symbols will be interchanges at appropriate places. The previously mentioned method can be also used to derive the upper bound for standard deviation using the formula for an observable M

$$
[\Delta(M)]^2 = \left\langle (M-\langle M\rangle)^2 \right\rangle = \left\langle M^2 \right\rangle - \langle M\rangle^2 , \qquad \text{Eq (2.3.20)}
$$

proved in [4]. The standard deviation in the analysed case can be written as

$$
\begin{aligned}
[\Delta(C)]^2 = \left\langle \vec{\gamma},\vec{\beta} \middle| C^2 \middle| \vec{\gamma},\vec{\beta} \right\rangle - \left\langle \vec{\gamma},\vec{\beta} \middle| C \middle| \vec{\gamma},\vec{\beta} \right\rangle^2 = \\
= \sum_{\langle u,v\rangle,\langle j,k\rangle \in E} \Big[\langle s|U_P^\dagger(\gamma_1)\ldots U_M(\beta_p) H_{P\langle u,v\rangle} H_{P\langle j,k\rangle} U_M(\beta_p)\ldots U_P(\gamma_1)|s\rangle \\
- \langle s|U_P^\dagger(\gamma_1)\ldots U_M(\beta_p) H_{P\langle u,v\rangle} U_M(\beta_p)\ldots U_P(\gamma_1)|s\rangle \\
\cdot \langle s|U_P^\dagger(\gamma_1)\ldots U_M(\beta_p) H_{P\langle j,k\rangle} U_M(\beta_p)\ldots U_P(\gamma_1)|s\rangle \Big] \qquad \text{Eq (2.3.21)}
\end{aligned}
$$

The terms in sum in Eq (2.3.21), where the subgraphs $G_{(u,v)}$ and $G_{(j,k)}$ do not share common qubits will vanish. This condition is fulfilled if there is no path connecting $G_{(u,v)}$ with $G_{(j,k)}$ of length $2p+1$

or shorter. This implies that for each edge $\langle u, v\rangle$ the number of edges which can contribute to sum in Eq (2.3.21) is upper bound by N_{tree} from Eq (2.3.19), where $p := 2p + 1$. The summands can be also estimated, they are at most 1 in norm. Thus, the standard deviation $[\Delta(C)]$ can be bound as

$$[\Delta(C)]^2 = \left\langle \vec{\gamma}, \vec{\beta} \middle| C^2 \middle| \vec{\gamma}, \vec{\beta} \right\rangle - \left\langle \vec{\gamma}, \vec{\beta} \middle| C \middle| \vec{\gamma}, \vec{\beta} \right\rangle^2 \leq 2 \cdot \frac{(d-1)^{2p+2} - 1}{(d-1) - 1} \cdot m, \qquad \text{Eq (2.3.22)}$$

where d is the maximum degree of the graph and $|E| = m$. Thus, for fixed v and p the standard deviation $[\Delta(C)]$ is at most of the order $\sqrt{m}$. The interpretation of this result is that the sample mean calculated of order m^2 values of objective function $C(z)$ will be within 1 of $\langle H_P\rangle(\vec{\gamma}, \vec{\beta})$ with probability equal to $(1 - m^{-1})$. This generally means that there is only a small probability that we get an answer string z with $C(z)$ much bigger than $\langle H_p\rangle$.

Another significant result that quantifies the performance of QAOA for a specific problem is the approximation ratio for MaxCut on 3-regular graphs, also known as cubic graphs, for $p = 1$ which was shown in [16]. The first step is to analyse the investigated graphs. A cubic graph with vertices $|V| = n$ has $|E| = \frac{3n}{2}$ edges, because each vertice has three neighbours. Then the number of following subgraphs is read from the graph. Where the leaving edges in $G_\triangle$ end in distinct vertices and the two

(a) : Subgraph type $G_\triangle$ **(b) :** Subgraph type $G_\square$

Figure 2.4: Significant subgraphs for 3-regular graph, [16]

edges leaving $G_\square$ are not the same edge.

$$N_\triangle := \#(G_\triangle) \qquad \text{Eq (2.3.23)}$$
$$N_\square := \#(G_\square) \qquad \text{Eq (2.3.24)}$$

$G_\triangle$ and $G_\square$ do not have common vertices, this implies that $3N_\triangle + 4N_\square \leq n$. Using the notation from figure 2.3 it can be seen that the $G_\square$ consists of 1 edge with G_4 type subgraph and 4 edges with G_5 type subgraph, while the $G_\triangle$ has 3 edges with G_5 type subgraph. Other $(\frac{3n}{2} - 5N_\square - 3N_\triangle)$ edges there are G_6 type subgraph. Thus, $\langle H_P\rangle$ can be expressed as

$$\langle H_P\rangle = N_\square f_{G_4}(\gamma, \beta) + (4N_\square + 3N_\triangle) f_{G_5}(\gamma, \beta) + (\frac{3n}{2} - 5N_\square - 3N_\triangle) f_{G_6}(\gamma, \beta). \qquad \text{Eq (2.3.25)}$$

From the definition Eq (2.1.11) now follows that M_1 is function only of $n, N_\square, N_\triangle$

$$M_1(n, N_\square, N_\triangle) = \max_{\gamma,\beta} \langle H_P\rangle(\gamma, \beta). \qquad \text{Eq (2.3.26)}$$

Last step is to give an upper bound for the number of satisfied edges. For both subgraphs $G_\triangle$ and $G_\square$ the best cut leaves 1 edge not crossed. This gives the upper bound $(\frac{3n}{2} - N_\square - N_\triangle)$. Hence the ratio is

$$R \geq \frac{M_1(n, N_\square, N_\triangle)}{(\frac{3n}{2} - N_\square - N_\triangle)} = \frac{M_1(1, n_\square, n_\triangle)}{(\frac{3n}{2} - n_\square - n_\triangle)}, \qquad \text{Eq (2.3.27)}$$

where $n_\square := \frac{N_\square}{n}$ and $n_\triangle := \frac{N_\triangle}{n}$. The last expression can be computed using classic computer. Using this method in publication [16] it was shown that the minimum of Eq (2.3.27) is reached for $n_\square = 0$ and $n_\triangle = 0$ and is equal to 0.6924. In conclusion this means that the approximation ratio is

$$R \geq 0.6924. \qquad \text{Eq (2.3.28)}$$

2.3.2 Alternative approach to $\langle H_P \rangle$ simplification using Pauli Solver

The presented way of solving the problem of $\langle H_P \rangle$ simplification is naturally not the only possible. An interesting alternative, named Pauli Solver, to the first approach was presented in publication [19]. This approach is based on using more general properties of the objective function elements, so it can be applied to broader class of objective functions. Its basic principles make the Pauli Solver easier to implement on classic computer.

Definition 2.14 (Pauli Solver algorithm). Let $C(z) = \sum_{i=1}^{m} C_i(z)$ be the objective function with phase Hamiltonians H_P and H_{Pi}. For $p = 1$ define $Q := U_M(\beta)U_P(\gamma)$. The Pauli expansion of $Q^\dagger H_{Pi} Q$ is

$$Q^\dagger H_{Pi} Q = a_0 I + \sum_{j=1}^{n} \sum_{\sigma \in \{X,Y,Z\}} a_{j\sigma}\sigma_j + \sum_{j \neq k} \sum_{\sigma,\lambda \in \{X.Y,Z\}} a_{jk\sigma\lambda}\sigma_j\lambda_k + \ldots, \qquad \text{Eq (2.3.29)}$$

where $a_\alpha \in \mathbb{R}$. Now the following property of initial state is used

$$\langle +|I|+\rangle = 1 \qquad \langle +|Y|+\rangle = 0 \qquad \text{Eq (2.3.30)}$$
$$\langle +|X|+\rangle = 1 \qquad \langle +|Z|+\rangle = 0 \qquad \text{Eq (2.3.31)}$$

to write

$$\langle \gamma, \beta | H_{Pi} | \gamma, \beta \rangle = \langle s | Q^\dagger H_{Pi} Q | s \rangle = a_0 + \sum_{j=1}^{n} a_{jX} + \sum_{j \neq k} a_{jkXX} + \ldots. \qquad \text{Eq (2.3.32)}$$

The following algorithm is applied to compute $\langle H_P \rangle$. This algorithm is then generalized for level $p \geq 1$

Algorithm 1: Pauli solver algorithm

Data: Objective function $C(z)$
Result: Expectation value $\langle H_P \rangle$ formula
begin
  Decompose $C(z)$ to sum of $C_i(z)$
  for i ***in*** $[m]$ **do**
    Apply the Pauli expansion Eq (2.3.29) on $Q^\dagger H_{Pi} Q$
    Keep only the terms containing X and I operators.
    Set $\langle H_P \rangle$ to the sum of remainder according to Eq (2.3.32)

by using $Q_p = U_M(\beta_p)U_P(\gamma_p)\ldots U_M(\beta_1)U_P(\gamma_1)$

Apart from the obvious application of Pauli Solver to simplify expressions programmatically, e.g. using a Python program, the identities of the solver can be used to prove important closed form formulas for MaxCut problem[19].

Theorem 2.15 (QAOA for MaxCut). *Let $|\gamma,\beta\rangle$ be a 1-level QAOA state on a graph G. Then for each edge $\langle u,v\rangle$ the expectation value $\langle\gamma,\beta|C_{uv}|\gamma,\beta\rangle$ is*

$$\begin{aligned}\langle C_{uv}\rangle(U,V,T) :=& \langle\gamma,\beta|C_{uv}|\gamma,\beta\rangle \\ =& \frac{1}{2}+\frac{1}{4}\sin(4\beta)\sin\gamma(\cos^U\gamma+\cos^V\gamma) \\ &-\frac{1}{4}\sin^2(2\beta)\cos^{U+V-2T}\gamma(1-\cos^T(2\gamma))\end{aligned} \tag{Eq (2.3.33)}$$

where $U=(\deg u-1), V=(\deg v-1)$ [6] *and T represents the number of triangles in the graph containing the edge $\langle u,v\rangle$*[7]*. Hence, the expectation value for whole graph G is*

$$\begin{aligned}\langle C\rangle = \langle\gamma,\beta|C|\gamma,\beta\rangle &= \sum_{\langle u,v\rangle}\langle C_{uv}\rangle(U(u,v),V(u,v),T(u,v)) \\ &= \sum_{(U,V,T)}\langle C_{uv}\rangle(U,V,T)n(U,V,T)\end{aligned} \tag{Eq (2.3.34)}$$

where $n(U,V,T)$ is the number of edges with parameters (U,V,T) in the whole graph G.

Proof. Let $Q=\mathrm{e}^{-\mathrm{i}\beta B}$ according to definition 2.14. Using the formula Eq (2.3.14) it can be seen that

$$\begin{aligned}\langle C\rangle &= \langle s|Q^\dagger H_P Q|s\rangle \\ &= \langle s|Q^\dagger\Big(\sum_{\langle u,v\rangle,|E|=m}\frac{1}{2}(I-Z_uZ_v)\Big)Q|s\rangle \\ &= \frac{m}{2}-\frac{1}{2}\sum_{\langle u,v\rangle}\langle s|Q^\dagger Z_uZ_vQ|s\rangle\,.\end{aligned} \tag{Eq (2.3.35)}$$

Hence, only the second term needs to be simplified further. As in Pauli Solver, commutation properties imply $\mathrm{e}^{\mathrm{i}\beta H_M}Z_uZ_v\mathrm{e}^{-\mathrm{i}\beta H_M}=\mathrm{e}^{2\mathrm{i}\beta X_u}\mathrm{e}^{2\mathrm{i}\beta X_v}Z_uZ_v$. Using the identities[4]

$$\mathrm{e}^{-\mathrm{i}\theta\sigma_i}=\cos(\theta)I-\mathrm{i}\sin(\theta)\sigma_i \tag{Eq (2.3.36)}$$

where θ is an angle in $[0,2\pi]$ and σ_i for $i\in\{0,1,2,3\}$ is a Pauli matrix, and

$$\mathrm{i}X_iZ_i=-Y_i \tag{Eq (2.3.37)}$$

and substitution $a=\cos(2\beta), b:=\sin(2\beta)$, we can see that

$$\begin{aligned}\mathrm{e}^{\mathrm{i}\beta H_M}Z_uZ_v\mathrm{e}^{-\mathrm{i}\beta H_M} =& \big(\cos(2\beta)\big)^2Z_uZ_v+\sin(2\beta)\cos(2\beta)(Y_uZ_v+Z_uY_v) \\ &+\big(\sin(2\beta)\big)^2Y_uY_va^2Z_uZ_v \\ &+ab(Y_uZ_v+Z_uY_v)+b^2Y_uY_v\end{aligned} \tag{Eq (2.3.38)}$$

This expression can now be simplified term by term. Starting with first term, the commutation property of Z_uZ_v with $\mathrm{e}^{\pm H_P}$ along with identity Eq (2.3.30) imply that its contribution will be zero. For the second term a procedure similar to Eq (2.3.38) is used

$$\begin{aligned}\langle s|\mathrm{e}^{\mathrm{i}\gamma H_P}Y_uZ_v\mathrm{e}^{-\mathrm{i}\gamma H_P}|s\rangle &= \langle s|\mathrm{e}^{2\mathrm{i}\gamma C_{uv}}\mathrm{e}^{2\mathrm{i}\gamma C_u}Y_uZ_v|s\rangle \\ &= \langle s|\mathrm{e}^{-\mathrm{i}\gamma Z_uZ_v}\mathrm{e}^{-\mathrm{i}\gamma\sum_{w\in W}Z_uZ_w}Y_uZ_v|s\rangle \\ &= [\text{Substitution: } c:=\cos\gamma, d=\sin\gamma] \\ &= \langle s|(cI-\mathrm{i}dZ_uZ_v)\prod_{j=1}^{U}\Big(cI-\mathrm{i}dZ_uZ_{w_j}\Big)Y_uZ_v|s\rangle\end{aligned} \tag{Eq (2.3.39)}$$

[6] deg is the degree of a vertex, i.e. the number of its neighbours

[7] The case of a graph with no triangles is covered in Theorem 2.16.

where $C_x = \sum_{w \in W} C_{xw}$ and $W = \{w \in E | w \text{ is neighbour of } u\} \setminus \{v\}$. In the last result it can be seen that upon expanding the product the only term in resulting sum with non-zero contribution will be

$$\begin{aligned}\langle s|\mathrm{e}^{\mathrm{i}\gamma H_P} Y_u Z_v \mathrm{e}^{-\mathrm{i}\gamma H_P}|s\rangle &= \langle s|-\mathrm{i}dc^U Z_u Z_v I^{\otimes U} Y_u Z_v|s\rangle \\ &= \left[\text{Using identity: } Z_u Z_v I^{\otimes U} Y_u Z_v = -\mathrm{i}X_u\right] \\ &= \langle s|-\mathrm{i}dc^U(-\mathrm{i}X_u)|s\rangle \\ &= -\,\mathrm{i}dc^U\end{aligned} \qquad \text{Eq (2.3.40)}$$

Similarly for $\langle s|\mathrm{e}^{\mathrm{i}\gamma H_P} Z_u Y_v \mathrm{e}^{-\mathrm{i}\gamma H_P}|s\rangle = -dc^V$. The third term is simplified in following way

$$\begin{aligned}\langle s|\mathrm{e}^{\mathrm{i}\gamma H_P} Y_u Y_v \mathrm{e}^{-\mathrm{i}\gamma H_P}|s\rangle &= \langle s|\mathrm{e}^{2\mathrm{i}\gamma C_u}\mathrm{e}^{2\mathrm{i}\gamma C_v} Y_u Z_v|s\rangle \\ &= \langle s|\prod_{j=1}^{U}\left(cI - \mathrm{i}dZ_u Z_{w_j}\right)\prod_{k=1}^{V}\left(cI - \mathrm{i}dZ_u Z_{w_k}\right) Y_u Y_v|s\rangle\,.\end{aligned} \qquad \text{Eq (2.3.41)}$$

In the last expression the terms which contribute fall into several categories. The first category is

$$\left\langle c^{U+V-2}d^2\middle|(cI)^{U+V-2}(-\mathrm{i}dZ_u Z_w)(-\mathrm{i}dZ_v Z_w)Y_u Y_v\middle|c^{U+V-2}d^2\right\rangle \qquad \text{Eq (2.3.42)}$$

where the contribution is implied by the procedure applied before for second term. This category appears in the sum T times. If $T > 2$ other categories need to be considered because $Z_u Z_{w_i} Z_u Z_{w_i} = I$. The next category contains three different pairs of $Z_u Z_{w_i}$ and $Z_v Z_{w_i}$. This implies the contributions will be proportional to $\left(d^2\right)^3 = d^6$ and that this category contains $\binom{T}{3}$ terms. This observation can be generalized to formula

$$\begin{aligned}\langle s|\mathrm{e}^{\mathrm{i}\gamma H_P} Y_u Y_v \mathrm{e}^{-\mathrm{i}\gamma H_P}|s\rangle =& \binom{T}{1}(c^{U+V-2}d^2) + \binom{T}{3}(c^{U+V-6}d^6) + \binom{T}{5}(c^{U+V-10}d^{10}) + \dots \\ =& c^{U+V-2T}\sum_{j=1,3,5\dots}^{T}\binom{T}{j}c^{2(T-j)}d^{2j} \\ =& \left[\text{Binomial theorem for } N \in \mathbb{N}, A, B \in \mathbb{R}\right] \\ =& c^{U+V-2T}\frac{1}{2}(c^2+d^2)^T - (c^2+d^2)^T \\ =& \frac{1}{2}c^{U+V-2T}(1-\cos^T(2\gamma)),\end{aligned} \qquad \text{Eq (2.3.43)}$$

where the used binomial formula is $\sum_{i\in\{1,3,5,\dots\}}^{N}\binom{N}{i}A^{N-i}B^i = \frac{1}{2}((A+B)^N - (A-B)^N)$. The sum of the found simplifications gives proof of equation Eq (2.3.33) and sum over all edges $\langle u, v\rangle$ proofs Eq (2.3.34) □

What is left is to show the following theorem for much simpler case of graph without triangles.

Theorem 2.16 (QAOA for MaxCut for graph without triangles). *For QAOA on MaxCut applied to*

1. *a D-regular graph G without triangles the expectation value can be computed as*

$$\langle C\rangle = \frac{m}{2} + \frac{m}{2}\sin(4\beta)\cos^{D-1}\gamma. \qquad \text{Eq (2.3.44)}$$

The maximum value of $\langle C \rangle$ *is*

$$\max_{\gamma,\beta} \langle C \rangle = \frac{m}{2} + \frac{m}{2} \frac{1}{\sqrt{D}} \left(\frac{D-1}{D} \right)^{\frac{D-1}{2}}. \qquad \text{Eq (2.3.45)}$$

This value will be noted as $C^{reg}_{\max}(D) = \max_{\gamma,\beta} \langle C \rangle$ *For this case the approximation ratio 2.1 has lower bound*

$$\max_{\gamma,\beta} \langle R \rangle > \frac{1}{2} + \frac{1}{1\sqrt{\mathrm{e}D}} = \frac{1}{2} + \Omega\left(\frac{1}{\sqrt{D}} \right) \qquad \text{Eq (2.3.46)}$$

2. *an arbitrary graph* G *without triangles with maximum degree* D_G *with* N_D *vertices of degree* D*, for* $D \in \{2, 3, \dots\}$ *the expectation value can be computed as*

$$\langle C \rangle = \frac{m}{2} + \frac{1}{4} \sin(4\beta) \sin\gamma \sum_{D \in \{2,3,\dots\}} D \cos^{D-1} \gamma. \qquad \text{Eq (2.3.47)}$$

The $\max_{\gamma,\beta} \langle C \rangle$ *and* $\max_{\gamma,\beta} \langle R \rangle$ *are according to* Eq (2.3.45) *and* Eq (2.3.46) *respectively*

$$\max_{\gamma,\beta} \langle C \rangle \geq C^{reg}_{max}(D_G) > \frac{1}{2} + \frac{m}{2\sqrt{\mathrm{e}D_G}}, \qquad \text{Eq (2.3.48)}$$

$$\max_{\gamma,\beta} \langle R \rangle > \frac{1}{2} + \frac{1}{1\sqrt{\mathrm{e}D_G}}. \qquad \text{Eq (2.3.49)}$$

Proof. The equation Eq (2.3.44) is given by the previous Theorem 2.15, the following Eq (2.3.45) is derived by plugging angles from the next Theorem 2.18 into Eq (2.3.44). To get the third equation Eq (2.3.46) we use the approximations

$$\langle R \rangle \geq \frac{\langle C \rangle}{m} \qquad \text{Eq (2.3.50)}$$

and $\left(\frac{k}{k+1} \right)^k > \frac{1}{\mathrm{e}}$, both of which follow from respective definitions.

The edges one edge u of $\deg(u) = 1$ can be always cut, thus the considered $D \in \{2, 3, \dots\}$. Hence, the Eq (2.3.44) can be trivially modified to Eq (2.3.47). □

This theorem gives many interesting results without need to perform simulations such as those in [16]. The equation Eq (2.3.45) gives

D	2	3	4	5
$\max_{\gamma,\beta} \langle R \rangle$	0.75	0.69245	0.66238	0.64310

Table 2.2: Numeric values of approximation ratio for various graph degrees

From the second part of Theorem 2.16 it can be seen that $\max_{\gamma,\beta} \langle C \rangle > \frac{m}{2}$, and the approximation Eq (2.3.50) implies that approximation ratio can always be made greater than 0.5 by setting γ, β accordingly.

To get more valuable numeric results for graphs without triangles the following theorem[19] is to be proved.

Definition 2.17 (Optimal angle). [19] The angles $(\tilde{\gamma}, \tilde{\beta})$ are optimal, when the lower bound of approximation ratio is maximal for $(\tilde{\gamma}, \tilde{\beta})$, i.e.

$$\langle R \rangle (\tilde{\gamma}, \tilde{\beta}) = \max_{\gamma,\beta} \langle R \rangle . \quad \text{Eq (2.3.51)}$$

This condition can be approximated by calling $(\tilde{\gamma}, \tilde{\beta})$ optimal angles, if they maximize $\frac{\langle C \rangle}{m}$ according to Eq (2.3.50).

Theorem 2.18 (Properties of optimal angles for graph without triangles). *Consider 1-level QAOA applied to MaxCut problem on a graph G without triangles.*

1. *If G is a D-regular, then there is only one pair $(\tilde{\gamma}, \tilde{\beta})$*

$$(\tilde{\gamma}, \tilde{\beta}) = \left(\operatorname{arctg} \frac{1}{\sqrt{D-1}}, \frac{\pi}{8} \right), \quad \text{Eq (2.3.52)}$$

for $D \in \{2, 3, \dots\}$, such that all other optimal angles $(\tilde{\gamma}', \tilde{\beta}')$ satisfy $0 \leq \tilde{\gamma} < \tilde{\gamma}'$ and $0 \leq \tilde{\beta} < \tilde{\beta}$ Larger optimal angles are characterized by D.

- *For D even all optimal angles have forms:*

$$\left(\tilde{\gamma} + k\pi, \tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.53)}$$

$$\left(-\tilde{\gamma} + m\pi, -\tilde{\beta} + n\frac{\pi}{2} \right), \quad \text{Eq (2.3.54)}$$

  where $k, l, m, n \in \mathbb{Z}$

- *For D odd all optimal angles have forms:*

$$\left(\tilde{\gamma} + k2\pi, \tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.55)}$$

$$\left(-\tilde{\gamma} + k2\pi, -\tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.56)}$$

$$\left(-\tilde{\gamma} + (2k+1)\pi, \tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.57)}$$

$$\left(\tilde{\gamma} + (2k+1)\pi, -\tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.58)}$$

$$\text{Eq (2.3.59)}$$

  where $k, l \in \mathbb{Z}$.

2. *If G is an arbitrary graph with maximum degree D_G^{max} and minimum degree D_G^{min}, then there is only one pair $(\tilde{\gamma}, \tilde{\beta}) \in \left[0, \frac{\pi}{2}\right]^{\otimes 2}$*

$$\operatorname{arctg} \frac{1}{\sqrt{D_G^{max} - 1}} \leq \tilde{\gamma} \leq \operatorname{arctg} \frac{1}{\sqrt{D_G^{min} - 1}} \tilde{\beta} = \frac{\pi}{8} \quad \text{Eq (2.3.60)}$$

Using this pair $(\tilde{\gamma}, \tilde{\beta})$, all optimal angles can be written as

$$\left(\tilde{\gamma} + 2k\pi, \tilde{\beta} + l\frac{\pi}{2} \right), \quad \text{Eq (2.3.61)}$$

$$\left(-\tilde{\gamma} + 2m\pi, -\tilde{\beta} + n\frac{\pi}{2} \right), \quad \text{Eq (2.3.62)}$$

where $k, l \in \mathbb{Z}$.

D	2	3
$(\tilde{\gamma}, \tilde{\beta})$	$(\pi/4, \pi/8)$	$(0.6155, p/8)$

Table 2.3: Numeric values of optimal angles for various graph degrees

This theorem gives the values of optimal angles used in proof of 2.16.

Proof. To find the "smallest" optimal angles, the usual procedure for finding function extremes using partial derivatives is applied. The other optimal angles can be found by applying the following observations:

- The expectation value from equation Eq (2.3.44) from theorem 2.16 can be written as

$$\langle C \rangle (\gamma, \beta) = \frac{m}{2} + f(\gamma) \cdot g(\beta), \qquad \text{Eq (2.3.63)}$$

- $f(\gamma)$ and $g(\beta)$ are odd and periodic, i.e.

$$f(\gamma) = f(-\gamma) \qquad g(\beta) = g(-\beta) \qquad \text{Eq (2.3.64)}$$
$$f(\gamma) = f(\gamma + \pi) \qquad \text{For D odd} \qquad \text{Eq (2.3.65)}$$
$$f(\gamma) = f(\gamma + 2\pi) \qquad \text{For D even} \qquad \text{Eq (2.3.66)}$$
$$g(\beta) = g(\beta + \frac{\pi}{2}) \qquad \text{Eq (2.3.67)}$$

- $\sum_D DN_D = 2m$, where $m = |E|$, and $DN_D > 0, \forall D \in \{2, 3, \dots\}$. This means that Eq (2.3.44) can be viewed as a convex combination of Eq (2.3.47). Hence, we get the third part of theorem.

□

It is now straightforward to generalize the results for optimal angles to get the approximations for $\max_{\gamma,\beta} \langle C \rangle$ and $\max_{\gamma,\beta} \langle R \rangle$.

Corollary 2.19. *Consider QAOA applied to MaxCut problem on*

1. *a graph G, such that maximum degree is D_G and containing T triangles. Then*

$$\max_{\gamma,\beta} \langle C \rangle \geq \max \left\{ \frac{m}{2} + \frac{m}{2\sqrt{\mathrm{e} D_G}} - O\left(\frac{T}{D_G}\right), \frac{m}{2} \right\} \qquad \text{Eq (2.3.68)}$$

2. *a graph G, with maximum degree $D_G = \mathcal{O}(1)$. Then*

$$\max_{\gamma,\beta} \langle R \rangle \geq \frac{1}{2} + \frac{2}{2\sqrt{\mathrm{e} D_G}} - O\left(\frac{1}{D_G}\right). \qquad \text{Eq (2.3.69)}$$

Proof. The first statement is the result of plugging $\tilde{\gamma} = \operatorname{arctg} \frac{1}{\sqrt{D_G - 1}}, \tilde{\beta} = \frac{\pi}{8}$ from equation Eq (2.3.60) into Eq (2.3.44) and repeating selected of the approximation used to derive Eq (2.3.45). The theorems

for graph without triangles can be used because $\max_{\gamma,\beta} \langle C \rangle \geq \left\langle \tilde{\gamma}, \tilde{\beta} \middle| C \middle| \tilde{\gamma}, \tilde{\beta} \right\rangle$, where the term on right side is our approximation. For large T the expression $\frac{m}{2} + \frac{m}{2\sqrt{\mathrm{e}D_G}} - O\left(\frac{T}{D_G}\right)$ may become lower than $\frac{m}{2}$. In this case the trivial choice $\tilde{\gamma} = \tilde{\beta} = 0$ gives the other approximation.

To derive the second statement from the first, the lower bound for approximation ration Eq (2.3.50) is used along with in terms of $m = |E|$ we have $D_G(m) = \mathcal{O}(1)$ and $T = \mathcal{O}(m)$. □

The results from [19] were further developed and applied in [24].

To put the results above into perspective, few notable results are listed bellow. The result above improves upon the previous results from 2015 technical report [25] where the number of cut edges, i.e. $\max_{\gamma,\beta} \langle C \rangle$, for D-regular graph was bound by

$$\left(\frac{1}{2} + \frac{\mathcal{O}(1)}{\sqrt{D}\ln(D)}\right). \qquad \text{Eq (2.3.70)}$$

The approximation of $\langle C \rangle$ for D-regular triangle-free graphs is superior to best known classical alternative for $D > 3$. However, for $D = 3$, there is a classical algorithm with better result.[24]

D	Quantum $\langle C \rangle$	Classical $\overline{C(z)}$
> 3	$\left(\frac{1}{2} + \frac{0.3032}{\sqrt{D}}\right)$	$\left(\frac{1}{2} + \frac{0.3032}{\sqrt{D}}\right)$

Table 2.4: Comparison of classical and quantum algorithms for MaxCut[24]

2.4 Algorithms related to QAOA

Prior to discovery of the QAOA the Quantum Adiabatic Algorithm (QAA) was presented in [26]. As mentioned in 2.2 the QAA find an optimal solution unlike the QAOA, which find an approximate solution. However, the QAOA has several advantages over other algorithms. Firstly its low level variant has simpler implementation on quantum computers capable of working with arbitrary unitary gates. Secondly there are cases where the QAOA succeeds, whereas the QAA fails. An example was presented in publication [27]. It was shown that if the objective function is symmetric in all n bits, i.e. it depends only on the Hamming weight[8], the QAA produces false minimum for subexponential run time T. The properties of QAOA suggest that for large n there are angles γ and β such that the final state is near true solution for p as little as 1. Finally, its approximation of solution improves with increasing p. On the other hand the success probability of the QAA depends on total run time T. This relation may not be monotonous and the necessary run time T may grow exponentially with n.[16]

Since the QAOA was first presented, new modifications were suggested. The Quantum Alternating Operator Ansatz may be considered as an extension of the QAOA specialized in problems with

[8] The distance from all zero string, equal to number of ones in string.[4]

constraints. The essence of the extension is usage of partial mixing operators. The mixing operators $U_M(\beta)$ are decomposed into multiple partial mixing operators $U_{M,j}(\beta)$. This approach allows more convenient mapping of general problems to quantum circuits.[13, 28]

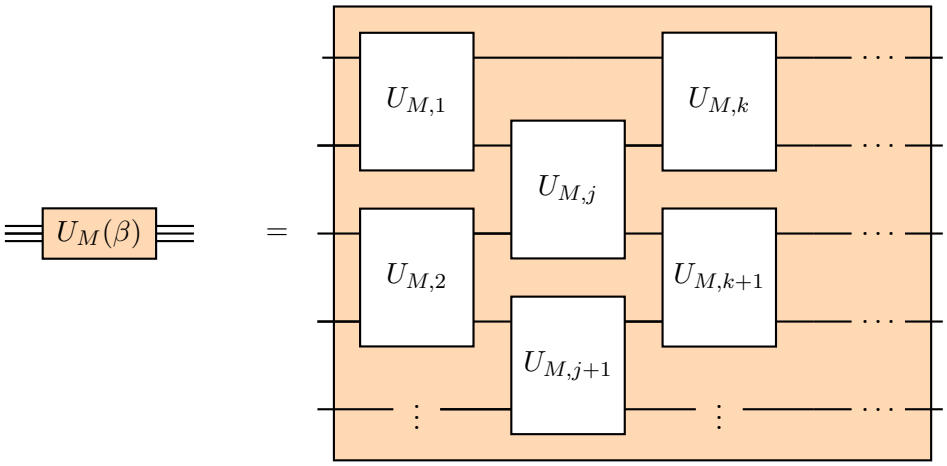


Figure 2.5: Schematic of a mixing operator decomposition into partial mixing operators[28]

Chapter 3

Variational quantum eigensolver

A common problem in many fields of research is to find the eigenvalues of a given operator, i.e. the values λ for a linear operator T such that

$$T(\mathbf{v}) = \lambda \mathbf{v}, \qquad \text{Eq (3.0.1)}$$

where v an eigenvector. In the terminology of quantum physics the problem can be seen as finding the states of the quantum system. A particular interest is in finding the state with the lowest energy, i.e. the ground state. To solve the eigenvalue problem, number of specialized algorithms were developed, which are referred to as eigensolvers.

Definition 3.1 (Eigenvalue algorithm, Eigensolver)**.** The group of algorithms used to find or estimate one or more eigenvalues of an operator, i.e. λ from Eq (3.0.1), is referred to as eigenvalue algorithms or eigensolvers. Some eigensolvers may also compute the eigenvectors of the operator. The eigenvalues can be than used to explore the spectrum of the studied operator.[20]

It needs to be emphasized that the algorithms for operators given by matrix with size larger than four are, except for special matrices, numeric. This means that their results are approximations of the eigenvalues. The numerous examples of classical eigensolvers includes Jacobi eigenvalue algorithm, power iteration algorithm and QR algorithm for Hessenberg matrices. But the broad definition 3.1 applies to both the classic and the quantum eigensolvers. In the case of quantum algorithms the problems are formulated in terms of eigenstates of Hamiltonians rather than general operators or matrices.

This brings us to the other algorithm covered in this project, that is the Variational Quantum Eigensolver (VQE). The VQE is an eigenvalue algorithm designed to find the upper bound for the ground-state of a given Hamiltonian. It was presented in 2015 article [29] along with practical a demonstration using photonic hybrid quantum computer and applied to problems of quantum chemistry. This result information from VQE can be then used to study properties of molecules, by the means of their electron structures[30], and condensed-state matter[31]. This gives the VQE a wide range of application including quantum chemistry[13], material sciences[32], medicine[33] and nuclear physics[34]. The topic of applications for discussed algorithms is continued in Chapter 5. Although VQE is not a quantum simulation, which could directly profit from the fact that the resources necessary for

simulating a quantum system grow exponentially for a conventional simulation and only linearly for quantum simulation, the VQE may still offer quantum advantage in comparison to classical eigenvalue algorithms.

As it was noted in the introductory chapter, VQE belongs to family of VQA algorithms which use an optimization loop and parametrized ansatz to reach the solution of the problem. The VQE is similar to QAOA described in chapter 2 in that it is based around applying a quantum circuit with parametrized Hamiltonian, that is initialized as an ansatz, on a quantum register in initial state $|0\rangle^{\otimes N}$, which is different from the initial state used in QAOA 2.5, and optimizing the parameters according to the result of measurement. This variational optimization of ansatz parameters is performed to minimize the trial energy[1] while satisfying the constraint to stay higher than exact ground-state energy by means of variational principle[35, 36][37]. Before we can proceed to describe the algorithm in pseudocode manner in Section 3.1.1, the motivation and theory needs to be presented.

3.1 Description of VQE

The theoretical foundations of VQE from article [29] were later built upon in research [38], to extend the possibilities of the new algorithm. In order to give a consistent picture of the algorithm, the formalism used in more recent [38] will be employed.

To describe the VQE we will consider two quantum systems, a quantum computer C consisting of N qubits and target system T with Hamiltonian H, which is the object of our study. The only requirement for T is that, the Hamiltonian H has to act on space of less than N qubits. Considering an operator O acting on system T, as a more general case instead of H. The second requirement is that O needs to have a decomposition

$$O = \sum_{\alpha} w_\alpha O_\alpha, \qquad \text{Eq (3.1.1)}$$

where O_α are terms easily measurable in system C. This requirement is implied by the limitations of the computer C to projective measurements. It is worth noting that the limitation can be solved by using the ancillary qubits, this allows the application of POVM measurements[4].

As mentioned in the introduction of this chapter, the VQE is based on optimizing the lowest upper bound for the minimal expectation value of a given observable, in the studied case a Hamiltonian H, by finding the optimal parametrization of trial wave function $|\psi(\boldsymbol{\theta})\rangle$, where $\boldsymbol{\theta}$ represents a vector of parameters. This can be seen from the variational theorem of quantum mechanics, which gives the following inequality for $|\psi(\boldsymbol{\theta})\rangle$ of C applied to Hamiltonian H of T,

$$E_0 \leq \frac{\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle}{\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta})\rangle}, \qquad \text{Eq (3.1.2)}$$

where E_0 is the ground-state for H, equivalent to the lowest eigenvalue, and $|\psi(\boldsymbol{\theta})\rangle$ is the trial wave function, which has the role of an approximation of an eigenvector. From Eq (3.1.2) follows the conclusion that by minimizing the right-hand side we can approximate the ground-state for H. The trial state $|\psi(\boldsymbol{\theta})\rangle$ is constructed according to the ansatz from the initial state $|\mathbf{0}\rangle := |0\rangle^{\otimes N}$ using a unitary

[1] Energy in the meaning of expectation value for hamiltonian with set *trial* parameters

operator $U(\theta)$ ($U(\theta)$ represents the part of circuit, not a single gate), where $\boldsymbol{\theta}$ is the parameter vector. To keep the inequality simple we require $|\psi(\boldsymbol{\theta})\rangle$ to be normalized, i.e. $\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta})\rangle = 1$. The equation Eq (3.1.2) takes form $\langle H\rangle(\boldsymbol{\theta}) := \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle \geq E_0$. This procedure ends with measurement of the trial energy, which is to be minimized

$$E_{\mathrm{VQE}} = \min_{\theta} \langle \mathbf{0}|U^{\dagger}(\boldsymbol{\theta})HU(\boldsymbol{\theta})|\mathbf{0}\rangle\,. \qquad \text{Eq (3.1.3)}$$

Appling the requirement Eq (3.1.1) we conclude that in order to implement the Hamiltonian, it has to be first mapped to spin operators, which are converted to so-called Pauli strings, i.e. a tensor products of N Pauli operators Eq (1.1.2), where N stands for the number of qubits in the circuit. Hence, the desired Hamiltonian in matrix form can be rewritten in form a weighted sum of Pauli strings

$$H = \sum_{\alpha}^{N_P} w_\alpha P_\alpha, \qquad \text{Eq (3.1.4)}$$

where $N_P \in \mathbb{N}$ is the number of Pauli strings in the Hamiltonian, $w_\alpha \in \mathbb{R}$ are the weight coefficients and finally $P_\alpha \in \{I, X, Y, Z\}^{\otimes N}$. The decomposition of Hamitonian in Eq (3.1.4) is possible for the Ising Hamiltonins[2] and more general Hamiltonian by virtue of encoding which is described for molecular Hamiltonians in Section 3.2.1. Thus, Eq (3.1.3) takes form

$$E_{\mathrm{VQE}} = \min_{\theta} \sum_{\alpha}^{N_P} w_\alpha \langle \mathbf{0}|U^{\dagger}(\theta)P_\alpha U(\theta)|\mathbf{0}\rangle\,. \qquad \text{Eq (3.1.5)}$$

This from clearly hints that a great advantage of VQE lies in parallelization. Each summand can be evaluated in parallel, leaving only summation and optimization to the conventional computer.

To see one of the advantages of VQE, we rewrite the trial function $|\psi(\boldsymbol{\theta})\rangle$ as a density matrix[4]

$$\rho(\boldsymbol{\theta}) = |\psi(\boldsymbol{\theta})\rangle\langle\psi(\boldsymbol{\theta})| \qquad \text{Eq (3.1.6)}$$

and write the expectation value formula

$$\langle H\rangle(\boldsymbol{\theta}) = \mathrm{Tr}[\rho(\boldsymbol{\theta})H]. \qquad \text{Eq (3.1.7)}$$

The ground-state variational principle still holds for an arbitrary $\boldsymbol{\theta}$, so $\langle H\rangle(\boldsymbol{\theta}) = \mathrm{Tr}[\rho(\boldsymbol{\theta})H] \geq E_0$. The equation Eq (3.1.7) does not apply only to pure states as Eq (3.1.6), but also to cases where the density matrix represents ensemble influenced by environment. This fact implies that the optimization in VQE is capable of suppressing some errors and noise in form mixed states.

It is worth mentioning the fact that the objective of the described algorithm is to find the approximation of ground-state E_0 does not mean VQE cannot be used to find higher eigenstates representing excited states. This can be achieved by utilizing the folded spectrum method[38] by applying the VQE to modified Hamiltonian $\tilde{H} := (H - \gamma I)^2$ where $\gamma \in \mathbb{R}$. This transformation ensures the ground-state of $\tilde{H}$ corresponds to the eigenvalue of H closest to the parameter γ.

3.1.1 VQE steps outline

The equations Eq (3.1.3) and Eq (3.1.5) suggest the individual steps of the algorithm:

[2] The Ising Hamitonians can also be product of encoding of a boolean function as it was shown in Eq (2.3.3).

1. Choice of ansatz, trial wavefunction parametrization by $\boldsymbol{\theta}$
2. Preparation of $|\psi(\boldsymbol{\theta})\rangle$
3. Measurement of $\langle H\rangle(\boldsymbol{\theta})$ expectation value
4. Classical optimization to retrieve new parameter vector $\boldsymbol{\theta}$ that decreases the expectation value $\langle H\rangle(\boldsymbol{\theta})$
5. Iteration of the last four steps until $\langle H\rangle(\boldsymbol{\theta})$ converges
6. The final parameter vector $\boldsymbol{\theta}$ defines the eigenvector approximation $|\psi(\boldsymbol{\theta})\rangle$ while the final value of $\langle H\rangle(\boldsymbol{\theta})$ holds the upper bound approximation of the lowest eigenvalue

To explore the properties of VQE i greater detail, the next subsection describes the individual components of VQE implementation.

3.2 Components of VQE

The VQE as described in previous subsection can be broken down into multiple components. For some component there exist various possible renditions suitable for different implementations. Term *VQE stack* is used to refer to a such implementation composed of selected components, The information presented below is based on recent publication [6], which details current progress on different components of VQE.

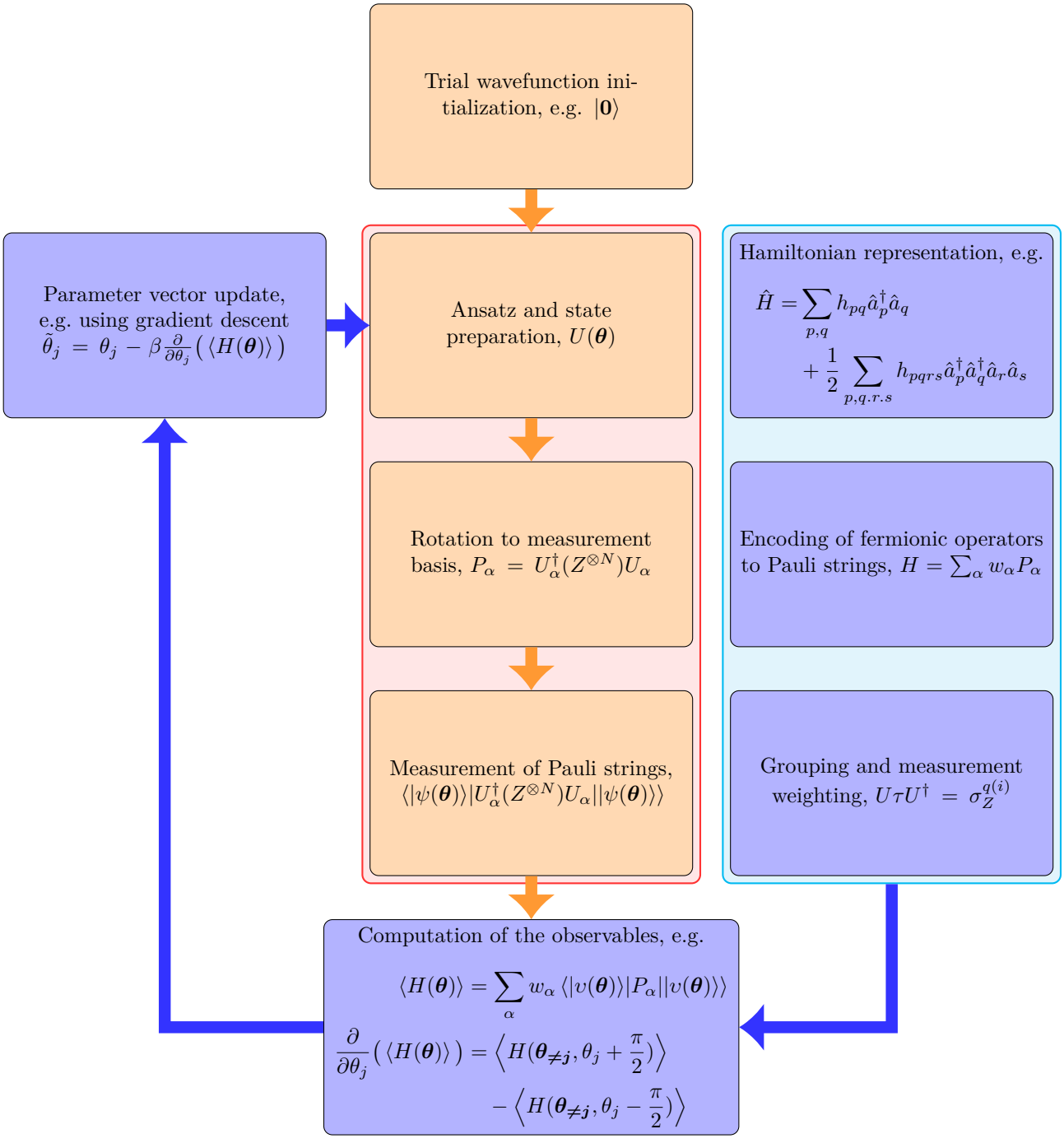


Figure 3.1: Schematic of the general VQE stack, based on[6]; The blocks with orange background ( ) represent the operations performed of a quantum computer, blocks with blue ( ) background represent operations on classical computer, blue arrows represent transported classical information, orange arrows represent transported quantum information, blocks grouped into red ( ) box can be performed in parallel and blocks grouped into cyan ( ) box are performed in preparatory phase of the algorithm; examples are further described in the Subsection 3.2.

3.2.1 Hamiltonian

Before any steps to prepare VQE can be taken, the studied system[3] needs to be defined. Since VQE was developed for the application in quantum chemistry, the most explored system models are the *ab initio* molecular, solid-state system or spin-lattice model[39].

To illustrate the structure of Hamiltonian we use the *ab initio molecular* Hamiltonian as an example.

The *ab initio molecular* Hamiltonian plays role of an operator representation of total energy for a studied molecular system. The objective to find the total energy requires to determine the electronic wavefunction equivalent to the probability amplitudes for the electrons around the nuclei. The input data for this computation are the atomic composition of the system and the relative positions of the nuclei. Widely used approach for the non-relativistic case is the Born-Oppenheimer approximation, which simplifies the model by considering the nuclei static[4]. For this approximation the Hamiltonian can be set up to a constant as following.

Definition 3.2 (ab initio molecular Hamiltonian)**.** Let $\boldsymbol{R}_k$ be the position and Z_k be the atomic number of the nucleus k, $\boldsymbol{r_i}$ be the i-th electron position, M_i be the i-th electron mass, e be the elementary charge, $\hbar$ be the reduced Planck constant and ∇_i^2 be the i-th electron Laplace operator. Then we define the *ab initio molecular* Hamiltonian for Born Oppenheimer approximation as

$$H := T_\mathrm{e} + V_\mathrm{ne} + V_\mathrm{ee}, \qquad \text{Eq (3.2.1)}$$

where

$$T_\mathrm{e} := -\sum_i \frac{\hbar^2}{2M_i}\nabla_i^2, \qquad \text{Eq (3.2.2)}$$

$$V_\mathrm{ne} := -\sum_{i,k} \frac{e^2}{4\pi\epsilon_0}\frac{Z_k}{|\boldsymbol{r}_i - \boldsymbol{R}_k|}, \qquad \text{Eq (3.2.3)}$$

$$V_\mathrm{ee} := \frac{1}{2}\sum_{i\neq j} \frac{e^2}{4\pi\epsilon_0}\frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \qquad \text{Eq (3.2.4)}$$

It is worth noting that the formula Eq (3.2.1) can be simplified by applying atomic units[39], which are derived from a dimensionless Schrödinger

$$\left[-\frac{\hbar^2}{2m_e\lambda^2}\nabla^2 - \frac{e^2}{4\pi\epsilon_0 r}\right]\phi = \mathcal{E}\phi \rightarrow \left[-\frac{\hbar^2}{2m_e\lambda^2}\nabla'^2 - \frac{e^2}{4\pi\epsilon_0 r'}\right]\phi' = \mathcal{E}\phi' \qquad \text{Eq (3.2.5)}$$

(where the substitution $x, y, z \longmapsto \lambda x, \lambda y, \lambda z$), which gives the natural requirement

$$\frac{\hbar^2}{m_e\lambda^2} \stackrel{!}{=} \frac{e^2}{4\pi\epsilon_0\lambda} := \mathcal{E}_a. \qquad \text{Eq (3.2.6)}$$

The atomic unit of energy $\mathcal{E}_a$ from previous equation Eq (3.2.6) is referred to as *Hartree*. The unit of length λ called *Bohr* is expressed from Eq (3.2.6) as

$$\lambda := \frac{4\pi\epsilon_0\hbar^2}{m_e e^2} = a_0, \qquad \text{Eq (3.2.7)}$$

[3] The system named T in 3.1.1.

[4] The movement of nuclei may be computed after the electronic wavefunction is found.

where a_0 is the Bohr radius. In the atomic units the equation Eq (3.2.1) takes form

$$H = -\sum_i \frac{1}{2}\nabla_i^2 - \sum_{i,k} \frac{Z_k}{|\boldsymbol{r}_i - \boldsymbol{R}_k|} + \frac{1}{2}\sum_{i \neq j} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|}. \qquad \text{Eq (3.2.8)}$$

Another step in the Hamiltonian construction is the so-called construction of the wavefunction. In this step the set of basis functions is defined, so that it can be used to represent the electronic wavefunction. The limitations of the real computer mean, we require the basis set to be compact and provide accurate description of the modelled system. It needs to be emphasized that the choice of basis has great impact of the overall performance of the VQE or any other used algorithm[6]. The size of the basis set also depends on the size of studied system, i.e. the number of qubits. An example of this step arises for the *ab initio* models. For these systems the atom-centred Gaussian orbitals are used as the basis set. Namely, for the VQE the most researched choice are Slater-Type Orbitals[40]. The functions from category of atomic orbitals are generally defined as weighted sums of Gaussian functions. This definition it ensures the expected radial distribution and long range behaviour are satisfied. The paper [6] gives an example of such function:

Example 3.3 (Minimal STO-3G basis)**.** In the minimal STO-3G basis for each atom the atomic orbital function is given by the following formula:

$$\xi(r) = c_1\gamma_1(r) + c_2\gamma_2(r) + c_3\gamma_3(r), \qquad \text{Eq (3.2.9)}$$

where γ_i represent Gaussian functions, r is the distance between electron and the nucleus of the atom and c_i are the weight parameter, which are fitted for the studied case.

The word minimal used in this paragraph refers to fact that the basis is chosen to be the smallest necessary to describe the relevant orbitals. The minimal basis usually describes orbital for the valence shell of the atom. To enable representations of correlation and study of more complex phenomena, the basis is expanded to contain higher energy atomic orbitals[41]. A great obstacle in research of other possible bases, such as plane wave basis or grid of points in real space, on current NISQ devices is the large number of physical qubits required for their implementation[42].

The process then continues by combination of the non-orthogonal atomic orbitals into orthogonal molecular orbital by the virtue of mean field theory such as Hartree-Fock theory[41]. This theory also give us the means to transform (by rotation) the Hamiltonian matrix elements into new molecular orbital basis and calculate the energy for single-particle molecular orbital[43]. To give more detailed description of this procedure of transition from single-particle basis functions to many-body basis, consider non-interacting Hamiltonian. For this case the Hartree-Fock theory gives a single many-body basis function with optimized orbitals[6]. The wavefunction constructed from product of many-body-functions describes the electrons in our model, thus it has to satisfy the Pauli exclusion principle. From this follows that it, as a function, has to be antisymmetric, if two electrons are exchanged. The antisymmetric property can be introduced into wavefunction in two fundamental ways, via the first or the second quantization formalism.

The first quantization

This formalism relies on the device of Slater determinants describing the many-body functions.

Definition 3.4 (Slater determinant)**.** Let $\phi_i(\boldsymbol{x_i})$ stand for a spin-orbital (function) of the chosen basis, where the variable $\boldsymbol{x}_i := (\boldsymbol{r}_i, \sigma_i)$ contains the spatial position and spin of the i-th electron. We define the Slater determinant as a representation of wavefunction of n occupied orbitals by the following formula

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n,) := \frac{1}{\sqrt{n!}} \begin{vmatrix} \phi_1(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_1) & \cdots & \phi_n(\boldsymbol{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \phi_2(\boldsymbol{x}_2) & \cdots & \phi_n(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_n) & \phi_2(\boldsymbol{x}_n) & \cdots & \phi_n(\boldsymbol{x}_n) \end{vmatrix}, \qquad \text{Eq (3.2.10)}$$

which denotes the many-body basis functions described above. With the Slater determinant we can introduce the *Fock space*, which is a abstract linear vector space with inner product. Thus, Fock space is a Hilbert space, consisting of so-called occupation number (ON) vectors representing the Slanter determinants through a map.

$$|\boldsymbol{k}\rangle := |k_1, k_2, \ldots, k_n\rangle\,, \quad k_i = \begin{cases} 1 & \phi_i(\boldsymbol{x}) \quad \text{is occupied} \\ 0 & \phi_i(\boldsymbol{x}) \quad \text{is unoccupied} \end{cases}, \quad \langle \boldsymbol{k}|\boldsymbol{m}\rangle := \prod_{P=1}^{n} \delta_{k_P m_P} \qquad \text{Eq (3.2.11)}$$

Note that for a system without electrons, the *vacuum state* is defined $|\text{vac}\rangle$. This map enables us to work with Slater determinant as with vectors. The vector representation has the advantage of being easy to realize on a quantum computer, e.g. the occupied spin-orbital can be denoted by an up-spin and similarly the unoccupied by a down-spin on a spin based quantum computer described in [4].

To keep the notation simple and more illustrative, we deviate from the traditional notation by setting

$$|\phi_1, \phi_2, \ldots, \phi_n\rangle := |k_1, k_2, \ldots, k_n\rangle\,. \qquad \text{Eq (3.2.12)}$$

In order to preserve the antisymmetry of the Slater determinant in the occupation number representation by defining

$$|\sigma(\phi_1, \phi_2, \ldots, \phi_n)\rangle := (-1)^{\pi(\sigma)}\, |\phi_1, \phi_2, \ldots, \phi_n\rangle\,, \qquad \text{Eq (3.2.13)}$$

where $\pi(\sigma)$ denotes parity of the permutation σ of the basis functions. Hence, the antisymmetry is addressed in the wavefunction, this is the main difference from the second quantization. Finally, the Hamiltonian is constructed in terms of the wavefunctions by the means of projection. For one-electron the resulting formulas are

$$\begin{aligned} h_{pq} &= \langle \phi_p | T_\text{e} + V_\text{ne} | \phi_q \rangle && \text{Eq (3.2.14)} \\ &= \int \mathrm{d}\boldsymbol{x} \phi_p^*(\boldsymbol{x}) \left(-\frac{\hbar^2}{2M_i} \nabla_i^2 - \frac{e^2}{4\pi\epsilon_0} \sum_k \frac{Z_k}{|\boldsymbol{r} - \boldsymbol{R}_k|} \right) \phi_q(\boldsymbol{x}) && \text{Eq (3.2.15)} \\ h_{pqrs} &= \langle \phi_p \phi_q | V_\text{ee} | \phi_r \phi_s \rangle && \text{Eq (3.2.16)} \\ &= \frac{e^2}{4\pi\epsilon_0} \int \mathrm{d}\boldsymbol{x}_1 \mathrm{d}\boldsymbol{x}_2 \frac{\phi_p^*(\boldsymbol{x}_1) \phi_q^*(\boldsymbol{x}_2) \phi_r(\boldsymbol{x}_2) \phi_s(\boldsymbol{x}_1)}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|}. && \text{Eq (3.2.17)} \end{aligned}$$

Hence, the Hamiltonian in the single-particle basis is expressed as

$$H = \sum_{i=1}^{m} \sum_{p,q=1}^{n} h_{pq} \left| \phi_p^{(i)} \right\rangle \left\langle \phi_q^{(i)} \right| + \frac{1}{2} \sum_{i \neq j}^{m} \sum_{p,q,r,s=1}^{n} h_{pqrs} \left| \phi_p^{(i)} \phi_q^{(j)} \right\rangle \left\langle \phi_r^{(i)} \phi_s^{(j)} \right| \qquad \text{Eq (3.2.18)}$$

To summarize the paragraphs above, in the general case after choosing the model it is necessary to set the parameters of the model according to the chosen model and its geometry. In order to construct the Hamiltonian for T, we first need to find the set of basis functions for the relevant problem, which

represent individual degrees of freedom for single particles. The Hamiltonian is composed of specific operators and their weights found from the selected basis. The possibility of different constructions follows from the fact that single-particle unitary transformations applied to the chosen basis functions preserve the ground-state energy. This implies that the individual degrees of freedom can be rotated while keeping total energy unchanged.

Now the question of applying the results of the first quantization theory to quantum computer can be addressed by means of method from paper [44]. Let n be the number of functions on the single-particle basis and m denote the number of electrons in the modelled wavefunction. We assume that one electron can occupy at most one function from basis. Now we apply a map to represent basis functions with qubits:

$$\phi_k(\boldsymbol{x}_k) \longmapsto |\phi_k\rangle = |(k)_2\rangle\,, \quad \forall k \in \{1,\ldots,n\}, \qquad \text{Eq (3.2.19)}$$

where $(k)_2$ is the big endian binary representation of the number k, e.g. $\phi_0(\boldsymbol{x}_0) \longmapsto |\phi_0\rangle = |00\ldots00\rangle$ and $\phi_1(\boldsymbol{x}_1) \longmapsto |\phi_1\rangle = |10\ldots00\rangle$. Thus, the basis function is represented by $\log_2(n)$ qubits. From this follows that the product state for m electrons takes $N := m \cdot \log_2(n)$ qubits. The paper [5] details the procedure to maintain the antisymmetry requirement for first quantization using $O(m\log_2(n))$ ancillary qubits and circuit depth bound as $O(\log_2^c(m)\log_2(\log_2(n)))$. The antisymmetrization algorithm can be summarized in the following way.

Example 3.5 (Antisymmetrization algorithm outline)**.** [5] Consider a quantum computer with the following registers:

- T - standing for target, contains sorted repetition-free quantum array
- S - standing for seed, ancillary register, is prepared according to function f defined bellow
- R - standing for record, ancillary register, initialization depends on the implementation details, e.g. $|\mathbf{0}\rangle$

Let η denote the number of particles, $f : \mathbb{R} \mapsto \mathbb{R}$ be a function satisfying the relation $f(\eta) \geq \eta^2, \forall\eta$. The antisymmetrization algorithm outlined bellow performs the following transformation:

$$T: \quad |r_1\ldots r_\eta\rangle \longmapsto \sum_{\sigma\in S_\eta} (-1)^{\pi(\sigma)} |\sigma(r_1,\ldots,r_\eta)\rangle\,, \qquad \text{Eq (3.2.20)}$$

where $\pi(\sigma)$ denotes the parity of the permutation σ.

1. S is prepared in an even superposition of states corresponding to all strings of numbers from $\{0,1,\ldots,f(\eta)-1\}$ of length η. For $f(\eta) := 2^\eta$ this is equivalent to applying Hadamard gate to every qubit.

$$S: \frac{1}{f(\eta)^{\eta/2}} \sum_{\boldsymbol{l}=\mathbf{0}}^{f(\eta)-1} |l_0, l_2, \ldots, l_{\eta-1}\rangle \qquad \text{Eq (3.2.21)}$$

2. S is sorted using a reversible sorting network which is storing its comparator output to register R. Thus the state before the can be written as

$$\sum_{0\leq l_0\leq\cdots\leq l_{\eta-1}\leq f(\eta)} \sum_{\sigma\in S_\eta} |\sigma(l_0, l_2, \ldots, l_{\eta-1})\rangle_S\, |\iota\rangle_R\,, \qquad \text{Eq (3.2.22)}$$

where ι denotes the identity permutation[5]. The state after the sorting network is applied is

$$\sum_{0\leq l_0\leq\cdots\leq l_{\eta-1}\leq f(\eta)} |l_0, l_2, \ldots, l_{\eta-1}\rangle_S \sum_{\sigma\in S_\eta} |\sigma_1, \ldots, \sigma_t\rangle_R \,, \qquad \text{Eq (3.2.23)}$$

Thus the resulting state after sorting is a product state, the register S can be discarded after the following step.

3. Collisions, i.e. strings with duplicate entries, are deleted from seed. This is performed via measurement of seed and accepting only the repetition free results.

4. Reverse sort is applied to T according to record in R. This step generates the desired antisymmetrized state.

The second step relies upon the theory of quantum sorting. There are various sorting algorithms, which can be implemented on quantum computers, for example heapsort or bubble sort have quantum versions. However, the complexity of the quantum is not the same as on conventional computer. The quantum heapsort needs $O(\eta^2)$ operations due to overhead of indexing elements, unlike the classical which has only $O(\eta)$ operations. The problems of quantum versions of classical algorithms lead to sorting networks being preferred. The sorting networks are a category of sorting algorithms which place comparisons and swaps on a set of fixed locations. An example of network sorting algorithms are bitonic sort algorithms, which exist for both classical and quantum computers. The sorting networks have a primary functional element, the comparator. The comparator component sorts two inputs like its classical analogy. For two numbers $x, y \in \mathbb{R}$ a classical comparator performs operation $(x, y) \mapsto \big(\min(x, y), \max(x, y)\big)$. The *quantum comparator* is a reversible comparator operating on a quantum register, which records the result of comparison and performs conditional swap. The broad topic of sorting will not be further

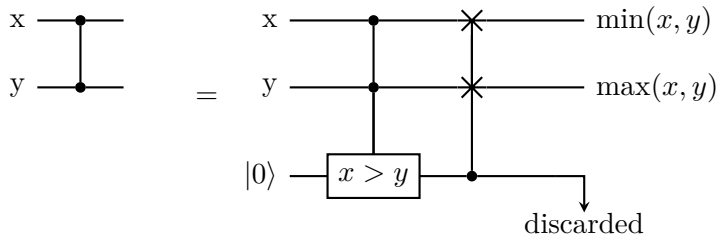


Figure 3.2: Quantum reversible comparator, recording the result to ancillary qubit[5]

discussed in this project.

In the following step the Hamiltonian is translated from $\left|\phi_p^{(i)}\right\rangle\!\left\langle\phi_q^{(i)}\right|$ to directly measurable operators by expressing the operators as tensor products of single qubits operators $|0\rangle\langle 0|, |0\rangle\langle 1|, |1\rangle\langle 0|, |1\rangle\langle 1|$, and mapping to Pauli operators using the identities from Eq (1.1.6) from Chapter 1. To quantify the performance of this approach we shall give an upper bound for the number of operators in Pauli strings expressing the Hamiltonian. The dominant part of the Hamiltonian are the two-body terms in the second sum. Their number in equation Eq (3.2.18) scales as $O(n^4m^2)$ (for the sum terms there are four spin-orbitals and two electrons). A representation of a spin-orbital takes $\log_2(n)$ qubits, so there are total of $2\log_2(n)$ tensored qubit outer products, where each outer product is decomposed into a sum of two Pauli matrices. In total we get a sum of $2^{2\log_2(n)} = n^2$ Pauli strings. Thus, the Pauli string s scale as $O(n^6m^2)$. To conclude the topic of first quantization formalism for VQE we discuss the

[5]The R register holds the encoded record of performed permutations. Thus identity permutation is the initial value representing that no sorting was performed.

significance of the first quantization in context of the NISQ computers. The current research of this method shows [45] that while the first quantization has high resource demands for number of qubits including the ancillary qubits and requires computation of spatial integrals (Eq (3.2.14)). However, the first quantization can offer advantage for special cases of models with large basis. This case appears for particular choice of basis functions in certain quantum chemistry problems explored in the article[42].

The second quantization

The second quantization approach to antisymmetry differs from first quantization in where the antisymmetry is encoded. The antisymmetry is enforced through the operators used to express the Hamiltonian and their properties. This formalism allows construction of all operators and states using the *fermionic* operators described in [41].

Definition 3.6 (Creation operators)**.** Consider a Hilbert space (most notably the Fock space) with dimension M containing occupation number vectors. Then M elementary *creation operators* $a_i^\dagger$ are defined by their actions:

$$a_i^\dagger \left|k_1, k_2, \ldots, 0_i, \ldots, k_M\right\rangle = \Gamma_i^{\boldsymbol{k}} \left|k_1, k_2, \ldots, 1_i, \ldots, k_M\right\rangle, \qquad \text{Eq (3.2.24)}$$

$$a_i^\dagger \left|k_1, k_2, \ldots, 1_i, \ldots, k_M\right\rangle = 0, \qquad \text{Eq (3.2.25)}$$

where symbol $\Gamma_i^{\boldsymbol{k}}$ denotes

$$\Gamma_i^{\boldsymbol{k}} := \prod_{j=i}^{i-1} (-1)^{k_j}, \quad \Gamma_i^{\boldsymbol{k}} = \begin{cases} +1 & \text{if \#(electrons in spin-orbitals } j < i) \text{ is even} \\ -1 & \text{if \#(electrons in spin-orbitals } j < i) \text{ is odd} \end{cases} \qquad \text{Eq (3.2.26)}$$

It can be seen that the creation operator action is equivalent to adding an electron to an orbital specified by the creation operator indices. Thus, it has some properties of the Slater determinant, in terms of the antisymmetry and being zero, if an element is repeated. From the meaning of the operator follows a useful notation

$$\left|\boldsymbol{k}\right\rangle = \left[\prod_{i=1}^{M} \left(a_i^\dagger\right)^{k_i}\right] \left|\text{vac}\right\rangle. \qquad \text{Eq (3.2.27)}$$

The advantage of this notation becomes clear from formulas below, which follow from the definition 3.6.

$$a_i^\dagger \left|\boldsymbol{k}\right\rangle = \delta_{k_i 0} \Gamma_i^{\boldsymbol{k}} \left|k_1, \ldots, 1_i, \ldots, k_M\right\rangle \qquad \text{Eq (3.2.28)}$$

$$a_i^\dagger a_i^\dagger \left|\boldsymbol{k}\right\rangle = a_i^\dagger \delta_{k_i 0} \Gamma_i^{\boldsymbol{k}} \left|k_1, \ldots, 1_i, \ldots, k_M\right\rangle = 0 \implies a_i^\dagger a_i^\dagger = 0 \qquad \text{Eq (3.2.29)}$$

$$a_i^\dagger a_j^\dagger \left|\ldots, k_i, \ldots, k_j, \ldots\right\rangle = \delta_{k_i 0} \delta_{k_j 0} \Gamma_i^{\boldsymbol{k}} \Gamma_j^{\boldsymbol{k}} \left|\ldots, k_i, \ldots, k_j, \ldots\right\rangle \qquad \text{Eq (3.2.30)}$$

$$a_j^\dagger a_i^\dagger \left|\ldots, k_i, \ldots, k_j, \ldots\right\rangle = \delta_{k_i 0} \delta_{k_j 0} \Gamma_i^{\boldsymbol{k}} (-\Gamma_j^{\boldsymbol{k}}) \left|\ldots, k_i, \ldots, k_j, \ldots\right\rangle \qquad \text{Eq (3.2.31)}$$

The summation of Eq (3.2.30) and Eq (3.2.31) gives

$$\left(a_i^\dagger a_j^\dagger + a_j^\dagger a_i^\dagger\right) \left|\boldsymbol{k}\right\rangle = 0 \implies \left\{a_i^\dagger, a_j^\dagger\right\} = 0, \qquad \text{Eq (3.2.32)}$$

where the implication follows from the fact that the first part hold even for $i > j$ from substitution of indices in Eq (3.2.30) and Eq (3.2.31). The other new operator is the conjugate of the creation operator, the annihilation operator. Its action and properties follow from the creation operator. Adjoining Eq (3.2.32) we get

$$a_i a_j + a_j a_i = 0. \qquad \text{Eq (3.2.33)}$$

To find the action the matrix element expressed

$$\langle \boldsymbol{m}|a_i|\boldsymbol{k}\rangle = \langle \boldsymbol{k}|a_i^\dagger|\boldsymbol{m}\rangle = \begin{cases} \delta_{m_i 0}\Gamma_i^{\boldsymbol{m}} & \text{for } k_i = m_i + \delta_{ij} \\ 0 & \text{otherwise} \end{cases} \quad \text{Eq (3.2.34)}$$

which can be simplified by realizing that $\Gamma_i^{\boldsymbol{m}} = \Gamma_i^{\boldsymbol{k}}$, $\langle \boldsymbol{m}|a_i|\boldsymbol{k}\rangle = \delta_{k_i 1}\Gamma_i^{\boldsymbol{k}}$, if $m_i = k_i - \delta_{ji}$ and 0 otherwise. Hence, by plugging in we get the definitional equation.

Definition 3.7 (Annihilation operators)**.** Consider the same Hilbert space as in the definition of the creation operator 3.6. The *annihilation operator* a_i is defined as a conjugate operator to the creation operator $a_i^\dagger$ with action

$$a_i\,|\boldsymbol{k}\rangle = \delta_{k_i 1}\Gamma_i^{\boldsymbol{k}}\,|k_1, \ldots, 0_i, \ldots, k_M\rangle \quad \text{Eq (3.2.35)}$$

From the definitions 3.6 and 3.7 follows the anticommutation relation

$$\{a_i, a_j^\dagger\} = \delta_{ij} \quad \text{Eq (3.2.36)}$$

Further on in this project the simplified notation utilizing the basis functions rather than the ON vectors, similar to the one used for the first quantization, will be used.

$$a_i^\dagger\,|\boldsymbol{\phi}\rangle = a_i^\dagger\,|\phi_1\phi_2\ldots\rangle = \delta_{\phi_i 0}\Gamma_i\,|\phi_1\phi_2\ldots 1_i\ldots\rangle \quad \text{Eq (3.2.37)}$$

$$a_i\,|\boldsymbol{\phi}\rangle = a_i\,|\phi_1\phi_2\ldots\rangle = \delta_{\phi_i 1}\Gamma_i\,|\phi_1\phi_2\ldots 0_i\ldots\rangle \quad \text{Eq (3.2.38)}$$

Using this notation we rewrite the equation Eq (3.2.18) as

$$H = \sum_{pq} h_{pq}a_p^\dagger a_q + \frac{1}{2}\sum_{pqrs} h_{pqrs}a_p^\dagger a_q^\dagger a_r a_s. \quad \text{Eq (3.2.39)}$$

To implement the fermionic operators in a quantum computer they need to be encoded into the measurable Pauli operators or more precisely to Pauli strings. This topic, in its entirety, is extensive and not related to the problem of Hamiltonian, so it is considered a separate component of the VQE stack and is included in the section below.

3.2.2 Encoding of fermionic operators

As it was mentioned in the preceding section the creation and annihilation operator representation of the Hamiltonian has to be encoded into the Pauli strings. The transformation has to preserve the antisymmetry of the operators. The problem is much like the encoding for the first quantization. Because the second quantization approach is considered more perspective for the NISQ computers, currently there are more transformations than this project can explore. We restrict ourselves to the selected transformations relevant to general case and discussed implementations. The theory of the transformations of the fermionic space to spin space is given in greater detail in the publication [6].

The considered transformations are the maps $\mathcal{T} : \mathcal{F}_n \mapsto (\mathbb{C}^2)^{\otimes N}$, where $\mathcal{F}_n$ denotes the Fock states of n orbitals, i.e. the ground states outlined in 3.2.1 and studied in detail in[39], and $(\mathbb{C}^2)^{\otimes N}$ stand for the Hilbert space of the operators acting on N qubits. The transformations are usually divided into generalized and special. The former are concerned with transformations of the whole Fock space $\mathcal{F}_n$ and are significant for the *ab initio* models. The latter are adjusted for special cases and their geometry. The special will not be discussed in this project as stated in the first paragraph. In [6] a set of three characteristics was suggested to compare the studied transformations.

- N_Q — denotes the number of qubits used to represent the wavefunction; N_Q depends on the number of modelled spin-orbitals; it is desirable to decrease N_Q as much as possible by concentrating the contained information as suggested in [46]

- W_P — defined in 1.5; lower W_P enables increased parallelization, increases the resilience to so-called barren plateau problem of the optimization algorithms and to some types of errors [12, 47]; the effects of W_P for different encodings were benchmarked in [23].

- N_P — stands for the number of Pauli strings, in [6] it was shown that N_P scales as $O(n^4)$ for the molecular Hamiltonian.

Method	W_P	Description
Jordan-Wigner	$O(n)$	Most commonly used method. Encodes orbital occupation directly and locally onto qubits.
Parity	$O(n)$	Encodes orbital parity directly and locally onto qubits.
Bravyi-Kitaev	$O(\log_2(n))$	Minimizes W_P by mixing occupation and parity encodings. Usually gives lower circuit depth, but not higher noise resilience.
Optimal general encoding on ternary trees	$O(\log_3(2n))$	Reaches optimal W_P asymptotically. Currently, not benchmarked.

Table 3.1: Comparison of fermionic operator encoding methods, abbreviated from [6]

Jordan-Wigner method

To encode the fermionic operators, so that $a_j^\dagger |0\rangle_j = |1\rangle_j$ and $a_j |1\rangle_j = |0\rangle_j$, a method inspired by the Jordan-Wigner transformation presented in 1928 may be applied. The Jordan-Wigner method maps the occupation number of n spin-orbitals to qubits by setting $|0\rangle_j$, if j-th orbital (ϕ_j) is unoccupied, and $|1\rangle_j$ otherwise. The indexing j merges the spatial and spin orbitals together. The original transformation was solving the opposite problem, so the Jordan-Wigner encoding is essentially inverse of this transformation. The basic ideal is to perform map

$$a_j^\dagger \to |1\rangle\langle 0|_j = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \frac{X_j - \mathrm{i}Y_j}{2}, \qquad a_j \to |0\rangle\langle 1|_j = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \frac{X_j + \mathrm{i}Y_j}{2}, \qquad \text{Eq (3.2.40)}$$

where the outer products are rewritten to Pauli operators according to Eq (1.1.6). This simple mapping has to be modified in order to reintroduce the antisymmetry. A string of tensor products $Z_0 \otimes \cdots Z_{j-1}$ is added by a tensor product to the representation Eq (3.2.40). The added string has an eigenvalue $+1$ and -1 for even and odd number of occupied orbitals respectively. Thus, the fermionic sign prescription is present. The anticommutation relations Eq (3.2.32), Eq (3.2.33) and Eq (3.2.36) follow from the antisymmetries $XZ = -ZX$, $YZ = -ZY$ and the definition

Definition 3.8 (Jordan-Wigner encoding for the fermionic operators)**.** For the fermionic operator the

Jordan-Wigner mapping is defined as

$$a_1/a_1^\dagger = \left(\frac{X \pm \mathrm{i}Y}{2}\right) \otimes I \otimes I \otimes I \cdots \otimes I \qquad \text{Eq (3.2.41)}$$

$$a_2/a_2^\dagger = Z \otimes \left(\frac{X \pm \mathrm{i}Y}{2}\right) \otimes I \otimes I \cdots \otimes I \qquad \text{Eq (3.2.42)}$$

$$a_3/a_3^\dagger = Z \otimes Z \otimes \left(\frac{X \pm \mathrm{i}Y}{2}\right) \otimes I \cdots \otimes I \qquad \text{Eq (3.2.43)}$$

$$a_j/a_j^\dagger = Z_0 \otimes \cdots Z_{j-1} \otimes \left(\frac{X \pm \mathrm{i}Y}{2}\right) \qquad \text{Eq (3.2.44)}$$

Apart from the notation used in Eq (3.2.41), some literature[38] introduces more compact symbolism for the fermionic operators

$$a_p^\dagger = \left(\prod_{m<p} \sigma_m^Z\right) \sigma_p^+, \quad a_p = \left(\prod_{m<p} \sigma_m^Z\right) \sigma_p^-, \quad \sigma^\pm := \frac{\sigma^X \mp \mathrm{i}\sigma^Y}{2} \qquad \text{Eq (3.2.45)}$$

The definition 3.8 directly gives the Pauli weight $W_P = O(N)$ as the result of adding the Z strings. The map creates two Pauli string for each fermionic operator and since the number of fermionic operators scales as $O(n^4)$, due to the two-body terms, we conclude that $N_P = O(N^4)$. There exist methods of lowering the W_P, such as the spin operator level parameters described in [48].

Parity encoding

In parity encoding method, the meaning of the qubit register is different from that in section 3.2.2. The encoding principle is explained in the definition 3.9 below.

Definition 3.9 (Parity encoding of state)**.** The states are encoded as

$$|0\rangle_j - \text{if the number of occupied orbitals with indices } \{0, 1, \ldots, j\} \text{ is even,} \qquad \text{Eq (3.2.46)}$$

$$|1\rangle_j - \text{if the number of occupied orbitals with indices } \{0, 1, \ldots, j\} \text{ is odd.} \qquad \text{Eq (3.2.47)}$$

This can be described for a fermionic state $|\phi_0\phi_1 \ldots \phi_n\rangle$ and qubit register state $|p_0p_1 \ldots p_n\rangle$ using modular summation

$$p_i = \sum_{j \le i} \phi_j \pmod 2 = \sum_j [\pi_n]_{ij} \phi_j \pmod 2 \iff |\boldsymbol{p}\rangle = |\pi_n(\boldsymbol{\phi})\rangle \pmod 2, \qquad \text{Eq (3.2.48)}$$

where $[\pi_n]_{ij}$ is an element of $n \times n$ matrix

$$\pi_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}. \qquad \text{Eq (3.2.49)}$$

From the definition follows that the occupation of orbital j appears as parity change, i.e. $|0\rangle_{j-1} \to |1\rangle_j$ or $|1\rangle_{j-1} \to |0\rangle_j$. This simplifies the encoding of the fermionic operators. For an isolated fermionic

operator trivially follows the desired form of encoding.

$$\begin{aligned} a_j^\dagger &\to |01\rangle\langle 00|_{j-1,j} - |10\rangle\langle 11|_{j-1,j} = \frac{Z_{j-1}\otimes X_j - \mathrm{i}Y_j}{2}, \\ a_j &\to |00\rangle\langle 01|_{j-1,j} - |11\rangle\langle 10|_{j-1,j} = \frac{Z_{j-1}\otimes X_j + \mathrm{i}Y_j}{2} \end{aligned} \qquad \text{Eq (3.2.50)}$$

For more qubits the idea has to be "fixed" by flipping the $|\boldsymbol{p}\rangle$ register accordingly by adding the X string $X_{j+1}\otimes\cdots\otimes X_{n-1}$.

Definition 3.10 (Parity encoding of the fermionic operators)**.** For the parity encoding method we define the fermionic operator representation as

$$a_j/a_j^\dagger \to \frac{Z_{j-1}\otimes X_j \pm \mathrm{i}Y_j}{2}\otimes X_{j+1}\otimes\cdots\otimes X_{n-1} \qquad \text{Eq (3.2.51)}$$

The formulas Eq (3.2.51) lead to same results as for Jordan-Wigner mapping: $W_P = O(N), \quad N_P = O(N^4)$.

Bravyi-Kitaev encoding

The last method reviewed in this project was presented in [46] and was further describe for the relevant electron problem in [49]. The foundation of the transformation is the use of β_y blocks with similar role to π_n matrix Eq (3.2.49). The indices $y = 2^x$ and blocks have size of β_{2^x} is $2^x \times 2^x$ and are defined recursively. Naturally the size of final block is $N \times N$. Thus, recursion stop when $x \geq \log_2(N)$ and the additional rows and columns can be discarded. The recursion ca be described as

$$\begin{aligned} \beta_1 &:= \begin{bmatrix}1\end{bmatrix}, \quad \beta_{2^x} := \begin{pmatrix}\beta_{2^{x-1}} & \mathbb{O} \\ \mathbb{M} & \beta_{2^{x-1}}\end{pmatrix} \\ \mathbb{O} &:= \begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}_{2^{x-1}\times 2^{x-1}}, \quad \mathbb{M} := \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 1 & 1 & 1\end{pmatrix}_{2^{x-1}\times 2^{x-1}} \end{aligned} \qquad \text{Eq (3.2.52)}$$

The resulting matrix for $x = 3$ is

$$\beta_8 = \left(\begin{array}{cccc|cccc} 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ \hline 0&0&0&0&1&0&0&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&1&0\\ 1&1&1&1&1&1&1&1 \end{array}\right) \qquad \text{Eq (3.2.53)}$$

By comparison to the methods 3.2.2 and 3.2.2 it can be seen the j-th qubit in register can hold two types of information base on the index:

- for j even: occupancy

- for j odd: occupancy and parity

and the parity information has two formats

- for $j \neq 2^k - 1$: parity of orbitals with indices $< j$ in the block
- for $j = 2^k - 1$: parity of orbitals with indices $\leq j$ in the entire set.

Based on this the qubits in the register are divided into four sets, for each the fermionic operator are encoded differently. With this definition we can proceed to the fermionic operator mapping for

Name of the set	Information	Indices/Set	Comment
Update $U(j)$	Parity of orbital j	Odd	Qubits that need to be updated, corresponds to jth column in β_y excluding qubit j
Parity $P(j)$	Parity, fermionic sign	Even and Odd	Parity of orbitals with indices in $< j$.
Flip $F(j)$	p_j is equal/opposite of ϕ_j	$F(j) \subset P(j)$	$F(j) = \emptyset$, if j is even.
Remainder $R(j)$	Other	$R(j) = P(j) \setminus F(j)$	

Table 3.2: Groups of qubits in Bravyi-Kitaev encoding, [6, 49]

individual groups of qubits.

- Occupancy qubits (j is even): For this group of qubits the form of operators follows from the principles of Jordan-Wigner encoding. It can be seen that qubits with indices $> j$ which are affected, that is the $U(j)$ set, need to be flipped using X gates. The sign is determined by the qubits with preceding indices, i.e. the $P(j)$ set, by Z gates.

$$a_j^\dagger \to Z_{P(j)} \otimes |1\rangle\langle 0|_j \otimes X_{U(j)} = \frac{1}{2} Z_{P(j)} \otimes (X_j - \mathrm{i}Y_j) \otimes X_{U(j)} \qquad \text{Eq (3.2.54)}$$

$$a_j \to Z_{P(j)} \otimes |0\rangle\langle 1|_j \otimes X_{U(j)} = \frac{1}{2} Z_{P(j)} \otimes (X_j + \mathrm{i}Y_j) \otimes X_{U(j)} \qquad \text{Eq (3.2.55)}$$

- Parity qubits[6] (j is odd): The encoding of the operators has to account for the meaning of the qubits, e.g. a creation operator can change $|1\rangle$ to $|0\rangle$ depending on parity for indices $< j$. For example if parity 1 for indices $< j$ and qubits j is $|0\rangle$, then orbital j is occupied when parity flips. The two possible cases are
  - number of $|1\rangle$ in $F(j)$ is even - qubits j is equal to jth orbital occupancy,
  - number of $|1\rangle$ in $F(j)$ is odd - qubits j is opposite to jth orbital occupancy.

This problem is solved by defining two projectors onto both states of $F(j)$

$$E_{F(j)} = \frac{1}{2}(I^{\otimes N} + Z_{F(j)}) \qquad \text{Projector onto even states of } F(j) \qquad \text{Eq (3.2.56)}$$

$$O_{F(j)} = \frac{1}{2}(I^{\otimes N} - Z_{F(j)}) \qquad \text{Projector onto odd states of } F(j) \qquad \text{Eq (3.2.57)}$$

Eq (3.2.58)

[6] This name of group does not refer to parity set $P(j)$ form table 3.2

Following the procedure described in [6], for isolated qubits we get encoding

$$a_j^\dagger \to E_{F(j)} \otimes |1\rangle\langle 0|_j + O_{F(j)} \otimes |0\rangle\langle 1|_j = \frac{X_j - \mathrm{i} Z_{F(j)} \otimes Y_j}{2} \quad \text{Eq (3.2.59)}$$

$$a_j \to E_{F(j)} \otimes |0\rangle\langle 1|_j + O_{F(j)} \otimes |1\rangle\langle 0|_j = \frac{X_j + \mathrm{i} Z_{F(j)} \otimes Y_j}{2} \quad \text{Eq (3.2.60)}$$

The following correction by adding flips by X and sign by Z gates, is simplified by the trivial identity $Z^2 = 1$, from which follows that $Z_{F(j)} \otimes Z_{P(j)} = Z_{P(j)\setminus F(j)} = Z_{R(j)}$. Hence, we get the encoding

$$a_j^\dagger \to \frac{Z_{P(j)} \otimes X_j \otimes X_{U(j)} - \mathrm{i} Z_{R(j)} \otimes Y_j \otimes Z X_{U(j)}}{2} \quad \text{Eq (3.2.61)}$$

$$a_j \to \frac{Z_{P(j)} \otimes X_j \otimes X_{U(j)} + \mathrm{i} Z_{R(j)} \otimes Y_j \otimes Z X_{U(j)}}{2} \quad \text{Eq (3.2.62)}$$

To conclude the topic of Hamiltonian construction we add the following figure which summarizes the process.

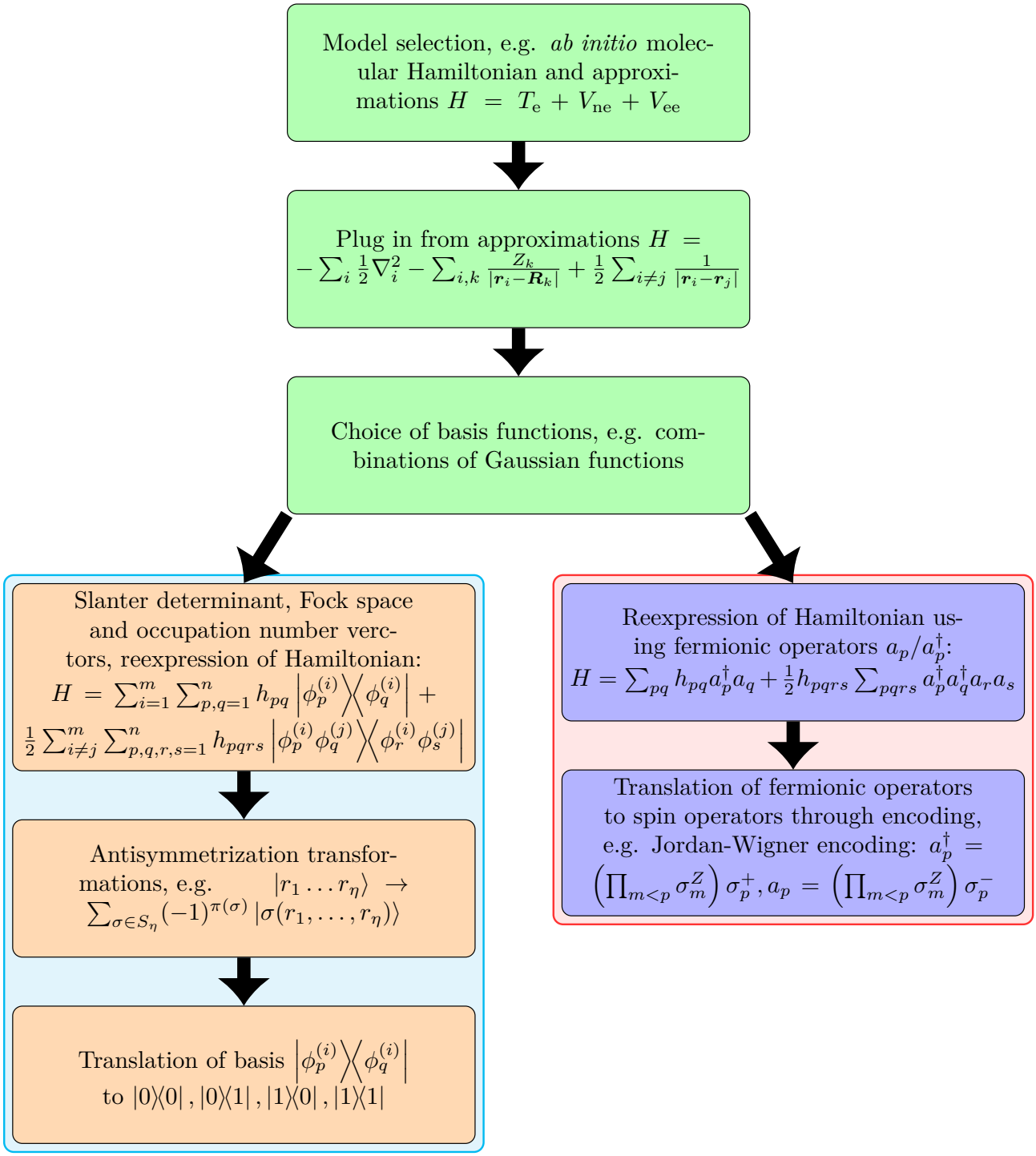


Figure 3.3: Schematic of the process of VQE Hamiltonian construction described in 3.2.1, where the blocks with green ( ) background are common and the blocks inside cyan ( ) region are the first quantization method and blocks inside red ( ) region are the second quantization.

3.2.3 Measurement optimizations

The measurement in VQE can be performed in various ways, which influence overall performance of algorithm. The objective of decreasing the number of needed circuit repetitions[7] gives rise to so-called *measurement strategies*. The strategies lower the number of repetition by the following groups of methods[6]

1. Weighting of the number measurement for operators
2. Searching commuting groups of operators inside the Pauli string and finding measurement basis for simultaneous measurement of the groups, before the measurement the groups have to be rotated into the selected basis
3. Using interference

Weighting methods

The principle of the method follows from observing, how number of *shots* scales. For a quantum circuit as a sampling experiment we have the *standard error*

$$\epsilon = \frac{\sigma}{\sqrt{S}}, \qquad \text{Eq (3.2.63)}$$

where σ denotes the *population standard deviation* and S stands for the *sample size*, i.e. number of shots. From this follows that in general case $O(1/\epsilon^2)$ shots are required for precision of ϵ of expectation value of an operator. This simple case is modified for Pauli strings.

Now consider a system with N_P Pauli strings, where the Pauli string P_a has Pauli weight w_a defined in 1.5. In [38] it was shown that the optimal distribution of measurements among Pauli strings satisfies

$$S \leq \left(\frac{\sum_a^{N_P} w_a}{\epsilon}\right), S = O\left(\frac{N^4}{\epsilon^2}\right). \qquad \text{Eq (3.2.64)}$$

To derive the method, the formula for standard error is rewritten using variance for Pauli strings P_a. The variance for a Pauli string can be simplified, because for Pauli string P_a we have $P_a^2 = I$,

$$\mathrm{Var}(P_a) = \langle\Psi|P_a^2|\Psi\rangle - \langle\Psi|P_a|\Psi\rangle^2 = 1 - \langle\Psi|P_a|\Psi\rangle^2 \leq 1 \qquad \text{Eq (3.2.65)}$$

Now we consider that we make a different number of measurements for individual Pauli strings. Thus, we have formula

$$\epsilon = \sqrt{\sum_{a=1}^{N_P} \frac{w_a^2 \mathrm{Var}(P_a)}{S_a}} = \sqrt{\sum_a^{N_P} w_a^2 \frac{1 - \langle\Psi|P_a|\Psi\rangle^2}{S_a}}, \qquad \text{Eq (3.2.66)}$$

where S_a is defined as the number of shots used for measurement expectation value of P_a. From this follows that $S = \sum_a S_a$. The formula can be simplified further for uniformly distributed measurements,

[7] Sometimes referred to as *shots* [6]

i.e. $S_a = S/N_P$.

$$S = N_p \sum_{a=1}^{N_p} \frac{w_a^2 \mathrm{Var}(P_a)}{\epsilon^2}. \qquad \text{Eq (3.2.67)}$$

From this easily follows that for Pauli weights, such that $1/w_a$ is proportional to $\sqrt{\mathrm{Var}(P_a)}$, we get the minimal number of shots S.

Instead of setting the same number of shots for all operators, the distribution of repetitions may favour the operators with greater impact on variance. In case the number of shots is not strictly limited we consider the number of repetitions proportional to the weight. From Eq (3.2.66) it can be seen that, if $\mathrm{Var}(P_a) \approx V, \forall a$ this setting decreases total variance. The optimal distributions of repetitions is $S_a \propto w_a \sqrt{\mathrm{Var}(P_a)}$ or in some case more simplistically $S_a \propto w_a$. For cases with highly limited number of shots, it is suggested to use random measurements with probabilities proportional to $w_a \sqrt{\mathrm{Var}(P_a)}$.[6]

Truncation

Another method used to increase speed up calculations is the truncation of terms with the least impact on the result. Since the calculation is based on sum of weighted Pauli strings, it can be seen that in order to quantify the significance of terms of the sum, the following trivial inequality can be used

$$|\langle\Psi|w_a P_a|\Psi\rangle| \leq w_a. \qquad \text{Eq (3.2.68)}$$

Next we define

$$s_k := \sum_{a=1}^{k} w_a, \quad k \leq N_P; \quad w_a \leq w_{a+1}. \qquad \text{Eq (3.2.69)}$$

Finally, by choosing a constant $C \in [0, 1)$, we now omit terms with indices $> l$ in Eq (3.2.67), where l satisfies $s_l < C \cdot \epsilon$. For this method it was shown that the total number of shot is then

$$S = (N_P - l) \sum_{a=l+1}^{N_P} \frac{w_a^2 \mathrm{Var}(P_a)}{(1 - C^2)\epsilon^2} \qquad \text{Eq (3.2.70)}$$

Pauli string grouping

The theory of Pauli string grouping is extensive and in this project only its basics will be outlined. The basic idea of this method is the fact that by measuring a Pauli string

$$P_a = \sigma_{p(a,1)} \otimes \sigma_{p(a,2)} \otimes \ldots \sigma_{p(a,N-1)} \otimes \sigma_{p(a,N)} \otimes, \quad p(k,i) \in \{I, X, Y, Z\} \qquad \text{Eq (3.2.71)}$$

we gain information about other string, e.g. P_b, which has some same elements, for example $p(a, j) = p(b, j)$. These elements are referred to as *overlapping* Pauli elements. In order to use this overlap we must simultaneously diagonalize a set of Pauli strings, which forms an Abelian group, by unitary rotations of measurement basis. Thus, the number of terms that need to be measured is reduced through the joint measurement of the qubits register for the whole Abelian group. This joint measurement

gives information about multiple Pauli operators simultaneously. The general prescription from the method starts with finding the generators of the Abelian group, these we denote $\{\tau_i\}$ and follows by finding a unitary U which satisfies

$$U\tau_i U^\dagger = Z_{q(i)}, \qquad \text{Eq (3.2.72)}$$

where $q(i)$ is a map from a generator index i to a qubits in register[8]. From the equation Eq (3.2.72) follows that the expectation value of τ_i can be found by measuring value for $Z_{q(i)}$. The same approach can be applied to the whole Abelian group, where U is applied to the group and all generators are then measured by measuring σ_Z. One implementation is the Qubit-wise commutativity (QWC). We define as a set where the Pauli operators with same indices commute in all Pauli strings of the set. For example strings XI, IZ and XZ form a QWC. For the QWC implementation we have single qubit $U_{\text{QWC}} := U$

$$Z = R_Y\left(-\frac{\pi}{2}\right)^\dagger X R_Y\left(-\frac{\pi}{2}\right), \quad Z = R_X\left(\frac{\pi}{2}\right)^\dagger Y R_X\left(\frac{\pi}{2}\right) \qquad \text{Eq (3.2.73)}$$

To conclude this section we mention there are more complex methods such as General Commutativity (GC), which have more complicated unitaries. Their advantage is greater reduction of the number of terms to measure, e.g. GC can for the *ab initio* molecular Hamiltonian reduce the number of terms from $O(N^4)$ to $O(N^3)$.[6]

3.2.4 Ansatz

The next essential part of a VQE is the *ansatz*. This term refers to the structure of the parametrized quantum circuit, which prepares the trial state, i.e. a model of the trial wavefunction. The prepared state is the used to measure the Hamiltonian as illustrated in diagram 3.1. During the VQE optimization loop the trial wavefunction takes form of the optimal wavefunction. There are a number of possible ansatze and the selection has to be made according to the solved problem. There exist metrics which are used to characterize the ansatz constructions. The most frequent are the *expressibility* and *trainability*. The former quantifies the uniformity of ansatz span across the unitary space. The rigorous definition uses the distance between the distribution of unitaries generated by ansatz and maximally uniform distribution, which is measured using the Haar measure. The latter stand for the quantified ability to find optimal parameter set to reach the optimal wavefunction through the iterative optimization with respect to expectation values of the problem Hamiltonian and the required time.[50]

A typical problem of VQE which has to be addressed in the ansatz is the *barren plateau problem*. The term stands for the situation in which the gradient, used for the selection of new set of parameters for new iteration, is vanishing. In this situation the cost function is effectively constant. The barren plateau is usually restricted to cases, when the gradient vanishes exponentially, because this leads to exponential increase of needed time, e.g. polynomial time vanishing is still considered trainable. This problem is not limited to VQE, it is present for the whole VQA category of algorithms. The problem can be visualized using an equivalent problem of *narrow gorges*. The term narrow gorge refers to situation, when a local minimum contracts exponentially with growing number of qubits of the model.

For a single parameter model the narrow gorge problem is illustrated in the figure 3.4, the minimum is increasingly difficult to find as the graph "gorge" becomes increasingly narrow. More rigorously,

[8] Z was used to represent the operator σ_Z to avoid confusing indices

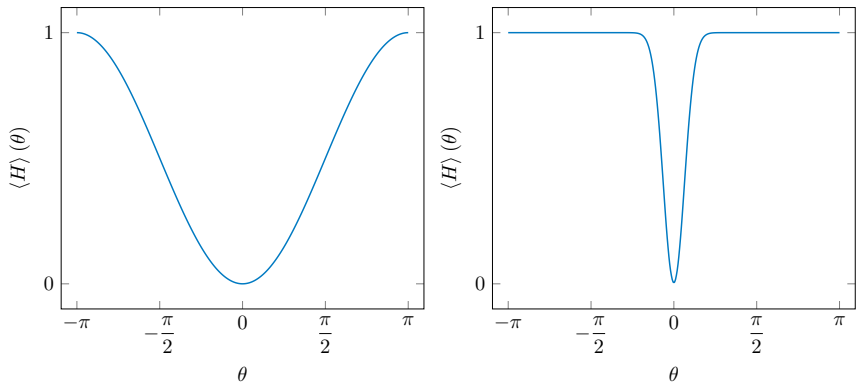


Figure 3.4: Comparison of an example expectation value $\langle H \rangle (\theta)$ with one parameter θ for the cases when the barren plateau does not occur (*subfigure on the left*) and for baren plateau occuring (*subfigure on the right*), inspired by [51]

consider VQE problem, for which the expectation value for problem Hamiltonian plays role of a cost function[9]

$$E(\boldsymbol{\theta}) = \langle \Psi(\boldsymbol{\theta}) | H | \Psi(\boldsymbol{\theta}) \rangle \,, \qquad \text{Eq (3.2.74)}$$

where $|\Psi(\boldsymbol{\theta})\rangle$ denotes the parametrized wave function. The barren plateau occurs for $E(\boldsymbol{\theta})$, if and only if

$$\forall \theta_i \in \boldsymbol{\theta}, \quad \forall \epsilon > 0, \quad \exists b > 1, \quad \text{Prb}\left(|\partial_{\theta_i} E(\boldsymbol{\theta})| \geq \epsilon\right) \leq O\left(\frac{1}{b^N}\right). \qquad \text{Eq (3.2.75)}$$

This equation Eq (3.2.75) can be seen as a consequence of the Chebychev's inequality[4].

There are many causes[10] of the barren plateau problem. It can be shown that the problem occurs more often with increase of system size or growth of the expressibility of ansatz[11][50]. The cost function also impacts the likelihood of barren plateau, specifically the locality of the cost function. More local cost functions, based on local observables are more resilient to the problem[51]. Other major factors which play role in the barren plateau are the noise and the degree of trial function entanglement[6]. From the barren plateau problem causes which are listed above follows that some of the most accessible means to avoid it are the use of less expressive ansatze and use of local encoding with lower Pauli weights.

Hardware-efficient ansatz

In order to give a complete picture of the ansatz topic in VQE, we give examples of some important classes of ansatze. First example of a VQE ansatz is the hardware-efficient ansatz (HEA), a type of fixed structure ansatz. The significance of HEA lies in its simplicity, which enables HEA to be modified for the particular quantum computer.

Definition 3.11 (Hardware efficient ansatz (HEA))**.** [6] We define the HEA ansatz as fixed structure ansatz constructed by repeating blocks of interweaved single qubit rotations gates controlled by

[9] Thus, it is to be minimized.
[10] The causes are in some literature referred to as *drivers* of the problem.
[11] The trainability and expressibility properties of the ansatz are related inversely[50].

parameters and "ladders" of entangling gates. Hence the trial state can be represented as

$$|\Psi(\theta)\rangle = \left(\prod_{i=1}^{d} (U_{\mathrm{ROT}}(\theta_i) \times U_{\mathrm{ENT}})\right) \times U_{\mathrm{ROT}}(\theta_{d+1}) |\Psi_{\mathrm{INIT}}\rangle, \quad U_{\mathrm{ROT}}(\theta_i) := \prod_{q,p}^{N,P} R_p^q(\theta_i^{pq}), \quad \text{Eq (3.2.76)}$$

where $q \in \{0, 1, \ldots, N\}$ indices are the qubit addresses in the register and $p \in \{x, y, z\}$ set the axis of rotation[12]. A basic example of a HEA block with R_x and R_y rotations and $U_{\mathrm{ENT}} = \texttt{CNOT}$ for $N = 2$ can be seen on figure 3.5.

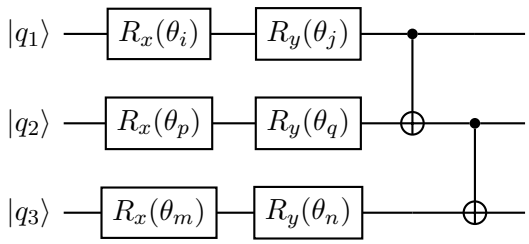


Figure 3.5: A basic example of a HEA block with R_x and R_y rotations and $U_{\mathrm{ENT}} = \texttt{CNOT}$ for $N = 2$, i.e. for three qubits, inspired by [6]

From the definition follows that the HEA is organized into L layers. Hence, the circuit depth is $O(L)$ and the number of parameters is $O(N \cdot L)$. The problem of HEA in order to guarantee success, it has to span over large part of the studied Hilbert space. In the worst case the number of layers L scales exponentially to cover the entire Hilbert space. From this follows that HEA is prone to barren plateaus.[6]

Unitary Coupled Cluster Ansatz (UCC)

The Unitary Coupled Cluster ansatz originates from quantum chemistry and nuclear physics. Partially thanks to the initial article describing VQE[29], where this ansatz was applied, the development of UCC for VQE lead to this ansatz being the currently most widely researched ansatz for the VQE and related algorithms. Since its formulation the method was developed further into multiple extensions for different problems. The unitary form of UCC suitable form quantum computation was derived from the Coupled Cluster ansatz used in classical quantum chemistry.[38]

To introduce the working of the ansatz, consider a reference state of an N qubit system $|\Psi_{\mathrm{REF}}\rangle$, for example $|\Psi_{\mathrm{REF}}\rangle = |\mathbf{0}\rangle = |0 \ldots 0\rangle$. This state plays role of a starting point from which the Hilbert space is gradually explored via change of parameters. The set of states which are explored is limited in sense of Hamming distance (defined in 1.4) reachable through "bit flips". This related method is referred to as *configuration interaction* and may be written as

$$|\Psi_{\mathrm{CI}}(\boldsymbol{\theta})\rangle = \sum_p \theta_p \sigma_p^+ |\Psi_{\mathrm{REF}}\rangle, \quad \theta_p \in \mathbb{C}, \qquad \text{Eq (3.2.77)}$$

where σ_p^+ were defined in Eq (3.2.45). The concept of configuration interaction is extended by the coupled cluster, where the spin-flip serves as a generator used to explore the Hilbert space from well

[12] The $\times$ symbol here stands for the successively applied gates not the thezor product $\otimes$

defines state with all spins aligned - $|\Psi_{\text{REF}}\rangle = |\mathbf{0}\rangle$. For example the restriction to a space of single spin-flip states can be expressed as

$$|\Psi_{\text{CC1}}(\boldsymbol{\theta})\rangle = \exp\Big(\sum_p \theta_p \left(\sigma_p^+ - \sigma_p^-\right)\Big)\,|\Psi_{\text{REF}}\rangle\,. \qquad \text{Eq (3.2.78)}$$

The portion of explorable space is increased by introducing more bit flips. The pitfall of this approach are the situations when the reference state $|\Psi_{\text{REF}}\rangle$ is result of some procedure, which may give a state such as $\Psi_{\text{REF}} = |+\cdots+\rangle\,, |+\rangle = (|0\rangle + |1\rangle)\big/\sqrt{2}$. For such reference state the flipping leads only to the same state. The main problem of coupled cluster follows from its description in fermionic operators

$$T(\boldsymbol{t}) := \sum_{i=1}^{k} T_i(\boldsymbol{t}), \qquad \text{Eq (3.2.79)}$$

$$T_1(\boldsymbol{t}) := \sum_{i,a} t_i^a a_a^\dagger a_i, \qquad \text{Eq (3.2.80)}$$

$$T_2(\boldsymbol{t}) := \sum_{i,j,a,b} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i \qquad \text{Eq (3.2.81)}$$

For the fermionic operator notation the coupled cluster takes form

$$|\Psi_{\text{CC}}\rangle = \exp(T)\,|\Psi_{\text{REF}}\rangle\,, \qquad \text{Eq (3.2.82)}$$

from which it can be seen that the issue of coupled cluster is that the $\exp(T)$ operator is not unitary, thus it can hardly be implemented on a quantum computer. A solution of this issue follows from the fact that T is antihermitean ($T^\dagger = -T$), thus its exponential is unitary ($\exp(T)\exp\left(T^\dagger\right) = \exp(T)\exp(-T) = I$). The UCC replaces the operator $\exp(T)$ with a unitary operator $\exp\left(T - T^\dagger\right)$, because $T - T^\dagger$ is, again, antihermitean. First we will return to the original notation. Using the fact that $\sigma^+, \sigma^-, \sigma^Z, I$ can serve as a base from which the Pauli operator can be expressed, we may move onto more general unitary coupled cluster ansatz.

$$\mathrm{i}\left(\sigma_p^+ + \sigma_p^-\right) = \mathrm{i}\sigma_p^x \begin{pmatrix} 0 & \mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}_p = \mathrm{i}\sigma_p^1 \qquad \text{Eq (3.2.83)}$$

$$\left(\sigma_p^+ - \sigma_p^-\right) = \mathrm{i}\sigma_p^y \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}_p = \mathrm{i}\sigma_p^2 \qquad \text{Eq (3.2.84)}$$

$$\mathrm{i}\sigma_p^z \begin{pmatrix} \mathrm{i} & 0 \\ 0 & \mathrm{i} \end{pmatrix}_p = \mathrm{i}\sigma_p^3 \qquad \text{Eq (3.2.85)}$$

The UCC builds upon the idea of cluster operators, which serve as generators.

Definition 3.12 (unitary Coupled Cluster operator)**.** [38, 52, 6] For the UCC we define the cluster operator as a generator used for the exploration of Hilbert space starting from the reference state A first order cluster operator may be written as

$$T_1(\boldsymbol{\theta}) = \mathrm{i}\sum_{p,\alpha} \theta_p^\alpha \sigma_p^\alpha, \qquad \text{Eq (3.2.86)}$$

where $\theta_p^\alpha \in \mathbb{R}$ and p are qubit addresses and $\alpha \in \{X, Y, Z\}$. The general cluster operator then takes form

$$T_k(\boldsymbol{\theta}) = \mathrm{i}\sum_{\boldsymbol{p},\boldsymbol{\alpha}} \theta_{\boldsymbol{p}}^{\boldsymbol{\alpha}} \sigma_{\boldsymbol{p}}^{\boldsymbol{\alpha}}, \qquad \text{Eq (3.2.87)}$$

where $\sigma_{\boldsymbol{p}}^{\boldsymbol{\alpha}}$ denotes $\sigma_{p_1}^{\alpha_1}\sigma_{p_2}^{\alpha_2}\ldots\sigma_{p_k}^{\alpha_k}$ and $\theta_{\boldsymbol{p}}^{\boldsymbol{\alpha}}$ stand for the tensor that holds the variational parameters. Thus, a full cluster operator up to order k is defined by formula

$$T^{(k)}(\boldsymbol{\theta}) = \sum_{i=1}^{k} T_i(\boldsymbol{\theta}) \qquad \text{Eq (3.2.88)}$$

From the general coupled cluster operator and the reference (state) $|\Psi_{\text{REF}}\rangle$ arises the UCC state of order k.

$$\left|\Psi_{\text{CC}}^{(k)}\right\rangle = \exp\left(T^{(k)}(\boldsymbol{\theta})\right)|\Psi_{\text{REF}}\rangle \qquad \text{Eq (3.2.89)}$$

The set of possible actions which can be performed using Eq (3.2.89) are all unitary transformations on k qubits which preserve the global phase, i.e. $\text{SU}(2^k)$. This representation has a total of $O((3N)^k)$ real parameters.

A common setting of k is $k = 2$, as for such value the number of parameters grows only quadratically with number of qubits. The implementation of the $\exp\left(T^{(k)}(\boldsymbol{\theta})\right)$, uses the technique of Trotterization[13]. Using the first order in a Trotter factorization for $k = 2$ with Trotter number N we get

$$|\Psi_{\text{cc}}(\boldsymbol{\theta})\rangle = \Big(\prod_{p_1,p_2,\alpha_1,\alpha_2} \exp\big(\mathrm{i}\frac{\theta_{p_1p_2}^{\alpha_1\alpha_2}}{N}\sigma_{p_1p_2}^{\alpha_1\alpha_2}\big)\Big)^N |\Psi_{\text{REF}}\rangle\,. \qquad \text{Eq (3.2.90)}$$

Firstly, from the Trotterization Eq (3.2.90) follows that $k = 2$ is enough to perform arbitrary 1 or 2 qubits gates. Secondly, it is worth noting that the approximative formula Eq (3.2.90) may also be considered an alternative definition of UCC 3.12.

To see how the UCC can be implemented for the problems related to quantum chemistry such as work with electronic orbitals, it is necessary to rewrite the UCC formalism again into the notation of fermionic operators. For the coupled cluster we have

$$|\Psi_{\text{REF}}\rangle = \prod_i a_i^\dagger |\text{vac}\rangle \qquad \text{Eq (3.2.91)}$$

$$|\Psi\rangle = \exp\left(t - t^\dagger\right)|\Psi_{\text{REF}}\rangle \qquad \text{Eq (3.2.92)}$$

$$T^{(1)}(\boldsymbol{t}) = \sum_{i_1,p_1} t_{i_1p_1}(a_{i_1}^\dagger a_{p_1} - a_{p_1}^\dagger a_{i_1}) \qquad \text{Eq (3.2.93)}$$

$$T^{(2)}(\boldsymbol{t}) = \sum_{i_1,i_2,p_1,p_2} t_{i_1i_2p_1p_2}(a_{i_1}^\dagger a_{p_1} a_{i_2}^\dagger a_{p_2} - a_{p_2}^\dagger a_{i_2} a_{p_1}^\dagger a_{i_1}) \qquad \text{Eq (3.2.94)}$$

From the fermionic UCC we can get the formulas similar to Eq (3.2.78) and even Eq (3.2.90) via grouping of the excitation terms in T with their conjugated counterparts from $T^\dagger$ into τ.

$$U(\boldsymbol{t}) := \exp\big(\sum_j t_j(\tau_j - \tau_j^\dagger)\big) \qquad \text{Eq (3.2.95)}$$

$$U(\boldsymbol{t}) \approx U_{\text{Trotter}}(\boldsymbol{t}) = \Big(\prod_j \exp\big(\frac{t_j}{N}(\tau_j - \tau_j^\dagger)\big)\Big)^N \qquad \text{Eq (3.2.96)}$$

$$(\tau_j - \tau_j^\dagger) \overset{\text{Encoding}}{=} \mathrm{i}\sum_k P_{k,j} \qquad \text{Eq (3.2.97)}$$

$$\qquad \text{Eq (3.2.98)}$$

Where using $N = 1$ in Trotterization we get more general form the original formula Eq (3.2.89): $U_1(\boldsymbol{t}) = \prod_{j,k} \exp(\mathrm{i}t_j P_{j,k})$.

Comparing the UCC and HEA ansatze, we can see the obvious advantage of UCC in the number of parameters $O(N^3)$[14], whereas HEA has $O(N \cdot L)$ parameters. From this follows that UCC has

[13] To be specific, the Suzuki trotter factorization is used for the exponential. [52]

[14] The number of parameters varies between UCC and its modifications, some of them have only $O(kN^2)$ parameters.[6]

better control over the ansatz span on the Hilbert space. UCC can also provide parametrization of an arbitrary electronic wavefunction. It can also be shown that it is more efficient than HEA for certain tasks. The main issue of UCC follows from the overhead of exponential operator implementation, only the number of `CNOT` gates used in exponential of Pauli strings for double excitation has scaling of $O((2N^4) \cdot (q-1))$, where q denotes the average Pauli weight of a string. This of course leads to large circuit depth, which is even bigger isssue for the NISQ devices.[6]

3.2.5 Optimization of the ansatz parameters

As it was mentioned in the Section 3.1.1, before the loop 3.1 starts anew, the parameters of the ansatz defining the trial functions need to be updated. For each optimization loop, several measurements of the Hamiltonian expectation value need to be made. That is the computation of the update consists of two steps, computation of gradient for current iteration and optimization itself. The choice of the optimization method influences the number of such measurements as well as the number of iterations the VQE stack needs before converging and lastly it has impact on the resilience of VQE stack to problem specific issues such as the already mentioned barren plateau problem 3.2.4. In contrast to classic optimization the optimization in context of VQA has to overcome the noise present on NISQ devices, which can act against convergence of optimization, as well as precision limited by the number of shots performed between iterations and finally problematic landscape of expectation value, e.g. barren plateau.

To introduce the methods of optimization used for VQAs it is suitable to start with notation which extends the notation used in chapter 2 with reference to publication [6]. The vector $\boldsymbol{O}(\boldsymbol{\theta}) = (O_1(\boldsymbol{\theta}^{(1)}), O_2(\boldsymbol{\theta}^{(2)}), \ldots, O_a(\boldsymbol{\theta}^{(a)}))$ stand for the set of observables from which the objective function is composed of, $a \in \mathbb{N}$ is the number of these observables. Next we define C, a function mapping the observables to the objective function, for this section, it can be thought as

$$C(\boldsymbol{X}) = \sum_i c_i X_i; \quad c_i \in \mathbb{R}, \forall i. \qquad \text{Eq (3.2.99)}$$

The parametric objective function $F(\boldsymbol{\theta})$ is then given as the expectation value of $C(\boldsymbol{O})$. For the observables we have expectation values defined as

$$\left\langle O_k(\boldsymbol{\theta}^{(k)}) \right\rangle = \langle \Psi_0 | U^{(k)\dagger}(\boldsymbol{\theta}^{(k)}) M^{(k)} U^{(k)}(\boldsymbol{\theta}^{(k)}) | \Psi_0 \rangle \,, \qquad \text{Eq (3.2.100)}$$

where $|\Psi_0\rangle$ is the initial state, $M^{(k)}$ represents a Hermitian measurement operator, and finally $U^{(k)}(\boldsymbol{\theta}^{(k)})$ is the chosen variational ansatz with parameters $\boldsymbol{\theta}^{(k)}$

$$U^{(k)}(\boldsymbol{\theta}^{(k)}) = \prod_j U_j^{(k)}(\theta_j^{(k)}), \quad U_j^{(k)} = \exp\left(\mathrm{i}\theta_j^{(k)}\right) P_j^{(k)}, \qquad \text{Eq (3.2.101)}$$

were $P_j^{(k)}$ is Hermitian, e.g. tensor product of Pauli matrices.

Finding the gradient

In order to find the value of gradient multiple method can be applied. The first is the *Finite Difference Stochastic Approximation*(FDSA), which uses difference between two function points to approximate

the gradient. This method can be described by formula

$$\left(g(\boldsymbol{\theta}_t)\right)_j := \frac{F(\boldsymbol{\theta} + c_t\mathbf{e}_j) - F(\boldsymbol{\theta} - c_t\mathbf{e}_j)}{2c_t}, \qquad \text{Eq (3.2.102)}$$

where $\left(g(\boldsymbol{\theta}_t)\right)_j$ represents j-th component of gradient (approximation) at $\boldsymbol{\theta}_t$ and $\mathbf{e}_j$ denotes a basis unit vector of the unit and c_j is the constant that scales the step[15]. This method needs only one measurement. Another method that can be used is the *Simultaneous Perturbation Stochastic Approximation*(SPSA)[16], which differs from FDSA by allowing different from unitary basis vector steps:

$$\left(g(\boldsymbol{\theta}_t)\right)_t := \frac{F(\boldsymbol{\theta} + c_t\boldsymbol{\Delta}_t) - F(\boldsymbol{\theta} - c_t\boldsymbol{\Delta}_j)}{2c_t(\boldsymbol{\Delta}_t)_j}, \qquad \text{Eq (3.2.103)}$$

where $\boldsymbol{\Delta}_t$ denotes a random perturbation.

Alternatively to the stochastic approximation methods, there exist means of calculation the gradient of measurement of observables using the quantum resources directly[53]. Using the slightly simplified notation from preceding paragraphs 3.2.5.

$$\frac{\partial \langle O_k(\boldsymbol{\theta})\rangle}{\partial\theta_j} = 2\,\mathrm{Im}\left(\langle\Psi_0|V_k^{j\dagger}(\boldsymbol{\theta})M^{(k)}U(\boldsymbol{\theta})|\Psi_0\rangle\right), \qquad \text{Eq (3.2.104)}$$

$$V_k^j(\boldsymbol{\theta}) = \exp\left(\mathrm{i}\theta_{N_k}P_k^{N_k}\right)\ldots\exp\left(\mathrm{i}\theta_j P_k^j\right)P_k^j\exp\left(\mathrm{i}\theta_{j-1}P_k^{j-1}\right)\ldots\exp\left(\mathrm{i}\theta_1 P_k^1\right) \qquad \text{Eq (3.2.105)}$$

The simplest way to implement $V_k^{j\dagger}(\boldsymbol{\theta})M^{(k)}U(\boldsymbol{\theta})$ is the use of an ancillary qubit[17].[53] The ancillary qubit is prepared into $|+\rangle = (|0\rangle + |1\rangle)\big/\sqrt{2}$ giving state

$$|\Psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |\Psi_0\rangle \qquad \text{Eq (3.2.106)}$$

Next the controlled P_j gate is applied with control ancillary qubit (controlled 1) and the rest of unitary is applied with control 0. In the following step the measurement operator is applied again with control on ancillary qubit (control 1). The resulting state is

$$|\Psi\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle \otimes U(\boldsymbol{\theta})\,|\Psi_0\rangle + |1\rangle \otimes M^{(k)}V_k^j\,|\Psi_0\rangle\Big) \qquad \text{Eq (3.2.107)}$$

Last step before measurement, of course in the appropriate basis, is the application of second Hadamard gate. Hence, $\mathrm{Im}\left(\langle\Psi_0|V_k^{j\dagger}(\boldsymbol{\theta})M^{(k)}U(\boldsymbol{\theta})|\Psi_0\rangle\right)$ is now in the ancillary qubits encoded in Y basis and can be measured using appropriate basis change. The process can be summarized into a fairly simple circuit, as seen on figure.

Gradient based optimization methods

The gradient calculation methods are key component in the optimization methods, so-called *optimizers* used to update the parameters between iterations. The optimizers can be divided into first order and

[15] Hence, it has to decrease among iterations

[16] The term Simultaneous Perturbation Stochastic Approximation also refers to a an optimization method which is based on the same random perturbations as this methos of gradient calculation. This method later mentioned in the Chapter 5

[17] This approach is in some literature referred to as *direct analytical gradient measurement*

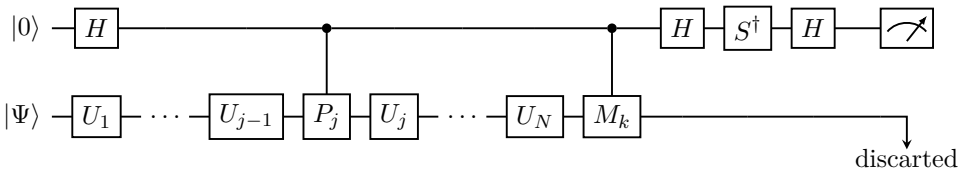


Figure 3.6: Circuit encoding the Im $\Big(\langle \Psi_0 | V_k^{j\dagger}(\boldsymbol{\theta}) M^{(k)} U(\boldsymbol{\theta}) | \Psi_0 \rangle \Big)$, for the direct analytical gradient measurement, inspired by [6]

second order optimizers. The former use only the first partial derivatives, whereas the latter utilize the second derivatives as well. The optimizer algorithms can be described using the same steps as the classical optimizers used for the deep learning purposes.[6, 54] An example of first order optimizers is the *simple gradient descent*, based on the gradient descent method, which dates to 19th century and is associated with mathematician Augustin-Louis Cauchy. Its basic idea is as simple as always taking a step in the opposite direction of the gradient with the step size determined by the absolute value of gradient and a preset constant parameter η, which is called *learning rate*. Thus, using the classic pseudocode notation[18], we have There are numerous stochastic optimization algorithms which improve

Algorithm 2: Simple gradient descent, inspired by [54]

Data: learning rate $\eta \in \mathbb{R}$, objective function F, point $\boldsymbol{\theta_t}$
Result: Optimimal parameter set $\boldsymbol{\theta}_o$
begin
  while *Not converged* **do**
    $g_t \leftarrow g(\boldsymbol{\theta}_t)$ `/* Evaluate gradient */`
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot g_t$

upon the basic gradient descent idea. Some of the more significant and recent are the RMSProp and the Adam optimizer[55]. The former modifies the gradient by division by square root of moving average as can be seen from algorithm below. Note that the ϵ constant prevents the division by zero,

Algorithm 3: RMSProp optimizer method, inspired by [55]

Data: learning rate $\eta \in \mathbb{R}$, moving average paramater $\gamma \in (0; 1)$, small constant $\epsilon \in (0; 1)$, bjective function F, point $\boldsymbol{\theta_t}$
Result: Optimimal parameter set $\boldsymbol{\theta}_o$
begin
  while *Not converged* **do**
    $g_t \leftarrow g(\boldsymbol{\theta}_t)$ `/* Evaluate gradient */`
    $E[g^2]_t \leftarrow \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \dfrac{\eta}{\sqrt{E_t} + \epsilon} g_t$

its value can be as small as 10^{-8}. The latter extends the RMSProp optimizer further by use of the moving square on both the gradient and its square. This approach requires a bias correction. In the introduction of this section we have mentioned the second order optimizers. A good example of a such method is the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) *algorithm* Below is a schematic description of the algorithm, its components such as the line search and matrix equation solution are out of the project scope. The algorithm is based on the approximation of the objective function Hessian, so new variables need to be defined. To conclude this section we mention there are also analytical method

[18] Unlike in equations in the algorithms, the convention of variable being set value is denoted as `variable` ← `value`.

Algorithm 4: Adam optimizer method, inspired by [55]

Data: Learning rate $\eta \in \mathbb{R}$, moving average parameter for gradients $\beta_1 \in (0;1)$, moving average parameter for squared gradients $\beta_2 \in (0;1)$, small constant $\epsilon \in (0;1)$, objective function F, point $\boldsymbol{\theta_t}$

Result: Optimal parameter set $\boldsymbol{\theta}_o$

begin

  while *Not converged* **do**

    $g_t \leftarrow g(\boldsymbol{\theta}_t)$ `/* Evaluate gradient */`

    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$

    $v_t \leftarrow \beta_2 v_{t-1} + (1-\gamma) g_t^2$

    $m_t' \leftarrow \dfrac{m_t}{1-(\beta_1)^t}$

    $v_t' \leftarrow \dfrac{v_t}{1-(\beta_2)^t}$

    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \dfrac{\eta}{\sqrt{v_t'}+\epsilon} m_t'$

Algorithm 5: BFGS second order optimizer, [6]

Data: Initial parameters $\boldsymbol{\theta}_0$, initial Hessian approximate B_0, objective function $F(\boldsymbol{\theta})$

Result: Optimal parameter set $\boldsymbol{\theta}_o$

begin

  if B_0 *not set* **then**

    Calculate B_0 from $\boldsymbol{\theta}_0$

  while *Not converged* **do**

    $\boldsymbol{p}_k \leftarrow \text{Solve}(B_k \boldsymbol{p}_k = -\nabla F(\boldsymbol{\theta}_k))$ `/* Find the descent direction */`

    $\alpha_k \leftarrow \text{argmin}(F(\boldsymbol{\theta}_k + \alpha_{k-1}\boldsymbol{p}_k))$ `/* Perform line search */`

    $\boldsymbol{s}_k \leftarrow \alpha_k \boldsymbol{p}_k$

    $\boldsymbol{\theta}_{k+1} \leftarrow +\boldsymbol{s}_k$ `/* Parameter update */`

    $\boldsymbol{y}_k \leftarrow \nabla F(\boldsymbol{\theta}_{k+1}) - \nabla F(\boldsymbol{\theta}_k)$

    $B_{k+1} \leftarrow B_k + \dfrac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \dfrac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k^T}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k}$

of optimization available. Example of such methods is the *Jaccobi diagonalization and Anderson acceleration*[56]. This type of method has the advantage of lower dependence on the gradient, giving them resilience to barren plateau problem, while creating greater computational overhead.

3.2.6 Suppression of errors and noise

An important topic for the VQAs applied on NISQ devices is the reduction of impact of present noise and overall error mitigation. This broad topic is completely out of the project scope, so we will restrict ourselves to reference to literature, which discusses the problem in greater depth. The problem of error mitigation is discussed in general in [4]. A recent example of a publication exploring this problem is the tech report [6].

Chapter 4

Quantum programming tools

While the quantum computers of current NISQ era are in general far from being practical in terms of general problem-solving, the development of quantum computers working with dozens of physical qubits and the rise in quantum algorithm research in recent years led to great demand for tools that could be used to program and simulate such a computer[7]. The set of tasks that need top be addressed can be formulated in the following way:

- **High level programming language**: To describe the quantum programs, there first needs to be a programming language, capable of modelling the qubits, quantum operations, measurements etc. The creation of such language is preceded by choice of the programming paradigm and the other parameters for such language. The high level language takes advantage of the simpler notation of more complex operations. The code programmed in the high level language may need to be first translated into low level language to be executed on quantum hardware[1].
- **Low level language**: The low level language is not an indispensable thing in quantum computation, because the high level language could be implemented directly. Nevertheless, in the current quantum technologies it plays essential role of intermediate representation of the quantum instructions, that are to be executed on the quantum hardware.
- **Quantum simulator**: During the development of quantum programs and algorithms testing it preferable to have access to means of experimenting without need for real quantum hardware. A quantum simulator offers numerical simulation facilities which serve this exact purpose.
- **Quantum algorithms**: For classical programming it is essential to have means of compiling algorithms and more complicated programs into packages or libraries, which can be further distributed, so they might be reused. The same problem should be addressed for quantum computing likewise.

In this chapter we provide a brief review of the tools that are currently available to perform the tasks listed above.

[1] This is the case of the Qiskit library mentioned later in this chapter.

4.1 Early quantum programming

In the 1990s the situation in quantum programming was in similar to early classical programming. The studied programs were mostly described using the pseudocode, i.e. a human-readable of lists of instructions for abstract hardware, that cannot be directly executed on any real hardware. The quantum pseudocode was formalized for the first time by a convention in 1996[57]. The imperative pseudocode with its formalism for describing low level operations and QRAM[2] became an inspiration for the class of imperative quantum programming languages. In the same year 1996 another approach quantum programming was presented in the publication [59]. The alternative to the imperative pseudocode is the functional lambda calculus. The quantum lambda calculus continues its development to this day and although it remains a description of quantum programs, rather than being used as a programming language, although there exists implementation in Scheme language. The lambada calculus in context of both classic and quantum computing can be considered a basis for the functional programming languages.

4.2 Quantum programming languages and SDKs

In the last twenty years the number of quantum computing languages raised significantly[3]. It should be mentioned that most of the quantum programming tools today are developed as open source software and proprietary components remain mostly in the domain of hardware specific software and components. In this section only the most significant projects will be listed.[60, 61]

4.2.1 Quantum instruction sets

The instructions sets are used in order to translate the high level algorithms often written using high level (quantum) programming languages. The instruction have the role of low level programming languages and are often hardware-specific. Some of the more widely known are

- **cQASM**: Common QASM is a quantum assembly language, which is hardware-agnostic, i.e. it offers possibility of information exchange between quantum compilation tools and the simulator software.
- **Quil**: Quil is a instruction set architecture with shared memory architecture, where the memory is accessible to quantum and classical component. Associated with Common Lisp.

[2] It needs to be emaphsized that the QRAM is rather a model of quantum computation rather than a quantum Random Access Memory, as the term could suggest. This concept is based on a quantum register of qubits on which gates are applied similarly to the circuit model, but the QRAM model explicitly accounts for the classical computer which is controling the quantum apparatus.[58]

[3] At the time of writing there is no comprehensive list of quantum programming languages. Nonetheless, there are fourteen major languages listed on the relevant Wikipedia page[60], excluding the low level instruction sets and quantum SDKs.

- **OpenQASM**: OpenQASM is part of Qiskit, it is based on circuit representation of programs. Its design is based on conventional hardware description languages. It was first described in 2017 paper[62].

4.2.2 Quantum programming languages

Apart from low level programming languages, which are used to run software "close" to the actual quantum hardware, there exist high level quantum programming languages, in this context they will be referred to only as quantum programming languages. These languages can also be regarded as meta-programming languages because they are not executed on the quantum hardware itself. The following subsections present a short list of major quantum programming languages sorted by their programming paradigm.

Imperative languages

In this list of imperative quantum programming languages we have excluded the low level languages, pseudocode and the quantum programming languages, that have form a library for a conventional programming language.

- **QCL**: Quantum Computation Language can be considered as one of the first implemented languages. It was introduced in 2017 paper. It is conceptually based on C programming language.
- **Q#**: Q Sharp is a full-stack programming language for the Quantum Development Kit created by Microsoft. It has syntax inspired by C Sharp.
- **Q language**: Historically the Q language was the second implemented quantum programming language. It is implemented as and extension of C++ programming language.
- **Silq**: Silq is one of the most recent languages, it was introduced in 2020. It is written as a high-level programming languages with complex features such as automatic uncomputation.

Functional languages

The functional languages are often praised for their closeness to lambda calculus concepts, which makes them more suitable for study of programming concepts in both conventional and quantum programming. The list below does not include the quantum lambda calculus and its various implementations to avoid confusion with separate programming languages based on more traditional approaches to functional programming.

- **QML**: QML stands for Quantum Meta Language[4]. It is heavily inspired by Haskell conventional programming language. The semantics of the language is based on quantum circuit representation of quantum programs.
- **LIQUi|>**[5]: This language is built as an extension of the F# conventional language and offers facilities such as simulators and variety of algorithms.
- **Quipper**: Quipper language is an extension of the conventional Haskell. Some of its functionality, e.g. simulators, are implemented as Haskell libraries.

4.2.3 Quantum SDKs by developer

The situation with quantum programming languages, in which the programming languages depend on multitude of tools, for example simulator, quantum hardware interfaces and libraries of quantum algorithms and functions, led to the creation of Software Development Kits (SDK), i.e. collections of tools related to quantum programming. The development of a quantum programming SDK or its analogy is a complex and expensive enterprise, so it is not surprising the most advanced projects are developed by large IT companies and quantum computing firms.

Developer	Project	Comment
IBM	Qiskit	`Python` library, based on `OpenQASM` circuit representation, offers simulators and hardware (local or cloud)
Microsoft	Quantum Development Kit	full-stack `Q#` language, part of `.NET framework`, only simulations available
Google	Cirq	`Python cirq` libraries, simulators and hardware internally at Google
Riggeti Computing	Forest	`Python pyquil` library translated to `quil` instructions, `Grove` library for optimization tasks
D-Wave	Ocean	tools for D-Wave hardware programming, Ising model and QUBO problems, `QMASM` low level language
Xanadu Quantum	Strawberry Fields	`Python` library and simulators, used for Xanadu's quantum photonic hardware, `Blackbird` instructions set
Xanadu Quantum	Pennylane	`Python` library and simulators, used for quantum devices by other developers
Budapest Quantum Computing Group	Piquasso	Programmed as a `Python` library, includes multiple simulators, specialised in photonic quantum computing

Table 4.1: Major quantum SDKs and their developers. Information taken from documentation of respective SDKs[63, 64, 65, 66, 67, 68, 69, 70]

[4]Not to be confused with Qt Modelling language and Quantum Machine Learning which share common abbreviation QML.

[5]The name of language is pronounced as *liquid*.

The quantum SDKs by IBM and Google deserve special attention, as they are associated with most advanced quantum hardware, and they are also involved in quantum programming education.

4.2.4 Qiskit

Qiskit, is an acronym for Quantum Information Science Kit, referring to a quantum SDK developed by IBM, namely the IBM Research department, under open source MIT licence. It was initially released in 2017 along with the OpenQASM instruction set mentioned in Section 4.2.1. The Qiskit is often regarded as the most full-featured quantum SDK today. [63, 71, 72]

The SDK includes tools for creation of quantum programs, which are represented by quantum circuits, simulation of quantum circuits and tools for quantum program execution on local or remote quantum hardware. The remote hardware is primarily accessed through IBM cloud computing service named IBM Quantum Experience, which gives user possibility to send the quantum programs for execution to multiple quantum devices across the globe. It should be emphasized that although the Qiskit is hardware-agnostic in concept[6], it currently supports only the devices based on superconducting qubits and trapped ions.

To run the quantum programs on hardware the programs are first translated from Python to the machine code level OpenQASM, which is then executed. The SDK allows user to work with OpenQASM directly, making the Qiskit suitable for low level programming. The OpenQASM is used not only for physical devices, in Qiskit it also plays role of programming language for the simulators.

Qiskit is built upon conventional Python3 language using libraries of the same name, which are loaded as necessary. The Python base gives Qiskit advantage of readable and easy to understand syntax, that is familiar to many programmers and also gives access to numerous data science and computer science tools that are available for Python, for example we mention NumPy, Pandas, Matplotlib. The Python integration is especially significant in context of interaction of user with Qiskit and installation. The official documentation suggests installation using the Anaconda Python distribution into an environment that is separated from the rest of operating system programs and libraries. This ensures the compatibility of all Python packages which are involved in work with Qiskit. Although the Qiskit code can be executed as an ordinary Python code, it is recommended to use the Jupyter Notebook. Jupyter Notebook[7] is a software used to work with interactive notebook documents can contain executable code, formatted Markdown and HTML text and images. The Notebook enables interactive work with code in Python and other computing languages similar to REPL functionality. The Jupyter Notebooks are used in the official Qiskit tutorials and in various educational programs associated with quantum computing and Qiskit, such as the courses and workshops by the QWorld organization.

The Qiskit is organized into multiple components which are listed below along with their functions to give a better idea about the structure of the software.

[6] Thanks to the universal quantum computation model of quantum circuits.

[7] Jupyter Notebook is built upon more extensive Jupyter project, which is a collection of standards, services and tools for interactive computing.

- **Qiskit Terra**: Contains the basic components, which create the quantum circuits and also performs the construction of the quantum gates of the circuits. Other components provide optimization and execution of code locally or remotely.
- **Qiskit Aer**: Provides quantum computing simulators and their infrastructure. Also contains realistic noise models.
- **Qiskit Ignis**: Now deprecated component for work with noise and error correction. Superseded by **Qiskit Experiments**.
- **Qiskit Aqua**: Now also deprecated component consisting of diverse tools and algorithms. Now replaced with **Qiskit Optimization**, **Qiskit Finance**, **Qiskit Machine Learning** and **Qiskit Nature**

In context of the two algorithms discussed in this project, it should be mentioned that thanks to the schematic nature of VQE and QAOA, both algorithms were implemented in the libraries of Qiskit. These implementations can be though of as basis which needs to be equipped with an optimizer and a quantum circuit defining the ansatz. Such classes can save time in certain situations, however use of such preprogrammed construct obscures the working of the algorithm and makes unsuitable for low-level research and education purposes.

To conclude the subsection about the Qiskit SDK we give a simple example of a quantum circuit, which performs the superdense coding with Qiskit code that performs the circuit and draws the image of circuit on figure.

Listing 4.1: Superdense coding algorithm circuit in Qiskit

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute,
    Aer
# Alice has bits to be encoded:
AbitA=False
AbitB=True
# Bob reads the measurement output
# initialise circuit and registers
q = QuantumRegister(2,"qr")
c = ClassicalRegister(2,"cr")
qc = QuantumCircuit(q,c)
# Actual circuit
qc.h(q[0])
qc.cx(q[0],q[1])
#qc.z(q[0])
#qc.x(q[1])
# Use of Python conditionals
if AbitA:
    qc.x(q[0])

if AbitB:
    qc.z(q[0])

#qc.z(q[0])
#qc.x(q[1])
qc.cx(q[0],q[1])
```

```
qc.h(q[0])
# Measurements
qc.measure(0,0)
qc.measure(1,1)
# Execution of the circuit on simulator and output
job = execute(qc,Aer.get_backend('qasm_simulator'),shots=1024)
counts = job.result().get_counts(qc)
tmp=tuple(list(counts.items())[0][0])
print(counts)
# Drawing the circuit using Mathplotlib library
qc.draw(output='mpl')
```

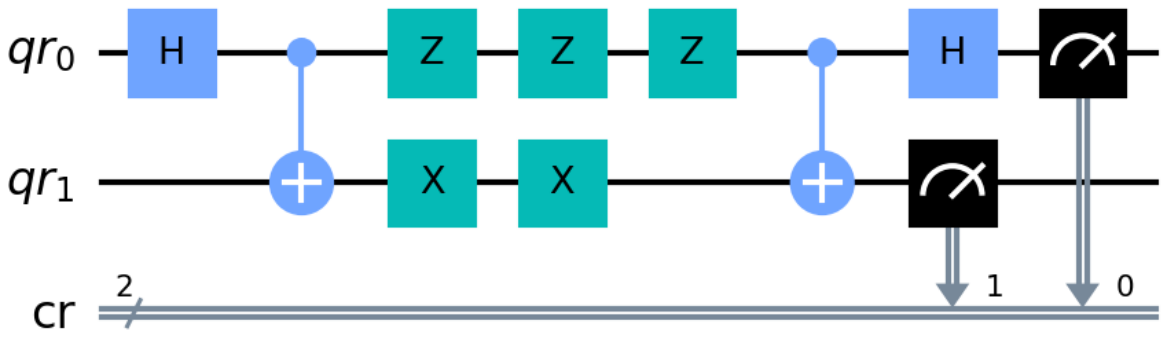


Figure 4.1: A quantum circuit diagram generated from qiskit code using `matplotlib` from code in listing 4.1

Projects associated with Qiskit

The information on the Qiskit SDK would not be complete without mentioning some of the projects which are closely linked with the SDK. The Qiskit has the advantage of the possibility of accessing the physical quantum hardware in the IBM Quantum project via the IBM Cloud. This way the code produced using the Qiskit can be tested with real noise and real limitations of currect NISQ quantum computers, e.g. limited set of quantum gates or virtual gates.

Another project worth mentioning is the currently developed Qiskit Metal, which is a software that can be used for quantum hardware prototyping and simulations. The software is meant to be compatible with the Qiskit SDK, so the developed circuits can be tested on the modeled hardware.

4.2.5 Cirq

Another quantum SDK which gained popularity in the recent years is the Cirq SDK developed by Google AI Quantum Team. Cirq, similarly to Qiskit, is constructed as a Python library, so the code can use the features of a modern programming language. Cirq SDK offers most of the features that

Qiskit has, such as work with quantum circuits, quantum gates and simulations. Currently the SDK does not have the functionality that would enable access to public physical quantum computers.

To give a better illustration of the Cirq SDK and the syntax of programming using its library, a code in Cirq analogical in terms of functionality to 4.1 is listed below.

Listing 4.2: Superdense coding algorithm circuit in Cirq

```
import cirq
from cirq import H, X, Y, Z, CX, measure
# Alice has bits to be encoded:
AbitA=False
AbitB=True
# Bob reads the measurement output
# initialise circuit and registers
q = cirq.LineQubit.range(2)
qc = cirq.Circuit()
# Actual circuit
qc.append(H(q[0]))
qc.append(CX(q[0], q[1]))
# Use of Python conditionals
if AbitA:
    qc.append(X(q[0]))

if AbitB:
    qc.append(Z(q[0]))


qc.append(CX(q[0], q[1]))
qc.append(H(q[0]))
# Measurements
qc.append(measure(*q, key='result'))
# Execution of the circuit on simulator and output
s = cirq.Simulator()
samples=s.run(qc, repetitions=1024)
print(qc)
print(samples.histogram(key='result',fold_func=bitstring))
```

Chapter 5

Algorithm implementation and application

This chapter is dedicated to the practical implementation of the QAOA and VQE algorithms. The necessary theory of quantum circuits and the studied algorithms were introduced in the first three chapters (1, 2, 3). The implementations presented here will be restricted to the Qiskit SDK and associated tool which were listed in the Chapter 4. Below are given multiple examples, which illustrate both simple and more complex implementations. Finally, few observations are made about the behaviour of the algorithms for different settings of the implemented circuits. Complete source code of the implementations is available on the Gitlab repository of the project[1]. It should be noted that all implementations utilize the noisy `qasm_simulator` from Qiskit SDK. The complete tabular data can be found on the project Gitlab repository.

5.1 Basic implementations

This section is dedicated to presenting the algorithms applied to rather simple problems. Although the algorithms for such settings do not have much practical use, nor they can reach quantum advantage, the examples are useful illustration of the algorithms and the used programming tools.

5.1.1 QAOA: Basic implementation

As we have shown in the Section 2.3, the QAOA can be easily implemented for objective functions based on boolean functions 2.1. The first problem that will be demonstrated with QAOA is the MaxCut, that was discussed in Section 2.3.1. To keep the problem easy to imagine, the partitioned graph is a simple four vertex graph displayed on figure 5.1. For this problem we have already derived the objective function Eq (2.3.13) and the problem Hamiltonian Eq (2.3.14) which follows from it. The problem unitary has form $\exp(-\mathrm{i}\gamma H_P)$, which can be simplified ignoring the global phase introduces

[1] A link to the repository can be found in the Appendix

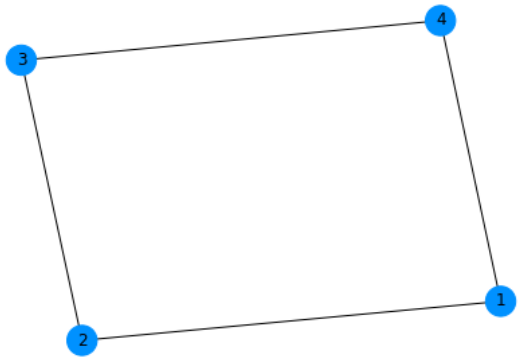


Figure 5.1: A simple graph with four vertices, on which the QAOA for MaxCut is demonstrated below.

by the identity matrices and only considering the effect of $Z_u Z_v$ terms. This leads to the question of implementing the unitary $\exp(-\mathrm{i}\gamma Z_u Z_v)$, which is refered to as RZZ gate. A simple matrix calculation can prove that the RZZ gate can be decomposed according to the following figure 5.2. Thus, we can

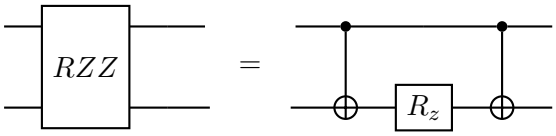


Figure 5.2: Circuit realizing the RZZ gate using one R_z gate and two `CNOT` gates

write the circuit for depth ($p = 1$) as depicted in figure 5.3[2]. An important component of QAOA

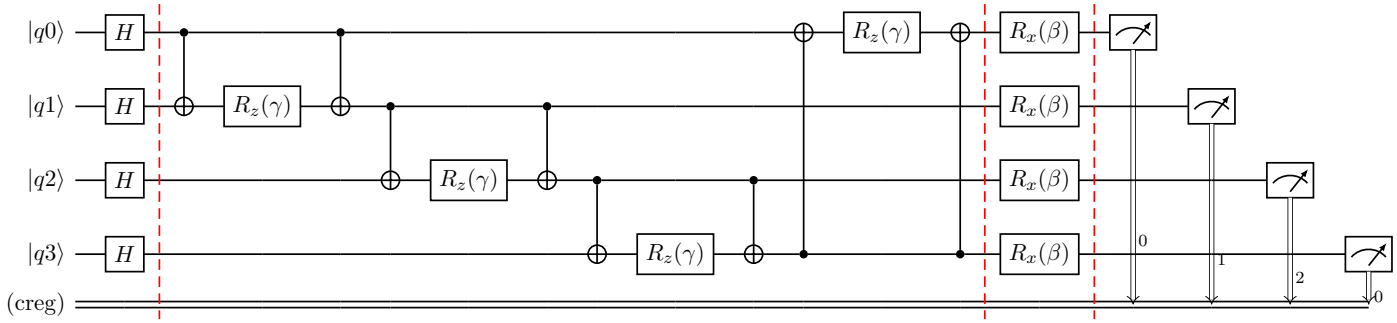


Figure 5.3: Depth 1 QAOA circuit for MaxCut

algorithm cannot be seen on the circuit diagram 5.3, the optimizer. In the Chapter 3, namely in Section 3.2.5, we have outlined the principles of multiple optimizers. For such a simple graph we can safely assume the gradient base optimizers can be applied. In order to make the examples more illustrative, the first order optimizers for simple gradient descent, gradient descent with scheduled learning rate[3] and RMSProp were implied from without use of dedicated package. The excerpts from the source code illustrating the optimizers are listed below. All the mentioned optimizers rely on gradient computation to make the implementation as simple as possible, the FDSA algorithms for gradient computation was chosen and implemented from scratch.

[2]The circuit for this implementation was programmed in Qiskit, so in order to keep the description of circuit concise, we have used the convention of measurement depiction used by Qiskit in the diagram above 5.3. As can be easily seen this way of depicting the sequential measurements is ill-suited for printed publication with limited horizontal space, for this reason in the following circuit the measurements will be symbolized by an indexed meter as in the previous chapters.

[3]The learning rate is unlike for simple gradient descent decreased between the iterations in order to avoid "stepping" over the minimum.

Listing 5.1: Basic implementation of Simple Gradient Descent optimizer described in 2

```
def gradient_descent(GRADIENT_FUN, INITIAL, ETA, MAX_ITER=100, EPSILON=1e-03):
    vector = np.array(INITIAL)
    learning_rate = ETA
    step = 0.1
    loops = 0
    for i in range(0, MAX_ITER):
        GRAD = GRADIENT_FUN(vector, step)
        diff = - learning_rate * np.array(GRAD)
        if np.all(np.abs(diff) <= EPSILON):
            break
        vector += diff
        for index in range(0, len(vector)):
            if np.abs(vector[index]) > 3.15/2:
                vector[index] /= 2
        step /= 1.02
        loops += 1
    return vector.tolist(), loops
```

Listing 5.2: Basic implementation of Scheduled Gradient Descent optimizer used in this project

```
def gradient_descent_scheduled(GRADIENT_FUN,INITIAL,ETA,MAX_ITER=100,EPSILON=1e-03)
    :
    vector = np.array(INITIAL)
    learning_rate = ETA
    step = 0.2
    loops = 0
    for i in range(0, MAX_ITER):
        GRAD = GRADIENT_FUN(vector, step=step)
        diff = - learning_rate * np.array(GRAD)
        if np.all(np.abs(diff) <= EPSILON):
            break
        vector += diff
        for index in range(0, len(vector)):
            if np.abs(vector[index]) > 3.15/2:
                vector[index] /= 2
        step = 2/(step+10)
        learning_rate = 1/(learning_rate+20)
        loops += 1
    return vector.tolist(), loops
```

Listing 5.3: Basic implementation of RMSProp optimizer discussed in 3

```
def RMSProp(GRADIENT_FUN, INITIAL, ETA, GAM, MAX_ITER=100, EPSILON=1e-03):
    vector = np.array(INITIAL)
    learning_rate = ETA
    avg_par = GAM
    step = 0.1
    loops = 0
    e_small = 0.001
    LEN = len(INITIAL)
    E = []
    for j in range(0,LEN):
```

```
        E.append(0)
    for i in range(0, MAX_ITER):
        GRAD = GRADIENT_FUN(vector, step)
        diff = []
        for j in range(0, LEN):
            E[j] = avg_par * E[j] + (1-avg_par)*(GRAD[j]**2)
            diff.append(- (learning_rate/(np.sqrt(E[j])+e_small)) * GRAD[j])
        if np.all(np.abs(diff) <= EPSILON):
            break
        vector += diff
        for index in range(0, len(vector)):
            if np.abs(vector[index]) > 3.15/2:
                vector[index] /= 2
        step = (np.array(diff).sum()/LEN)/2
        loops += 1
    return vector.tolist(), loops
```

Listing 5.4: Basic implementation of FDSA finite difference gradient method from Eq (3.2.102)

```
def gradient(point, step, obj_fun):
    grad = []
    for j in range(0,len(point)):
        point_plus = point
        point_minus = point
        point_plus[j] += step
        point_minus[j] -= step
        plus = (obj_fun(point_plus))
        minus = (obj_fun(point_minus))
        grad.append((plus-minus)/(2*step))
    return grad
```

Recall that for the simple gradient computation FDSA depends on the prescription of the `step` which is taken during computation in the cardinal directions. The application of gradient descent, even for such a simple problem, showed clearly the problems of the most basic method. The simple gradient descent is highly sensitive to the setting of the parameters, for our implementation that is the learning rate η and the prescription of the step decrease.

During the test the circuit was first optimized using the selected optimizer and then circuit was measured with parameters (β, γ) set to the optimizer result for the solution readout. All optimizers have eventually reached correct solution. The experiments with the number of shots used to compute the gradient have shown that the gradient descent methods are more susceptible to low precision of the gradient, because for a hundred shots, the method often did not converge at all. In that case the optimizers were stopped by the set maximum number of iterations (set to 100) rather that the convergence. Even in these cases the optimizers have reached results good enough for reading the solution, although it is obvious that for a more complex task the solution would be lost. The RMSProp optimizer, which utilizes concept of momentum, that is used to scale the gradient before calculating new position, has proven itself more resilient to gradient errors and reached the correct result for a hundred shots.

In order to give a different perspective of the problem solution, a gradient-free optimizer was also

tested for the problem. The chosen optimizer is the COBYLA method implemented in the `scipy` package as a minimization method. This optimizer obviously does not rely on the use of the gradient computation function, but its performance was measured via the number of iterations it needed to converge, i.e. effectively the number of circuit executions. COBYLA, thanks to its implementation is the most resilient method in terms of parameter changes, but the number of runs it needs to reach solution may be comparable to gradient method. It is also worth noting that this method is based on linear approximations, so it may fail for some problems with complicated parameter landscape. To

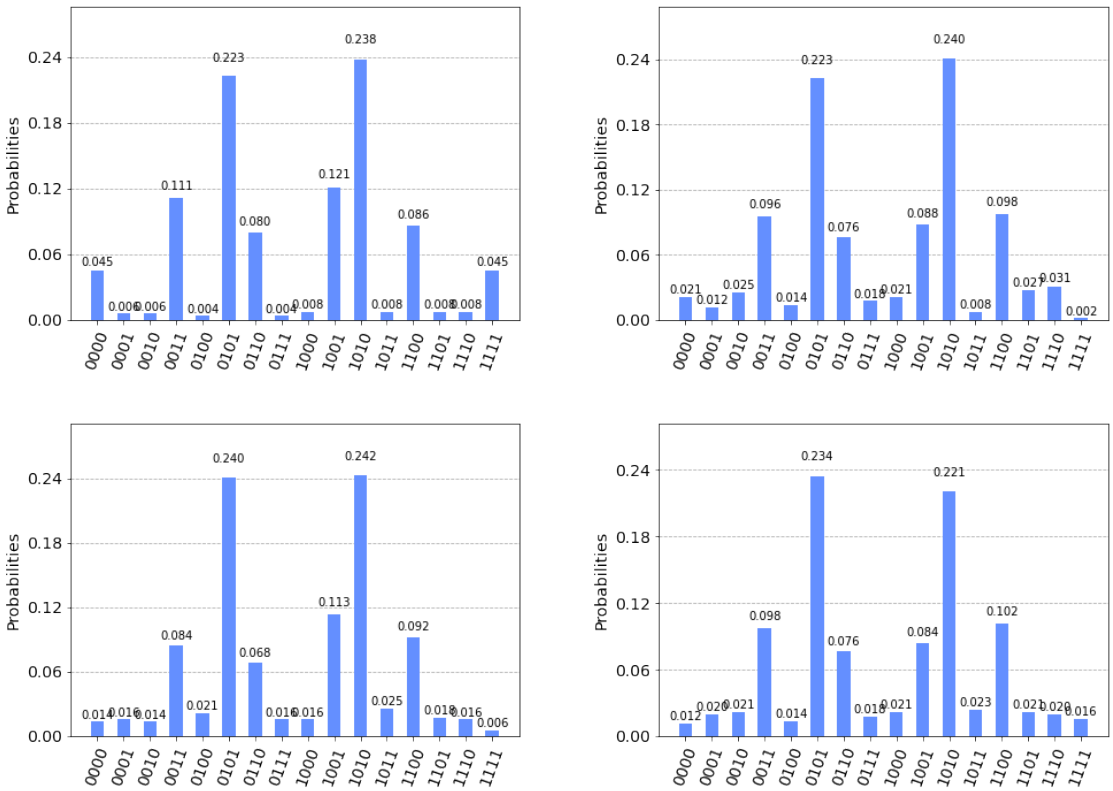


Figure 5.4: The histograms of measurement result of the depth 1 QAOA for MaxCut for optimizers (ordered from left to right): Simple Gradient Descent, Scheduled Gradient Descent, RMSProp, COBYLA

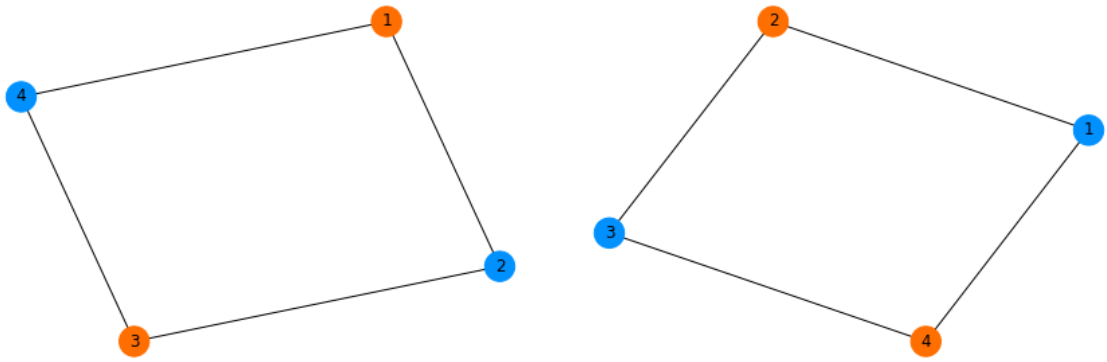


Figure 5.5: The partitions of the simple graph, that solve the MaxCut problem, coloured according to the solution bitstrings produced by the algorithm

analyse the performance of the optimizers of this example problem one hundred repeated runs of the optimizers with required high precision were performed and logged the number of cycles necessary to converge with set precision. In order to perform comparison of the selected methods the number of

shots was set to 300 equally. The average number of iterations are listed in the following table 5.1.

Method	0.01	0.02	0.03	0.04	0.05
Simple GD	27.05	19.55	17.05	11.20	10.00
Scheduled GD	16.95	12.60	8.50	7.85	7.60
COBYLA	18.30	17.05	14.55	14.20	13.95
RMSProp	17.80	11.20	10.75	9.70	6.15

Table 5.1: Average number of iterations needed to reach convergence with tolerance set to value in top row using 300 shots for the chosen optimizers listed in the first column

5.1.2 VQE: Basic implementation

The application of VQE in quantum chemistry context is beyond the scope of basic examples. For this reason we will consider the problem of finding the eigenvalue of chosen matrix.

$$M = \begin{pmatrix} 0 & 0 & -0.5 \\ 0 & 0.75 & -0.5 \\ -0.5 & -0.5 & -0.25 \end{pmatrix}, \quad \lambda_1 = -0.75, \quad \lambda_2 = 0.25, \quad \lambda_3 = 1 \qquad \text{Eq (5.1.1)}$$

To solve this problem, the simplified VQE was used, by this it is meant that for this problem there is no need for complicated choice of base or encoding. Nevertheless, all basic components are present and the principle of VQE can be easily seen in action. To link this demonstration to previously explained theory, we emphasize that this implementation relies upon the HEA ansatz, which was introduced in definition 3.11, and the RMSProp optimizer[4], presented in 3. For comparison we have included a non-gradient optimizer, the COBYLA method from the `scipy` python package. This problem also illustrates one complicated aspect of VQE, in order to work with the given Hamiltonian, e.g. matrix M from Eq (5.1.1), it first needs to be converted into a sum of Pauli strings which can then be implemented on the circuit. This conversion process for studied settings was already outlined in Chapter 3. For an "random" matrix such as M this process can be difficult. Another problem is linked with the form of Hamiltonian. The sum of Pauli strings contains coefficients for all summands and as such they cannot be implemented directly in the circuit, because the gates would lose the unitarity. Computation of the overall expectation value accounting for all coefficients is done through circuit analysis and weighting of the results, which follows from the rules of tensor product and the description of measurement in quantum physics. A sufficiently general implementation of an algorithm is beyond the scope of this project. For these reasons the following implementations use the tools that are provided by the Qiskit.

With this knowledge we can now proceed to present the details of the implementation. The new functions are the HEA block constructing function, which adds a simple HEA block to the circuit of arbitrary size upon call. The next new function is the VQE running function which is called to perform whole quantum routine of VQE with set parameters and return the computer expectation value. The computation of expectation value is based on successive repetitions of the circuit and measurement. This means that one call of VQE running functions is equivalent to number of circuit executions set by the parameter `shots`. To illustrate the working of our algorithm implementation we include the following excerpts from the source code.

[4]Other optimizers and their results can be reviewed in the Jupyter notebooks which can be found on the project Gitlab repository.

Listing 5.5: The function constructing the HEA ansatz in the circuit, based on figure 3.5

```
def add_HEA(parameters, circuit):
    p = int(len(parameters)/2)
    beta = parameters[:p]
    gamma = parameters[p:]
    for it in range(0,len(beta)):
        circuit.rx(beta[it], it)
    for it in range(0,len(gamma)):
        circuit.ry(gamma[it], it)
    for i in range(0,len(beta)-1):
        circuit.cx(i, i+1)
```

Listing 5.6: The function used to perform a VQE circuit for a given Hamiltonian

```
def run_vqe(parameters, layers, hamiltonian_op, shots):
    NQUBITS = int(len(parameters)/2)
    # Create quantum circuit
    QC = QuantumCircuit(NQUBITS)
    add_HEA(parameters,QC)
    backend=Aer.get_backend('aer_simulator', device='gpu')
    exp_converter = ExpectationFactory.build(hamiltonian_op, backend)
    measurable_expression =  ~StateFn(hamiltonian_op) @ StateFn(QC)
    expect_op = exp_converter.convert(measurable_expression)
    sampled_op = CircuitSampler(backend).convert(expect_op)
    expectation_value = sampled_op.eval().real
    return expectation_value
```

Listing 5.7: Example of the use of RMSProp optimizer to find eigenvalues using VQE

```
GRAD = get_gradient_vqe(1, H_op, 300)
result3, num_loops = RMSProp(GRAD, [1.0,1.0,1.0,1.0], 0.29, 0.2 , 200, EPSILON=0.1)

print('Result␣parameters', result3)
print('Number␣of␣gradient␣calls', num_loops)
print('Ground␣state␣approximation:␣', run_vqe(result3,1, H_op, 400))
```

The results for the RMSProp and COBYLA optimizers are summarized in the following figures.

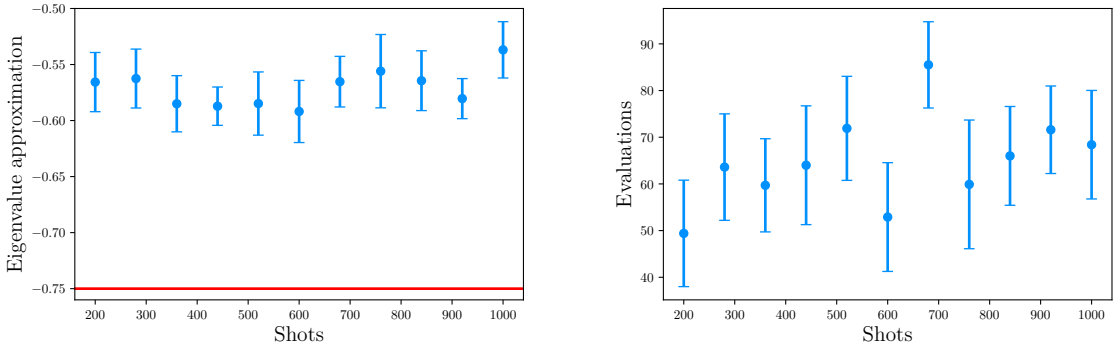


Figure 5.6: The figure on the left shows the graph of the approximations of eigenvalue found using the RMSProp optimizer. The figure on the right gives the respective number of iterations needed for convergence

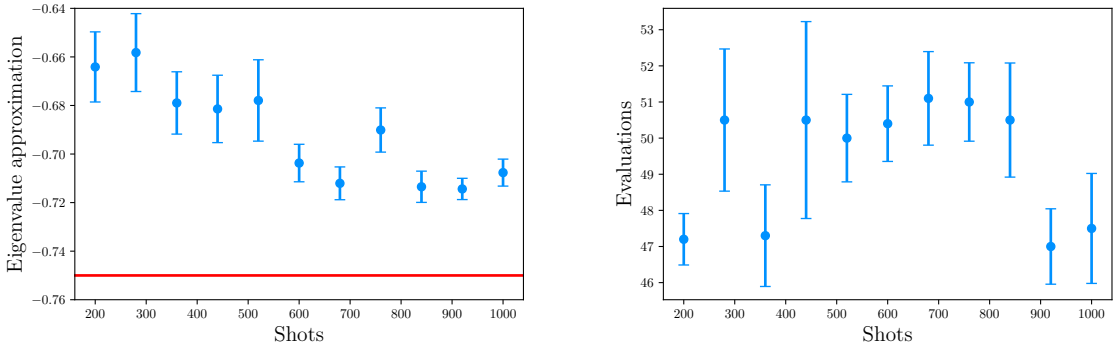


Figure 5.7: The figure on the left shows the graph of the approximations of eigenvalue found using the COBYLA optimizer. The figure on the right gives the respective number of iterations needed for convergence

Qiskit built-in implementations

Both QAOA and VQE have since their discovery reached recognition as perspective quantum algorithms that may even reach quantum supremacy in certain situations[73, 74]. It is not surprising that an extensive SDK such as Qiskit includes its own implementation of the algorithms. The implementations `QAOA` and `VQE` are a part of `qiskit.algorithms` module, and they represent an algorithms template, which is completed by adding an ansatz and an optimizer[5]. Such implementations have the advantage of using complex Python code create robust algorithm, which does not suffer many problems, which are present in our simple implementation, such as initial point handling. The main disadvantages of the built-in algorithms are the extensive code, which obscures the inner working of the algorithm, and the dependency on optimizers, which need to fullfill strict requirements, making it unnecessarily demanding to experiment with new optimizer code or use code from external library. The optimizers offered by Qiskit include the `GradientDescent`[6], `SPSA` and `COBYLA`. An example of use of built-in VQE with the HEA block used in our VQE example can be found below.

Listing 5.8: Example of use of Qiskit built-in implementation of VQE (using SPSA optimizer) with the detailed HEA block construction

```
p = ParameterVector('p', 4)
```

[5] In case of VQE is it important to choose the expectation value convertor, as this choice impacts the presence of noise in the calculation.

[6] It is worh noting that this gradient descent does not keep the learning rate η constant. This maked this optimizer similar to scheduled gradient descent with multiple improvements, which may not be desired by the user.

```
hamiltonian_op = H_op
shots = 300
NQUBITS = 2
# Create quantum circuit
QC = QuantumCircuit(NQUBITS)
for it in range(0,2):
    QC.rx(p[it], it)
for it in range(0,2):
    QC.ry(p[it+2], it)
QC.cx(0, 1)
QC.draw()

qinstance = QuantumInstance(backend, shots = 400, seed_simulator=2)
vqe = VQE(ansatz=QC, initial_point = [1.0, 1.0, 1.0, 1.0], optimizer=SPSA(),\
          quantum_instance=qinstance)
result = vqe.compute_minimum_eigenvalue(H_op)
print(result)
```

For more detailed examples of use of built-in VQE in context of the problems above refer to the project Jupyter notebooks[7].

5.2 Applications

In this final section of the work we present a practical non-trivial problem for each studied algorithm and present its solution along with analysis of the implementation and its performance. The problem were chosen to be related to previously mentioned topics, so the these implementations can be considered to be their direct extension. The code from the previous section 5.1, will be partially reused, in to avoid repetition of essentially same code, only the relevant code excerpt will be referenced.

5.2.1 QAOA application

Relevant problem

The QAOA is as a combinatorial algorithm has a wide range of potential application. Only for the MaxCut combinatorial problem, which we have already mentioned, exist a variety of problems that can be mapped to it, including machine scheduling, QUBO problems, image recognition, etc.[75]. One such problems is the *clustering* method for unsupervised machine learning[8].

[7] Which can be found at the project repository linked in the Appendix.
[8] The term is in detail explained in [76]

Definition 5.1. The term *clustering*, sometimes referred to as *cluster analysis*, stands for the task of grouping the elements of a (training) set into groups, so that object which share a group are similar to each other.[76]

The notion of similarity here may refer to various characteristics. In this project we will consider sets of k-dimesional vectors (points), for this reason the only considered characteristic is the relative distance of the elements in space. This application is heavily inspired by the paper [23], where the idea of using QAOA for clustering. To be specific in this section, similarly to the paper we will consider only bi-clustering, i.e. the elements will be categorized into only two groups. The following algorithm puts the clustering into context of MaxCut Now we can address the question, how much does the previously

Algorithm 6: MaxCut bi-clustering algorithm, inspired by [75]

Data: Dataset $S \subset \mathbb{R}^{\aleph} = \{p_i | i \in \hat{N}\}$, metric $\rho(\cdot, \cdot)$
Result: Grouping of elements of S into S_1 and S_2
begin
  for (i, j) ***in*** $\hat{N} \otimes \hat{N}$ **do**
    $C_{ij} \leftarrow \rho()$
  $H_P \leftarrow$ `Encode`(Objectivefunction, weightmatrix$C$)
  bitstring $\leftarrow$ `QAOA`(H_P)
  for i ***in*** $\hat{N}$ **do**
    if *bitstring[i] == 1* **then**
      S_1.append(p_i)
    else
      S_2.append(p_i)

implemented MaxCut on graph and bi-clustering have in common. The answer follows from observing the differences of the problems. The main differences are the presence of imaginary line connecting every point, now taken as a vertex, with every other point, and the fact that the lines have various lengths. The problem can be seen as 2 MaxCut on graph with weighted edges, where weight stands for the distance. The first difference can be implemented easily. The second difference means that the objective value calculator and in consequence the expectation value calculator need to be modified. Weighting also affects the problem Hamiltonian which now takes form

$$H_P = \sum_{\langle u,v \rangle} w_{\langle u,v \rangle} \frac{1}{2}(1 - Z_u Z_v), \qquad \text{Eq (5.2.1)}$$

where $w_{\langle u,v \rangle}$ represents the weight. This change does not have a large impact on the implementation, since the effect of identities is ignored and the multiplication of $Z_u Z_v$ by weight only changes the values of the optimal parameters.

Before the algorithms can be applied, the dataset is randomly generated. This work only studies the problem for $S \in \mathbb{R}^2$. The figure below shows the used dataset. In order to check the QAOA result we also generate the correct solution using brute force approach.

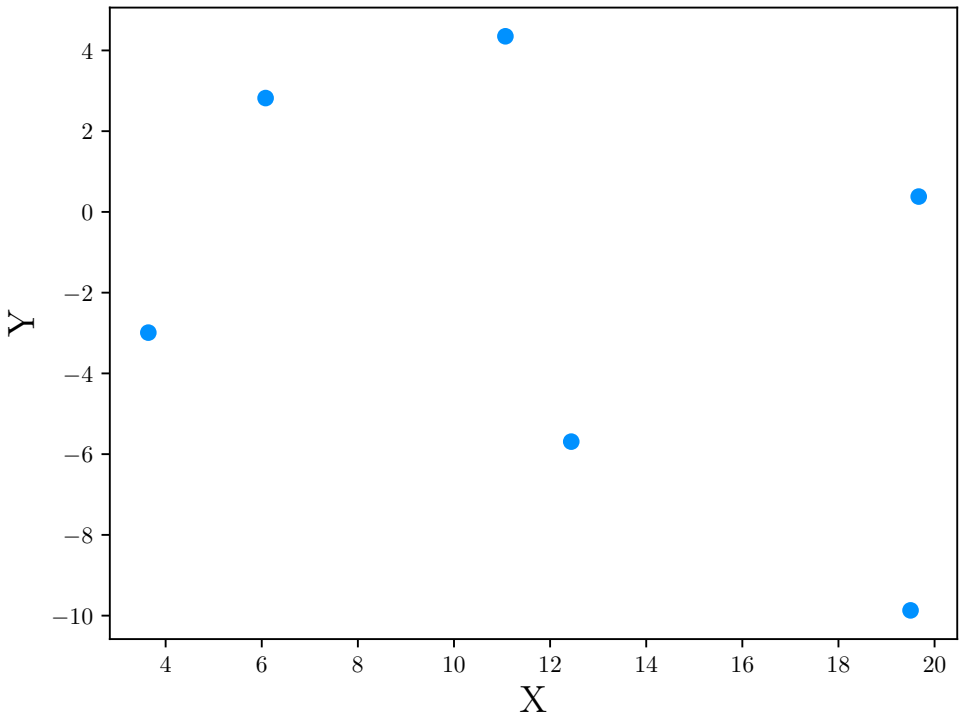


Figure 5.8: Randomly generated dataset, that is to be grouped by bi-clustering through QAOA

Applications of algorithm

To implement the algorithm, the code for simple QAOA was adapted to be in order with the algorithm on figure 6, by changing the expectation value calculation and the QAOA constructing function. For this problem the depth 2 QAOA was used, this means that the used circuit has two layers and four parameters instead of two. The main reason for this was the need for high precision of parameters for the depth one QAOA, which lead to unnecessary amount of shots needed for solution. The first step is to compute the weight matrix, containing the distances between points, which is then used by the expectation value calculator[9]. The rest of the code, apart from addition of result string picker, remains the same.

Analysis of application

In this subsection we briefly review our results of the implementation of QAOA for bi-clustering of dataset. First let us emphasize that by setting the parameters of the optimizers accordingly, the correct result was reached for both project original RMSProp optimizer and for the `scipy` package COBYLA optimizer. To study the behaviour of problem solution, the solving procedure was repeated for various number of shots, as this parameter together with the number of algorithms iterations is key for the need resource of the algorithms.

[9]Note that for this implementation, the classical expectation value calculation is not used, instead a "calculator" function is used, which returns a value that can be used to approximate the expectation value

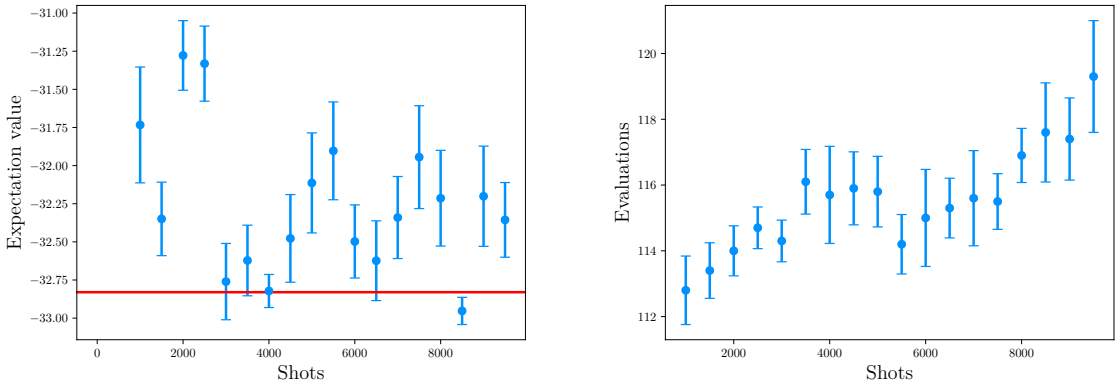


Figure 5.9: The figure on the left shows the chart of expectation value approximation found using the COBYLA optimizer. The figure on the right gives the respective number of iterations needed for convergence

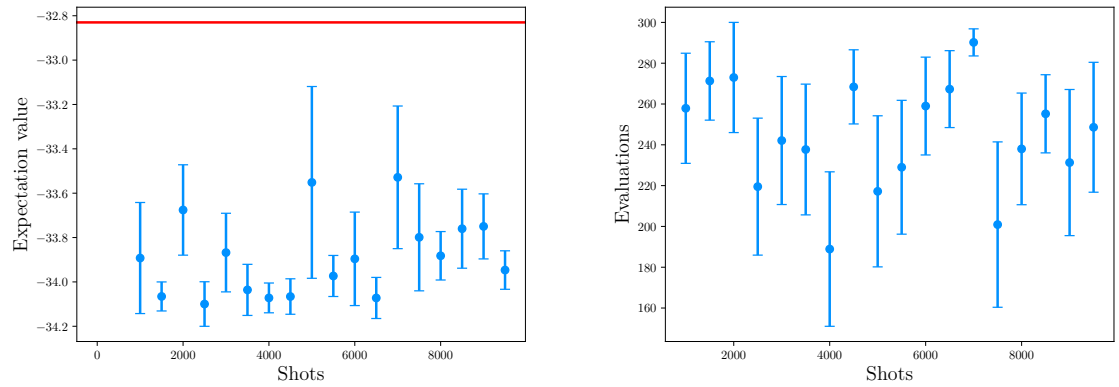


Figure 5.10: The figure on the left shows the chart of expectation value approximation found using the RMSProp optimizer. The figure on the right gives the respective number of iterations needed for convergence

Chapter 6

Conclusion

This project aims to give a comprehensive and complete overview of the topic of elementary programming of quantum computers and recent quantum algorithms. Particularly this work focuses on the Quantum Approximate Optimization Algorithm (QAOA) and the Variational Quantum Eigensolver (VQE).

The first chapter gives a transparent introduction to the topic of quantum algorithms by explaining the basic terminology, theoretical concepts and methods. In this chapter are also summarized few identities important in context of quantum programming. The quantum circuit formalism is illustrated using a short list of most important quantum gates, including the multiqubit gates. The matrix forms of gates are also given in this chapter. The last part of the chapter opens the topic of quantum algorithms, by explaining their importance and the outlining the characteristics of the studied algorithms.

The following chapter is completely dedicated to QAOA algorithm. The beginning introduces the combinatorial problems and approximations linked to QAOA. The first section gives an extensive list of definitions of terms that are linked to QAOA and finally giving the rigorous definition of the algorithms itself. The definitions are complemented by schematics of the algorithms and a number of equations. The properties of QAOA are studied in the next sections. Basic data are given on the general QAOA, and more extensive overview of properties is given for fixed depth QAOA. Particularly the fixed depth QAOA is studied in context of the MaxCut problem, which is defined and explained in depth in this chapter, and the problems which can be described by a Boolean objective functions. Few theorems for these objective functions and their representation by a problem Hamiltonian are presented and proven. The properties of fixed depth QAOA are shown and proven for graphs of bound degree and using the technique of Pauli Solver. The chapter ends with a short overview of algorithms related to QAOA.

In the third chapter the VQE is presented. The terms defining the problem are given in the beginning along with a brief description of the algorithm and its steps. The second more extensive section describes the individual components of VQE. First the preparatory steps, such as the choice of basis according to application, needed to build a working VQE implementation are explained. Next the Hamiltonian and its construction are shown in detail. The following section present the problems of operator encoding and multiple solutions, methods of VQE optimization and the question of ansatz choice and parametrization. The chapter is closed by an overview of optimization methods that can be used in the optimization parameters during the algorithms loop.

The fourth chapter reviews the currently available quantum programming tools, including the low level quantum instruction sets, quantum programming languages and finally the quantum software development kits (SDK). The SDKs are compared in a transparent table listing their features. The focus is on the Python based Qiskit SDK by IBM Research which is one of the most feature-complete SDKs available. Next the components of Qiskit are explained. The chapter ends with an example of a Qiskit code and the quantum circuit generated using the code.

The final fifth chapter deals with implementation of the algorithms, first on two basic problems and two more practical problems. The solutions of the problems are reached using the simple implementations of the algorithms written for the project and also for comparison are included solutions using the Qiskit built-in implementations. The implementations rely on the optimizers which are coded according to their respective definitions in the third chapter. The project original code is available for review on the project online repository linke in the appendix.

Appendix A

Additional information

A.1 Acronyms

This part of appendix contains comprehensive list of acronyms used in the project and their meaning.

QAOA	Quantum Approximate Optimization Algorithm or Quantum Alternating Operator Ansatz
VQA	Variationa Quantum Algorithm
VQE	Variationa Quantum Eigensolver
CC	Coupled Cluster
UCC	Unitary Coupled Cluster
HEA	Hardware Efficient Ansatz
QAA	Quantum Adiabatic Algorithm
NISQ	Noisy Intermediate-Scale Quantum (devices)
SPSA	Simultaneous Perturbation Stochastic Approximation
FDSA	Finite Difference Stochastic Approximation
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation (stochastic approximation method)
BFGS	Broyden–Fletcher–Goldfarb–Shanno (algorithm)
QRAM	Quantum Random Access Memory
QUBO	Quadratic unconstrained binary optimization
QASM	Quantum Assembly Language
QML	Quantum Meta Language
SDK	Software Development Kit
REPL	Read-eval-print loop

Source code

Due to the extent of used code in this project, a decision was made to omit the full source code from the text of the work. In order to keep the code available, it was uploaded to Gitlab repository. Below a hyperlink and a QR code is presented which lead to the this repository.
Link to project repository: `https://gitlab.com/fjfi_studies/bc_project`

Figure A.1: QR code leading to the project repository

Appendix B

Bibliography

1. FEYNMAN, Richard P. Simulating physics with computers. *Int. j. Theor. phys.* 1982, vol. 21, no. 6/7.
2. RIEFFEL, Eleanor; POLAK, Wolfgang. *Quantum computing: a gentle introduction*. Cambridge, Mass: The MIT Press, 2011. Scientific and engineering computation. ISBN 9780262015066. OCLC: ocn641998800.
3. HIDARY, Jack D. A Brief History of Quantum Computing. In: *Quantum Computing: An Applied Approach*. Cham: Springer International Publishing, 2019, pp. 11–16. ISBN 978-3-030-23922-0. Available from DOI: 10.1007/978-3-030-23922-0_2.
4. NIELSEN, Michael A.; CHUANG, Isaac L. *Quantum computation and quantum information*. 10th anniversary ed. Cambridge ; New York: Cambridge University Press, 2010. ISBN 9781107002173.
5. BERRY, Dominic W.; KIEFEROVÁ, Mária; SCHERER, Artur, et al. Improved techniques for preparing eigenstates of fermionic Hamiltonians. *npj Quantum Information* [online]. 2018, vol. 4, no. 1, pp. 1–7 [visited on 2022-05-29]. ISSN 2056-6387. Available from DOI: 10.1038/s41534-018-0071-5.
6. TILLY, Jules; CHEN, Hongxiang; CAO, Shuxiang, et al. *The Variational Quantum Eigensolver: a review of methods and best practices* [online]. 2021-11 [visited on 2022-05-13]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.2111.05176. arXiv:2111.05176 [quant-ph] type: article.
7. PRESKILL, John. Quantum Computing in the NISQ era and beyond. *Quantum* [online]. 2018, vol. 2, p. 79 [visited on 2022-05-14]. Available from DOI: 10.22331/q-2018-08-06-79.
8. ARUTE, Frank; ARYA, Kunal; BABBUSH, Ryan, et al. Quantum supremacy using a programmable superconducting processor. *Nature* [online]. 2019, vol. 574, no. 7779, pp. 505–510 [visited on 2022-05-14]. ISSN 1476-4687. Available from DOI: 10.1038/s41586-019-1666-5.
9. WU, Yulin; BAO, Wan-Su; CAO, Sirui, et al. Strong Quantum Computational Advantage Using a Superconducting Quantum Processor. *Physical Review Letters* [online]. 2021, vol. 127, no. 18, p. 180501 [visited on 2022-05-14]. Available from DOI: 10.1103/PhysRevLett.127.180501.
10. KRUSE, Regina; HAMILTON, Craig S.; SANSONI, Linda, et al. Detailed study of Gaussian boson sampling. *Physical Review A* [online]. 2019, vol. 100, no. 3, p. 032326 [visited on 2022-07-04]. Available from DOI: 10.1103/PhysRevA.100.032326.

11. PRESKILL, John. *Quantum computing and the entanglement frontier* [online]. 2012-11 [visited on 2022-05-14]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1203.5813. arXiv:1203.5813 [cond-mat, physics:quant-ph] type: article.

12. CEREZO, M.; ARRASMITH, Andrew; BABBUSH, Ryan, et al. Variational quantum algorithms. *Nature Reviews Physics*. 2021, vol. 3, no. 9, pp. 625–644. Available from DOI: 10.1038/s42254-021-00348-9.

13. CHOI, Jaeho; KIM, Joongheon. A Tutorial on Quantum Approximate Optimization Algorithm (QAOA): Fundamentals and Applications. In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. 2019, pp. 138–142. Available from DOI: 10.1109/ICTC46691.2019.8939749.

14. MCCLEAN, Jarrod R.; HARRIGAN, Matthew P.; MOHSENI, Masoud, et al. Low-Depth Mechanisms for Quantum Optimization. *PRX Quantum*. 2021, vol. 2, p. 030312. Available from DOI: 10.1103/PRXQuantum.2.030312.

15. AUSIELLO, Giorgio; MARCHETTI-SPACCAMELA, Alberto; CRESCENZI, Pierluigi, et al. *Complexity and Approximation*. Springer Berlin Heidelberg, 1999. Available from DOI: 10.1007/978-3-642-58412-1.

16. FARHI, Edward; GOLDSTONE, Jeffrey; GUTMANN, Sam. *A Quantum Approximate Optimization Algorithm*. 2014. Available from arXiv: 1411.4028 [quant-ph].

17. PILLAI, S.U.; SUEL, T.; CHA, Seunghun. The Perron-Frobenius theorem: some of its applications. *IEEE Signal Processing Magazine*. 2005, vol. 22, no. 2, pp. 62–75. ISSN 1558-0792. Available from DOI: 10.1109/MSP.2005.1406483.

18. NINIO, F. A simple proof of the Perron-Frobenius theorem for positive symmetric matrices. *Journal of Physics A: Mathematical and General* [online]. 1976, vol. 9, no. 8, pp. 1281–1282 [visited on 2022-05-15]. ISSN 0305-4470. Available from DOI: 10.1088/0305-4470/9/8/017.

19. HADFIELD, Stuart. *Quantum Algorithms for Scientific Computing and Approximate Optimization*. 2018. Available from arXiv: 1805.03265 [quant-ph].

20. KAO, Ming-Yang (ed.). *Encyclopedia of algorithms*. Second edition. New York, NY: Springer Reference, 2016. ISBN 9781493928637 9781493928651.

21. BARAHONA, Francisco; GRÖTSCHEL, Martin; JÜNGER, Michael, et al. An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design. *Operations Research*. 1988, vol. 36, no. 3, pp. 493–513. Available from DOI: 10.1287/opre.36.3.493.

22. M. R. GAREY, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W H FREEMAN WORTH PUB 3PL, 1979. ISBN 0716710455. Available also from: https://www.ebook.de/de/product/3637119/m_r_garey_d_s_johnson_computers_and_intractability_a_guide_to_the_theory_of_np_completeness.html.

23. CLINTON, Laura; BAUSCH, Johannes; CUBITT, Toby. Hamiltonian Simulation Algorithms for Near-Term Quantum Hardware. *Nature Communications* [online]. 2021, vol. 12, no. 1, p. 4989 [visited on 2022-06-02]. ISSN 2041-1723. Available from DOI: 10.1038/s41467-021-25196-0. arXiv:2003.06886 [quant-ph].

24. PAREKH, Ojas D.; RYAN-ANDERSON, Ciaran; GHARIBIAN, Sevag. *Quantum Optimization and Approximation Algorithms*. [Online]. 2019-01 [visited on 2022-05-14]. Tech. rep. Available from DOI: 10.2172/1492737.

25. FARHI, Edward; GOLDSTONE, Jeffrey; GUTMANN, Sam. *A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem* [online]. 2015-06 [visited on 2022-05-17]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1412.6062. arXiv:1412.6062 [quant-ph] type: article.

26. FARHI, Edward; GOLDSTONE, Jeffrey; GUTMANN, Sam, et al. *Quantum Computation by Adiabatic Evolution*. 2000. Available from arXiv: quant-ph/0001106 [quant-ph].

27. FARHI, Edward; GOLDSTONE, Jeffrey; GUTMANN, Sam. *Quantum Adiabatic Evolution Algorithms versus Simulated Annealing*. 2002. Available from arXiv: quant-ph/0201031 [quant-ph].

28. HADFIELD, Stuart; WANG, Zhihui; O'GORMAN, Bryan, et al. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. *Algorithms*. 2019, vol. 12, no. 2. ISSN 1999-4893. Available from DOI: 10.3390/a12020034.

29. PERUZZO, Alberto; MCCLEAN, Jarrod; SHADBOLT, Peter, et al. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*. 2014, vol. 5, no. 1, p. 4213. ISSN 2041-1723. Available from DOI: 10.1038/ncomms5213.

30. HEIFETZ, Alexander (ed.). *Quantum Mechanics in Drug Discovery*. Springer US, 2020. ISBN 978-1-0716-0281-2. Available from DOI: 10.1007/978-1-0716-0282-9.

31. CONTINENTINO, Mucio Amado. *Key Methods and Concepts in Condensed Matter Physics*. Institute of Physics Publishing, 2021. Available also from: https://www.ebook.de/de/product/40701199/mucio_amado_continentino_key_methods_and_concepts_in_condensed_matter_physics.html.

32. LORDI, Vincenzo; NICHOL, John M. Advances and opportunities in materials science for scalable quantum computing. *MRS Bulletin*. 2021, vol. 46, no. 7, pp. 589–595. Available from DOI: 10.1557/s43577-021-00133-0.

33. CAO, Y.; ROMERO, J.; ASPURU-GUZIK, A. Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*. 2018, vol. 62, no. 6, 6:1–6:20. Available from DOI: 10.1147/jrd.2018.2888987.

34. MICELI, Raffaele; MCGUIGAN, Michael. Effective matrix model for nuclear physics on a quantum computer. In: *2019 New York Scientific Data Summit (NYSDS)*. IEEE, 2019. Available from DOI: 10.1109/nysds.2019.8909693.

35. ARFKEN, George B. *Mathematical Methods for Physicists*. Elsevier Science & Techn., 2013. ISBN 9781483277820. Available also from: https://www.ebook.de/de/product/23273197/george_b_arfken_mathematical_methods_for_physicists.html.

36. YSERENTANT, Harry. A Short Theory of the Rayleigh–Ritz Method. *Computational Methods in Applied Mathematics*. 2013, vol. 13, no. 4, pp. 495–502. Available from DOI: 10.1515/cmam-2013-0013.

37. LLOYD N. TREFETHEN, David Bau I. I. I. *Numerical Linear Algebra*. CAMBRIDGE, 1997. ISBN 0898713617. Available also from: https://www.ebook.de/de/product/6906297/lloyd_n_trefethen_david_bau_iii_numerical_linear_algebra.html.

38. MCCLEAN, Jarrod R.; ROMERO, Jonathan; BABBUSH, Ryan, et al. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* [online]. 2016, vol. 18, no. 2, p. 023023 [visited on 2022-05-06]. ISSN 1367-2630. Available from DOI: 10.1088/1367-2630/18/2/023023.

39. SZABO, Attila; OSTLUND, Neil. *Modern quantum chemistry*. Dover Publications, 2012. Available also from: https://www.ebook.de/de/product/21951617/attila_szabo_neil_s_ostlund_modern_quantum_chemistry.html.

40. LEE, Joonho; HUGGINS, William J.; HEAD-GORDON, Martin, et al. Generalized Unitary Coupled Cluster Wave functions for Quantum Computation. *Journal of Chemical Theory and Computation* [online]. 2019, vol. 15, no. 1, pp. 311–324 [visited on 2022-05-27]. ISSN 1549-9618. Available from DOI: 10.1021/acs.jctc.8b01004.

41. HELGAKER, Trygve; OLSEN, Jeppe; JORGENSEN, Poul. *Molecular Electronic-Structure Theory*. Wiley-Blackwell, 2013. ISBN 1118531477. Available also from: https://www.ebook.de/de/product/19812715/trygve_helgaker_jeppe_olsen_poul_jorgensen_molecular_electronic_structure_theory.html.

42. MCARDLE, Sam; ENDO, Suguru; ASPURU-GUZIK, Alán, et al. Quantum computational chemistry. *Reviews of Modern Physics*. 2020, vol. 92, no. 1, p. 015003. Available from DOI: 10.1103/revmodphys.92.015003.

43. JENSEN, Frank. *Introduction to Computational Chemistry*. John Wiley & Sons, 2016. Available also from: https://www.ebook.de/de/product/28368245/frank_jensen_introduction_to_computational_chemistry.html.

44. ABRAMS, Daniel S.; LLOYD, Seth. Simulation of Many-Body Fermi Systems on a Universal Quantum Computer. *Physical Review Letters* [online]. 1997, vol. 79, no. 13, pp. 2586–2589 [visited on 2022-05-29]. Available from DOI: 10.1103/PhysRevLett.79.2586.

45. MOLL, Nikolaj; BARKOUTSOS, Panagiotis; BISHOP, Lev S., et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology* [online]. 2018, vol. 3, no. 3, p. 030503 [visited on 2022-05-31]. ISSN 2058-9565. Available from DOI: 10.1088/2058-9565/aab822.

46. BRAVYI, Sergey; GAMBETTA, Jay M.; MEZZACAPO, Antonio, et al. *Tapering off qubits to simulate fermionic Hamiltonians* [online]. 2017-01 [visited on 2022-06-02]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1701.08213. arXiv:1701.08213 [quant-ph] version: 1 type: article.

47. AKSHAY, V.; PHILATHONG, H.; MORALES, M. E. S., et al. Reachability Deficits in Quantum Approximate Optimization. *Physical Review Letters* [online]. 2020, vol. 124, no. 9, p. 090504 [visited on 2022-05-03]. ISSN 0031-9007, ISSN 1079-7114. Available from DOI: 10.1103/PhysRevLett.124.090504. arXiv: 1906.11259.

48. YORDANOV, Yordan S.; ARMAOS, V.; BARNES, Crispin H. W., et al. Qubit-excitation-based adaptive variational quantum eigensolver. *Communications Physics* [online]. 2021, vol. 4, no. 1, p. 228 [visited on 2022-06-03]. ISSN 2399-3650. Available from DOI: 10.1038/s42005-021-00730-0. arXiv:2011.10540 [quant-ph].

49. SEELEY, Jacob T.; RICHARD, Martin J.; LOVE, Peter J. The Bravyi-Kitaev transformation for quantum computation of electronic structure. *The Journal of Chemical Physics* [online]. 2012, vol. 137, no. 22, p. 224109 [visited on 2022-06-03]. ISSN 0021-9606. Available from DOI: 10.1063/1.4768229.

50. HOLMES, Zoë; SHARMA, Kunal; CEREZO, M., et al. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum* [online]. 2022, vol. 3, no. 1, p. 010313 [visited on 2022-06-05]. ISSN 2691-3399. Available from DOI: 10.1103/PRXQuantum.3.010313. arXiv:2101.02138 [quant-ph, stat].

51. CEREZO, M.; SONE, Akira; VOLKOFF, Tyler, et al. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications* [online]. 2021, vol. 12, no. 1, p. 1791 [visited on 2022-06-02]. ISSN 2041-1723. Available from DOI: 10.1038/s41467-021-21728-w.

52. ANAND, Abhinav; SCHLEICH, Philipp; ALPERIN-LEA, Sumner, et al. A Quantum Computing View on Unitary Coupled Cluster Theory. *Chemical Society Reviews* [online]. 2022, vol. 51, no. 5, pp. 1659–1684 [visited on 2022-06-06]. ISSN 0306-0012, ISSN 1460-4744. Available from DOI: 10.1039/D1CS00932J. arXiv:2109.15176 [physics, physics:quant-ph].

53. SCHULD, Maria; BERGHOLM, Ville; GOGOLIN, Christian, et al. Evaluating analytic gradients on quantum hardware. *Physical Review A* [online]. 2019, vol. 99, no. 3, p. 032331 [visited on 2022-06-08]. ISSN 2469-9926, ISSN 2469-9934. Available from DOI: 10.1103/PhysRevA.99.032331. arXiv:1811.11184 [quant-ph].

54. POLYAK, Boris T. *Introduction to Optimization*. New York : Optimization Software, Inc., 1987.

55. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization* [online]. 2017-01 [visited on 2022-06-09]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1412.6980. arXiv:1412.6980 [cs] type: article.

56. PARRISH, Robert M.; IOSUE, Joseph T.; OZAETA, Asier, et al. *A Jacobi Diagonalization and Anderson Acceleration Algorithm For Variational Quantum Algorithm Parameter Optimization* [online]. 2019-04 [visited on 2022-06-09]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1904.03206. arXiv:1904.03206 [quant-ph] type: article.

57. KNILL, E. Conventions for quantum pseudocode. 1996. Available from DOI: 10.2172/366453.

58. SELINGER, Peter. A Brief Survey of Quantum Programming Languages. In: *Functional and Logic Programming*. Springer Berlin Heidelberg, 2004, pp. 1–6. Available from DOI: 10.1007/978-3-540-24754-8_1.

59. MAYMIN, Philip. *Extending the Lambda Calculus to Express Randomized and Quantumized Algorithms* [online]. 1997-01 [visited on 2022-06-11]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.quant-ph/9612052. arXiv:quant-ph/9612052 type: article.

60. WIKIPEDIA CONTRIBUTORS. *Quantum programming — Wikipedia, The Free Encyclopedia*. 2022. Available also from: https://en.wikipedia.org/w/index.php?title=Quantum_programming&oldid=1083094034. [Online; accessed 11-June-2022].

61. HEIM, Bettina; SOEKEN, Mathias; MARSHALL, Sarah, et al. Quantum programming languages. *Nature Reviews Physics*. 2020, vol. 2, no. 12, pp. 709–722. Available from DOI: 10.1038/s42254-020-00245-7.

62. CROSS, Andrew W.; BISHOP, Lev S.; SMOLIN, John A., et al. *Open Quantum Assembly Language* [online]. 2017-07 [visited on 2022-06-11]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.1707.03429. arXiv:1707.03429 [quant-ph] type: article.

63. IBM RESEARCH. *Qiskit* [online]. [N.d.] [visited on 2022-07-01]. Available from: https://github.com/Qiskit.

64. *Piquasso* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/Budapest-Quantum-Computing-Group/piquasso. original-date: 2021-05-08T09:51:19Z.

65. *PyQuil: Quantum programming in Python* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/rigetti/pyquil. original-date: 2017-01-09T21:30:22Z.

66. *quantumlib/Cirq* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/quantumlib/Cirq. original-date: 2017-12-14T23:41:49Z.

67. *dwavesystems/dwave-ocean-sdk* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/dwavesystems/dwave-ocean-sdk. original-date: 2017-11-21T19:24:37Z.

68. *XanaduAI/strawberryfields* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/XanaduAI/strawberryfields. original-date: 2018-03-26T14:38:39Z.

69. *PennyLaneAI/pennylane* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/PennyLaneAI/pennylane. original-date: 2018-04-17T16:45:42Z.

70. *Microsoft Quantum Development Kit Samples* [online]. 2022 [visited on 2022-07-04]. Available from: https://github.com/microsoft/Quantum. original-date: 2017-11-08T23:24:33Z.

71. WIKIPEDIA CONTRIBUTORS. *Qiskit — Wikipedia, The Free Encyclopedia*. 2022. Available also from: https://en.wikipedia.org/w/index.php?title=Qiskit&oldid=1085866294. [Online; accessed 11-June-2022].

72. WILLE, Robert; VAN METER, Rod; NAVEH, Yehuda. IBM's Qiskit Tool Chain: Working with and Developing for Real Quantum Computers. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 1234–1240. ISSN 1558-1101. Available from DOI: 10.23919/DATE.2019.8715261. ISSN: 1558-1101.

73. FARHI, Edward; HARROW, Aram W. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. *arXiv:1602.07674 [quant-ph]* [online]. 2019 [visited on 2022-05-03]. Available from: http://arxiv.org/abs/1602.07674. arXiv: 1602.07674.

74. DALZELL, Alexander M.; HARROW, Aram W.; KOH, Dax Enshan, et al. How many qubits are needed for quantum computational supremacy? *Quantum*. 2020, vol. 4, p. 264. ISSN 2521-327X. Available from DOI: 10.22331/q-2020-05-11-264.

75. OTTERBACH, J. S.; MANENTI, R.; ALIDOUST, N., et al. *Unsupervised Machine Learning on a Hybrid Quantum Computer* [online]. 2017-12 [visited on 2022-06-19]. Tech. rep. arXiv. Available from: http://arxiv.org/abs/1712.05771. arXiv:1712.05771 [quant-ph] type: article.

76. GOODFELLOW, Ian; BENGIO, Joshua; COURVILLE, Aaron. *Deep Learning*. MIT Press Ltd, 2016. ISBN 0262035618. Available also from: https://www.ebook.de/de/product/26337726/ian_goodfellow_joshua_bengio_aaron_courville_deep_learning.html.

77. MERRIS, Russell. Laplacian matrices of graphs: a survey. *Linear Algebra and its Applications* [online]. 1994, vol. 197-198, pp. 143–176 [visited on 2022-05-08]. ISSN 0024-3795. Available from DOI: 10.1016/0024-3795(94)90486-3.

78. ZHOU, Leo; WANG, Sheng-Tao; CHOI, Soonwon, et al. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *Phys. Rev. X*. 2020, vol. 10, p. 021067. Available from DOI: 10.1103/PhysRevX.10.021067.

79. BENNETT, Charles H.; BERNSTEIN, Ethan; BRASSARD, Gilles, et al. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing* [online]. 1997, vol. 26, no. 5, pp. 1510–1523 [visited on 2022-04-25]. ISSN 0097-5397, ISSN 1095-7111. Available from DOI: 10.1137/S0097539796300933. arXiv: quant-ph/9701001.

80. YANG, Zhi-Cheng; RAHMANI, Armin; SHABANI, Alireza, et al. Optimizing Variational Quantum Algorithms Using Pontryagin's Minimum Principle. *Physical Review X* [online]. 2017, vol. 7, no. 2, p. 021027 [visited on 2022-04-25]. Available from DOI: 10.1103/PhysRevX.7.021027.

81. VERDON, Guillaume; PYE, Jason; BROUGHTON, Michael. A Universal Training Algorithm for Quantum Deep Learning. *arXiv:1806.09729 [quant-ph]* [online]. 2018 [visited on 2022-04-25]. Available from: http://arxiv.org/abs/1806.09729. arXiv: 1806.09729.

82. ARORA, Sanjeev; LUND, Carsten; MOTWANI, Rajeev, et al. Proof verification and the hardness of approximation problems. *Journal of the ACM* [online]. 1998, vol. 45, no. 3, pp. 501–555 [visited on 2022-05-03]. ISSN 0004-5411. Available from DOI: 10.1145/278298.278306.

83. GOEMANS, Michel X.; WILLIAMSON, David P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* [online]. 1995, vol. 42, no. 6, pp. 1115–1145 [visited on 2022-05-03]. ISSN 0004-5411. Available from DOI: 10.1145/227683.227684.

84. CROSSON, Elizabeth; FARHI, Edward; LIN, Cedric Yen-Yu, et al. Different Strategies for Optimization Using the Quantum Adiabatic Algorithm. *arXiv:1401.7320 [quant-ph]* [online]. 2014 [visited on 2022-05-03]. Available from: http://arxiv.org/abs/1401.7320. arXiv: 1401.7320.

85. NANNICINI, Giacomo. Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E* [online]. 2019, vol. 99, no. 1, p. 013304 [visited on 2022-05-06]. Available from DOI: 10.1103/PhysRevE.99.013304.

86. CAO, Yudong; ROMERO, Jonathan; OLSON, Jonathan P., et al. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews*. 2019, vol. 119, no. 19, pp. 10856–10915. Available from DOI: 10.1021/acs.chemrev.8b00803.

87. BHARTI, Kishor; CERVERA-LIERTA, Alba; KYAW, Thi Ha, et al. Noisy intermediate-scale quantum (NISQ) algorithms. *Reviews of Modern Physics* [online]. 2022, vol. 94, no. 1, p. 015004 [visited on 2022-05-19]. ISSN 0034-6861, ISSN 1539-0756. Available from DOI: 10.1103/RevModPhys.94.015004. arXiv:2101.08448 [cond-mat, physics:quant-ph].

88. TAKAGI, Ryuji; ENDO, Suguru; MINAGAWA, Shintaro, et al. *Fundamental limits of quantum error mitigation* [online]. 2022-03 [visited on 2022-05-19]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.2109.04457. arXiv:2109.04457 [quant-ph] version: 2 type: article.

89. BITTEL, Lennart; KLIESCH, Martin. Training variational quantum algorithms is NP-hard. *Physical Review Letters* [online]. 2021, vol. 127, no. 12, p. 120502 [visited on 2022-05-19]. ISSN 0031-9007, ISSN 1079-7114. Available from DOI: 10.1103/PhysRevLett.127.120502. arXiv:2101.07267 [quant-ph].

90. GONTHIER, Jérôme F.; RADIN, Maxwell D.; BUDA, Corneliu, et al. *Identifying challenges towards practical quantum advantage through resource estimation: the measurement roadblock in the variational quantum eigensolver* [online]. 2020-12 [visited on 2022-05-19]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.2012.04001. arXiv:2012.04001 [quant-ph] version: 1 type: article.

91. ELFVING, V. E.; BROER, B. W.; WEBBER, M., et al. *How will quantum computers provide an industrially relevant computational advantage in quantum chemistry?* [Online]. 2020-09 [visited on 2022-05-20]. Tech. rep. arXiv. Available from DOI: 10.48550/arXiv.2009.12472. arXiv:2009.12472 [physics, physics:quant-ph] type: article.

92. STEUDTNER, Mark; WEHNER, Stephanie. Fermion-to-qubit mappings with varying resource requirements for quantum simulation. *New Journal of Physics* [online]. 2018, vol. 20, no. 6, p. 063010 [visited on 2022-06-02]. ISSN 1367-2630. Available from DOI: 10.1088/1367-2630/aac54f.

93. BRAVYI, Sergey B.; KITAEV, Alexei Yu. Fermionic Quantum Computation. *Annals of Physics* [online]. 2002, vol. 298, no. 1, pp. 210–226 [visited on 2022-06-03]. ISSN 0003-4916. Available from DOI: 10.1006/aphy.2002.6254.