Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Applications of the Quantum-Classic Split Pattern in Quantum Computing

Daniel Georg

Course of Study:	Informatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Fabian Bühler, M.Sc. Alexander Mandl, Dipl.-Ing.
Commenced:	April 11, 2022
Completed:	September 20, 2022

Abstract

Quantum computers have the potential to solve certain problems faster or with more precision than classical computers. Therefore, quantum computing saw a rise in popularity and consequently much research is performed to achieve quantum advantage. However, today's quantum computers are not capable of executing most of these algorithms with reasonable instance sizes. To benefit from current hardware, algorithms are split into a classical part and a quantum part, such that recent quantum computers can execute it. This abstract idea of splitting is formulated in the Quantum-Classic Split pattern. This thesis examines these hybrid algorithms with a focus on Variational Quantum Algorithms and their split-points. Different implementations of the Variational Quantum Eigensolver and Quantum Approximation Optimization Algorithm are tested with several components. Many classical components are reusable. If the split-points are set well, Classical optimizers, classical pre-processing and classical post-processing can be tested and implemented by classical software engineers. The benefit of splitting algorithms are, that the implementation can be distributed between software engineers and quantum experts and the classical parts can be reusable. The guidelines on how to set split-points are structured similar to patterns as pattern-candidates.

Kurzfassung

Quantencomputer haben das Potenzial bestimmte Probleme schneller oder genauer zu lösen als klassische Computer. Deshalb wurde Quantumcomputing immer populärer. Das sorgte für mehr Forschung in diesem Bereich um den Vorteil, den Quantencomputer bieten, zu nutzen. Heutige Quantencomputer können noch nicht alle Algorithmen mit relevanten Instanzgrößen ausführen. Um heute schon von Quantencomputern zu profitieren, werden Algorithmen in klassische und Quantenteile getrennt, sodass aktuelle Quantencomputer diesen Quantenteil ausführen können. Dieses Vorgehen, Algorithmen aufzuteilen, wurde im Quantum-Classic Split pattern zusammengefasst. In dieser Masterarbeit werden hybride Algorithmen, vor allem Variationelle Quantenalgorithmen und deren Split-points untersucht. Verschiedene Implementierungen des Variationellen Quanten Eigensolvers und des Quantum Approximation Optimization Algorithm wurden mit verschiedenen Komponenten getestet. Viele der klassischen Komponenten sind wiederverwendbar und können von klassischen Softwareentwicklern implementiert und getestet werden. Vorteile, einen Algorithmus aufzuteilen sind, dass die Implementierung zwischen Quantenexperten und Softwareentwicklern aufgeteilt werden kann und, dass die klassischen Teile wiederverwendbar sein können. Die Richtlinien, wie diese Split-points gesetzt werden sollten, wurden ähnlich wie Pattern als Patternkandidaten strukturiert.

Contents

List of Figures

1 Introduction

Quantum computing saw a rise in popularity in recent years due to the potential to solve tasks faster or with higher accuracy than classical computers can [NC11]. The way quantum computers work differs fundamentally from classical computers, as Quantum bits (Qubits) can only be manipulated via unitary operations instead of logical gates [NC11]. These operations are, among others, rotations with possibly irrational angles, so they have limited precision as those angles cannot be expressed with arbitrary precision on a computer, which leads to errors, especially when many gates in sequence are applied. Qubits suffer from decoherence [Pre18], therefore the execution time is further limited.

With all these problems mentioned, many algorithms are not executable for larger problem instances, as they need too many qubits or the quantum circuit is so deep, that the errors get too big. In [Pre18] these issues established the term Noisy Intermediate-Scale Quantum (NISQ) to describe the state of quantum machines at the moment. The main idea is to keep the execution time and the number of required qubits small [SBLW20]. This is achieved by splitting the algorithm into a classical part and a quantum part which only contains parts that are infeasible to run on a classical computer. A whole category of algorithms not only split the execution into classical pre-processing, quantum part and post-processing but into a loop with a classical part and a quantum part [CAB+20]. Because of the alternating behaviour between classical and quantum, this loop is referred to as a hybrid loop. Algorithms containing a hybrid loop with a parametrized quantum circuit are called hybrid or Variational Quantum Algorithms [CAB+20].

These quantum algorithms can be very complex and forces or structures often occur repeatedly. Similar to software engineering, patterns can help developers to create or apply these algorithms [Ley19]. Patterns define the context of a problem, which needs to be solved and it provides an abstract solution for that problem. The Quantum-Classic Split pattern encapsulates the idea, to split a quantum algorithm into classical parts and quantum parts [WBLV21]. The solution of this pattern is very abstract and can be refined [WBLV21]. These refined patterns take the idea and provide a more concrete solution. A prominent refinement is the Variational Quantum Algorithm pattern. The pattern splits the execution to optimize parameters during execution. This pattern also utilizes the state preparation pattern, which can be found in the related patterns. Following the related pattern forms a line to follow which is a pattern language [Ley19]. With a pattern language, developers can keep track in such a highly complex field as quantum computing.

The quantum software lifecycle [WBL+20] consists of phases to follow when new quantum algorithms are developed. The process provides guidance for the development, as it mentions important aspects, specific to quantum computing like the proper hardware selection or error mitigation.

To execute these algorithms, there are already public offerings from IBM [Qis22e], Amazon [Ama22a], Rigetti [Rig22] and Microsoft [Mic22] to name a few. The QPUs are accessible via the cloud, where the user sends the quantum circuit and gets back the measured result. Hybrid

execution runtimes like Qiskit runtime [Qis22k] or Amazon Hybrid Jobs [Ama22c] can be used to execute the whole code in the cloud including both, the classical part and the quantum part. The classical part normally runs on the users' machine. This influences the implementation because the communication in hybrid environments differs from regular communication between a classical computer and the cloud.

This thesis provides an overview of hybrid algorithms and their components. The overall goal is to examine these components and give guidelines on how to use them to split algorithms. As they split the algorithms in a quantum part and a classical part, setting a split-point deliberately could bring benefits in terms of code re-usability and distribution of the implementation task between quantum experts and classical software engineers. Some use-cases are shown, including their split-points and execution results on actual quantum hardware and simulators. Can quantum algorithms be split besides the split between the quantum part and a classical part and how much does one need to know, to use these implementations? The findings are structured and presented in a way that essential ideas are captured. The formulated pattern-candidates are applied to the Quantum Software Lifecycle [WBL+20]. The goal is to verify or extend the phases, such that Variational Quantum Algorithms are specifically addressed.

In Chapter 2 quantum computing basics are provided to understand the algorithms presented in this thesis. Then a short explanation about patterns to structure the search for split-points followed by an overview of today's quantum hardware from IBM and its usage is given. Chapter 3 shows quantum algorithms are split and in particular quantum algorithms which refine the Quantum-Classic Split pattern. In Chapter 4 the covered algorithms are executed in different ways. Then, in Chapter 5 findings are documented and finally applied to the quantum software lifecycle. Chapter 6 shows related work, more precisely papers which work on the same algorithms with various methods and different aspects. In Chapter 7 a summary and a conclusion about the examined split-points and algorithms is given. Then a short outlook and further topics are named.

2 Background

In this thesis, quantum-related topics are examined and therefore this chapter provides the basics of quantum computing such as how a quantum computer or a quantum algorithm works and mathematical basics. This chapter also includes a short introduction to patterns, as algorithms can be described through patterns for a quick overview and better understanding. For visualization, quantum workflows are used. The last section in this chapter will be about the general structure of Variational Quantum Algorithms because they are the primary use-case of the Quantum-Classic Split. The parts about quantum fundamentals are based on [NC11] and [Hom18], but examples and definitions will be marked explicitly.

2.1 Quantum computing background

A classical computer uses bits and can be interpreted as switches which are on or off and in a more mathematical way 0 or 1. The first difference to quantum computers is, that quantum computers work with quantum bits (qubits). In the quantum world, the Dirac notation with brackets like $|\cdot\rangle$ is commonly used to display states. These states are interpreted as vectors where $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. They form a Basis in the complex vector space $\mathbb{C}^2$.

Definition 2.1.1 (Qubit)
A qubit in some arbitrary state $|\psi\rangle$ can be written as a linear combination with the basis $\{|0\rangle, |1\rangle\}$ known as the computational basis

$$|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle \tag{2.1}$$

where α and β are complex numbers and are called amplitudes. With the amplitudes, probabilities can be calculated and need to fulfil the equation $|\alpha|^2 + |\beta|^2 = 1$ [NC11]. A superposition describes the behaviour, that a qubit can be in the state $|0\rangle$ and $|1\rangle$ at the same time [Hom18]. A prominent example is the uniform superposition

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{2.2}$$

This state is written as $|+\rangle$. A way to look at qubits is as vectors on the three-dimensional Bloch sphere.

Definition 2.1.2 (Bloch sphere)
To determine the point which corresponds to a state $|\psi\rangle$ of a single qubit on the Bloch sphere, two angles (θ, ϕ) are needed, where the amplitudes α and β are written as $\alpha = cos(\frac{\theta}{2})$ and $\beta = e^{i\phi} sin(\frac{\theta}{2})$.

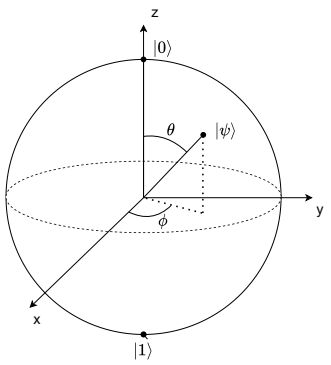


Figure 2.1: An arbitrary state $|\psi\rangle$ represented on the Bloch sphere with the 2 angles (θ, ϕ).

A qubit in state $|\psi\rangle$ can be represented with two angles as shown in Figure 2.1.

2.1.1 Quantum register

As in classical computing which operates on many bits, for quantum computers it is needed to operate on more than one qubit as well. Classical computers use registers which consist of bits, the equivalent in the quantum world is the quantum register. A quantum register consisting of the two qubits $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ and $|\phi\rangle = \beta_0|0\rangle + \beta_1|1\rangle$, following the Dirac notation, can be written as $|\Psi\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle$.

Definition 2.1.3 (Tensor product)
Let V be an n-dimensional vectorspace and W be an m-dimensional vectorspace. For the tensor product, the symbol $\otimes$ is used and the result is an $n \cdot m$ dimensional vectorspace [Hom18].

The tensor product is needed to describe the state of a quantum register with more than one qubit. In the example, the tensor product was applied $|\psi\rangle \otimes |\phi\rangle$ to describe the quantum register. This is often called quantum parallelism, because a classic register holds one value out of 2^n for n bits, while a quantum register can hold all 2^n values at the same time in the form of a superposition.

2.1.2 Operations on Quantumbits

Until now, qubits and their state were described, but to do a calculation it is necessary to manipulate qubits. For the manipulation of qubits, unitary operations are needed, because every possible state needs to be transformed to another possible state.

Definition 2.1.4 (Unitary operation [Hom18; NC11])
A unitary operation U fulfills the property $U^\dagger U = I$, where I is the identity and $U^\dagger = \overline{U^T}$.

The major difference from classical operations is, that unitary operations are reversible. Classical gates are in general not reversible. The result of the AND-gate does not provide a way to reconstruct the input, but a unitary quantum operation can always be reverted. Unitary operations mathematically

describe transformations from one quantum state to another quantum state. The effect of single qubit gates can be represented as 2×2 matrices. Very often used single qubit operators are the Pauli-operators, which are defined as

$$\sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.3}$$

Especially for this thesis, these matrices will appear in almost all examined algorithms. Building upon the Pauli-matrices, rotational gates $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ around the axes on the Bloch sphere for arbitrary angles θ can be built.

Definition 2.1.5 (Decomposition)
Every single qubit unitary operation can be expressed as a sequence of rotations. One example here is called the ZY-decomposition

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta). \tag{2.4}$$

The values of α, β, γ and δ are real-valued [NC11]. With this idea and the extension of multi-qubit gates described in [NC11, pp. 20 sqq.], a finite set of quantum gates is universal, comparable to the universality of the NAND in classical computing. Another important single qubit operator is the Hadamard-operator, which is defined as

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.5}$$

and creates a superposition when applied on a qubit in the computational basis. So applied on a qubit in state $|0\rangle$ it yields

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{2.6}$$

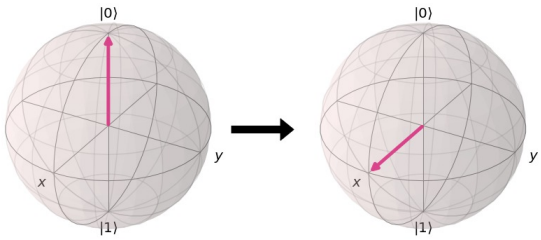


Figure 2.2: Hadamard on the blochsphere. The transition from $|0\rangle$ to $|+\rangle$ using Hadamard.

2.1.3 Quantum gate

In the classical world, logical gates are used to manipulate bits, where one can easily calculate the result for a given input, which is often done with truth tables. The input can be lost, after the application of the classical gate in contrast to unitary quantum gates as they are reversible. In this thesis, the focus lies on quantum gate-based devices. Figure 2.3 shows the mathematically described

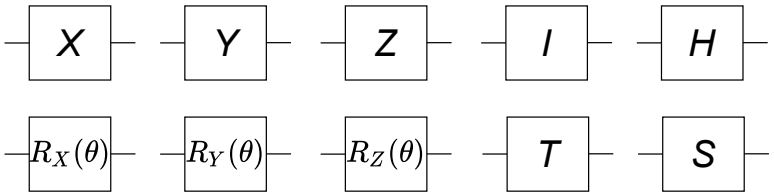


Figure 2.3: Top left to right: σ_x, σ_y, σ_z, σ_0, Hadamard. Bottom left to right: Rotation around the X-axis, Y-axis, Z-axis with the angle θ. The T-gate is a rotation around the Z-axis with $\theta = \pi/4$. The S-gate is a rotation around the Z-axis with $\theta = \pi/2$.

operators as blocks which represent the gates. The quantum gate applies the unitary operation described by the matrix. As every unitary operation can be expressed as a rotation, every quantum device supports a rotation gate that can be sequentially applied to equal the unitary operation. Another often seen quantum-gate is the $U(\theta, \phi, \lambda)$, which is a universal rotation gate. The angles θ, ϕ and λ describe the rotation on the Bloch sphere with a ZY-decomposition [CBSG17].

Another powerful concept of quantum computers is entanglement. Two qubits are entangled when they are not separable into states of single qubits. They are separable if they can be written as a tensor product of states of the single qubits. When qubits are entangled, they can be spatially divided and remain entangled [Hom18]. The primary examples are the Bell states $\{\Phi^+, \Phi^-, \Psi^+, \Psi^-\}$ [Hom18]. The concept here is, that if the first qubit is measured, the superposition collapses and the state of the second qubit is set. The way to produce such an entanglement is done via 2-qubit operations, very often with the CNOT operation. As an example consider the state $\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$ which is separable and can be written as $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$. When CNOT is applied to the register, it is in the state

$$CNOT\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{2.7}$$

which is the Bell-state $|\Phi^+\rangle$.

2.1.4 Quantum circuits

A quantum circuit is composed of quantum gates that are applied in sequence on quantum registers with a measurement at the end. The operations shown in Figure 2.4 were covered in the previous subsection. Each line represents a qubit and every quantum gate is an operation on that qubit. The width (w) of the quantum circuit is the number of qubits which are manipulated during the execution. The depth (d) of the circuit is the number of the longest sequential application of quantum gates on a qubit.

2.1.5 Measurement

The measurement is of fundamental importance for this thesis, as it marks a point where the quantum world and classical world interact. In the process of the measurement, the superposition is destroyed and the state is projected on the basis which was used for the measurement , which is the

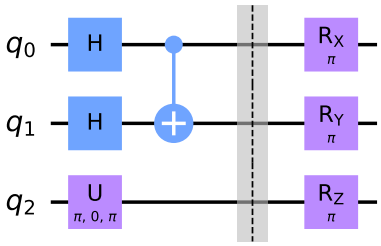


Figure 2.4: A quantum circuit with 3 qubits composed of several single qubit operations and a CNOT gate. The barrier does not influence the quantum circuit.

computational basis $\{|0\rangle, |1\rangle\}$ most of the time. When the state $|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ is measured, the state is with the probability $|\alpha|^2$ in state $|0\rangle$ and with the probability $|\beta|^2$ in state $|1\rangle$ [NC11]. When a quantum register is measured, the probability of the state is given by the amplitudes of the respective basis vectors. Like this, the complete register can be measured. A way to describe the measurement is as a projection with an observable [Hom18].

Definition 2.1.6 (Observable)
An observable O is defined as a Hermitian operator that has a spectral decomposition expressed through

$$O = \sum_{\lambda} \lambda P_{\lambda}. \tag{2.8}$$

The eigenvalues of O are λ and P_λ is a projector on the eigenspace of the respective eigenvalue. This way, the probability to measure the eigenvalue λ when the system is in the state $|\psi\rangle$ is

$$Pr_{measure}(\lambda) = \langle\psi|P_\lambda|\psi\rangle. \tag{2.9}$$

An often seen form of measurement is the projective measurement, where an operator O is defined, which tells the quantum computer how to measure [Hom18] and then interpret the results. Measurement is illustrated in quantum circuits as in Figure 2.5. The double lines where the values are read out and saved are classical connections and registers.

In the context of this thesis, the measurement if often done with Pauli-strings because the Pauli matrices are Hermitian [NC11] and most Variational Quantum Algorithms depend on an efficient measurement. This cannot be achieved trivially. The operators of Pauli-strings can be very large and every summand needs to be measured. To measure in the computational basis, Pauli Z is used. It is sufficient to measure with Pauli Z, if an algorithm is applied, to create and transform the state such that it can be measured [GAD+19].

2.1.6 Quantum Workflows

Before quantum algorithms are examined, here quantum workflows are introduced. Workflows in general are visual instructions with directions like a step-by-step manual. This intuition is one of the benefits because they are easy to read and very clear. Whether the model is a Petri net or a Business

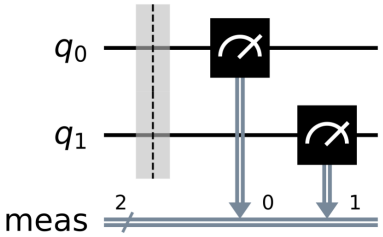


Figure 2.5: Measurement of 2 qubits in the computational basis.

process model (BPM), with very few symbols and conventions a whole workflow can be described in very different areas like traffic or complex economic processes. In this thesis, the focus lies on the Business Process Model and Notation (BPMN2) [BPM22]. The different levels of BPMN2 are:

Level	Name	Description
Level 1	Descriptive Modeling	Very basic description of the process with graphical elements
Level 2	Analytical Modeling	Adds technical details, like data structures but not executable
Level 3	Executeable Modeling	Executable by extending Level 2 Model with executable details

The diagrams in this thesis are drawn following the descriptions of level 1 or level 2. The models used for algorithms are level 1 or level 2, but as in [WBLW20] described, methods for translation into native workflow models are proposed. However, the models in this work are held informal with no guarantee that they are transformable.

Unified Modelling Language

In the Unified Modelling Language (UML) [UML22], different types of diagrams can be divided in 2 groups. The structure diagrams describe components to give an overview of the full application. A class diagram also shows methods, variables and relations between different classes. The second group are behaviour diagrams, and they show the functionality of applications. Therefore, they can be seen as an alternative way to display a workflow. There are a couple of different diagrams such as activity diagrams, communication diagrams, sequence diagrams and use-case diagrams, to name a few. In this thesis, activity diagrams fulfil the purpose to illustrate simple workflows without many technical details. The semantics is mostly expressed from the labels of the different components. Whenever only high-level concepts need to be known to understand the behaviour, activity diagrams are used. The symbols are simple shapes. Ellipses are actions, where a verb shows the action done in the respective step. A diamond for decisions in the sense of if-else statements. Arrows indicating the direction of the flow and black bars for concurrent executions. Black circles serve as the start and end-point.

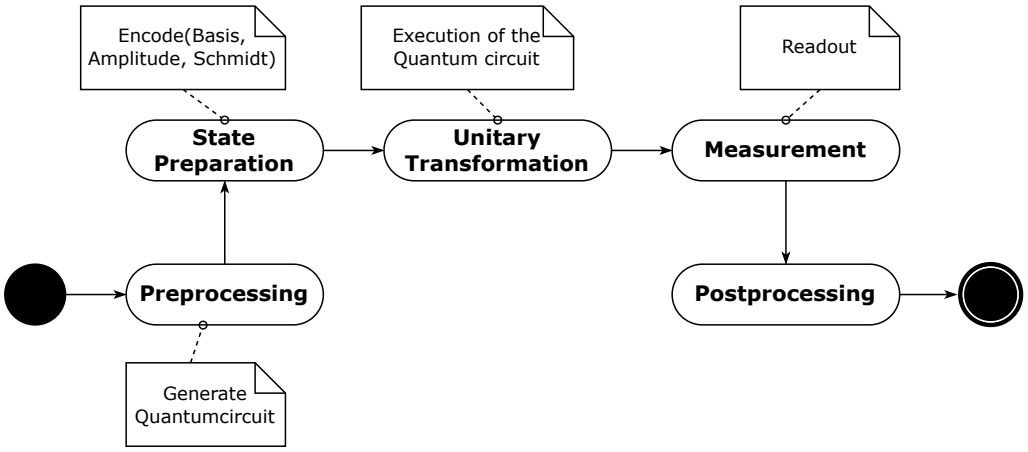


Figure 2.6: The workflow of a quantum algorithm, adapted from [LB20]

2.1.7 Quantum Algorithm

By intuition, quantum algorithms run on a quantum computer, but that is not the whole truth. Most quantum algorithms contain a classical part and a quantum part. The quantum part is a quantum circuit which runs on the quantum computer and the classical part can be pre-processing or post-processing. Pre-processing can be data preparation to prepare the state of the qubits in the quantum computer. When measurement in a quantum circuit is done, the result is written to a classical register, where it can be computed further on a classical device or plotted as a histogram of expectation values. The concept is visualized in Figure 2.6. The visualization is done with UML-activity symbols, introduced in 2.1.6.

As shown, a quantum algorithm is hybrid because they consist of a classical and a quantum part. A whole class of algorithms which should not be confused with quantum algorithms described in this section are hybrid quantum algorithms. They are further discussed in Section 2.3 because their structure is a bit more complex.

2.1.8 Noisy Intermediate-Scale Quantum

Quantum computers today are called Noisy Intermediate-Scale Quantum (NISQ) because the execution of quantum circuits is error-prone which limits the possibilities for a quantum advantage when solving real problem instances [Pre18]. To be more precise, qubits suffer from decoherence. This contains the relaxation of a qubit, which means the transition from the state of a qubit to an orthogonal state of the qubit. The other effect is dephasing, which is a small random change of the phase of a qubit. Another source for errors is low gate-fidelity. An example of a source of low fidelity are irrational angles when applying rotational gates which results in rotations with small errors when translating them into rational inputs on the computer. With the representation of the Bloch sphere with rotations describing the state of a qubit in mind, all these errors must not be ignored, because the accuracy of the angles is limited. Qubits can interact with the environment and can therefore be unintendedly transformed. In [SBLW20] metrics are presented to decide whether a quantum circuit is executable with acceptable error rates or not. The criteria with depth (d) is

defined as sequential executed quantum gates and width (w) as number of qubits. A rule of thumb is given [SBLW20] as

$$wd \ll \frac{1}{\epsilon}, \tag{2.10}$$

where ϵ is the effective error rate. This needs to be fulfilled, such that the execution of an algorithm is feasible.

2.2 Patterns

The first appearance of a pattern language is in the book "A Pattern Language: Towns, Buildings, Construction" by Christopher Alexander [AIS77]. Patterns are structured documents with a description and solution for a problem. The structure depends on the domain but should be consistent if possible. A pattern has a "name" and a small "icon" which can be self-explanatory when common symbols are used. That provides a way to recognize and remember a pattern better than with only a title. Directly with the icon comes the driving question of the pattern for a quick overview to decide whether a pattern is useful for a certain problem or not. The "context" combines several aspects. It describes a situation and problems which was the inspiration for the pattern. The "solution" describes an abstract solution for the problem. It does not specifically define technical details and does not show concrete instances where the pattern is applied. In the"known uses" part, references to the application of the pattern are shown. A user could use these references as examples of how a solution looks like and possibly take certain parts when it fits the concrete situation. The "Related pattern" section references patterns which are more abstract or more concrete, but also patterns which are part of the context or the solution. Often related patterns are not explicitly written but are mentioned in the other sections of the pattern description.

Initially, patterns were not built in the context of software development, but as software got more complex, patterns provide a good way to structure ideas and keep track of tested and proven solutions. Thus, patterns are prominent in software development nowadays, here as overview [1] for example in cloud computing. In [FLR+14], the patterns compress fundamental and important information for building cloud computing service.

When many patterns are related, they can form a directed graph, where one can jump from pattern to pattern to solve problems raised in the process. These patterns referring to each other are the graphical equivalent to a pattern language introduced by Christopher Alexander. Like a geographical atlas, every language represents a map and the connections between the maps are the connections between patterns from different domains. An example in [LB21] combines the microservice and the cloud computing domain to get the combined pattern language for the development of microservices in the cloud. Similar to other domains, patterns were found for the quantum computing domain. Patterns related to quantum computing are proposed in [Ley19; WBLS21; WBLV21].

[1] https://www.cloudcomputingpatterns.org

2.2.1 Pattern identification, authoring and application process

The process of finding patterns is mostly done by pattern experts. When experts, possibly from different domains work together, standardized steps to follow will help to coordinate the team. Here the phases and steps proposed in [FBBL14] are shortly explained.

In the Pattern identification phase, in the *Domain Definition* step, the domain needs to be identified and described, so that everyone working on the patterns has the necessary knowledge. In the *Coverage Consideration* step, the information from the first step is structured to filter the relevant information from a possibly large domain. When there are findings, in the *Information Format Design* step, the data should be presented similarly. In the IT context, UML diagrams can be used. Pattern primitives can be established which are domain-specific elements. The *Information Collection* step is to condense the information into a defined document format. The ideas for patterns can be identified by experts, but can also come in the form of existing examples. From the example, the relevant information is used for the pattern. In the *Information Review* step which is the last step of the phase, the coverage of the domain is reviewed. This results in a refined structure that can show similarities between solutions because, in a smaller domain, there are fewer comparisons needed. The drawback could be, that a smaller domain excludes important findings or solution and therefore needs to be done with care. The phase can be re-iterated.

In the Pattern Authoring phase, the information from the last phase is used to design the patterns. First of all, in the *Pattern Language Design* step, a convention for the style of the patterns needs to be defined. There can be differences between the domains, but best-practise pattern formats can serve as a starting point where adjustments can be made. Integral is the problem statement when to use the pattern and the abstract solution. After the overall pattern format is defined, the pattern primitives are defined in the *Primitive Definition* step. These primitives can be the same as in the last phase, but can also be altered. The *Composition Language Definition* step is to specify how the primitives are used to create graphical representation. The graphical elements could be formalized for clearer usage. Then follows the *Pattern Writing* step where all the preparation is used to write the pattern as a structured document. The abstraction level needs to be set detailed enough, so that users can apply the pattern but not with too much detail, because then the concrete problem instance might differ too much such that the pattern cannot be applied. To address this problem, the pattern is discussed by authors and users to improve the level of abstraction. After the patterns are written, in the *Pattern Language Revision* step, the relations between the patterns are explored and then added to the patterns to form the language or directed graph like in the pattern atlas.

The Pattern Application phase can be seen as a stand-alone phase because it describes the application of a pattern language. The *Pattern Search and Recommendation* step for a user is to search for suited patterns, which is not trivial because there could be many patterns. With the related patterns, the user can search for the right context and collect suitable patterns before reading the detailed descriptions and the solution. When the patterns are chosen, the user can get an idea from the solution the pattern was abstracted from in the *Pattern-based Solution Design* step. Next is the *Refinement of the Solution Design* step, where the pattern is applied and refined to fit the specific case. The problem is, that the technology which can be used is limited in companies. A solution could be to use freely usable resources or from other companies, to give the user more freedom for the concrete pattern application.

2.2.2 Quantum-Classic Split Pattern

Patterns can also be applied in quantum computing. Especially in this new domain, abstract proven solutions for reoccurring problems can assist research or application through a reoccurring problem. An overview of quantum computing patterns can be found in [Pat22].

This section is about the pattern defined in [WBLV21]. The Quantum-Classic Split pattern is on top of further refined patterns and the focus of this thesis. As the name suggests, a quantum algorithm following the idea of the Quantum-Classic Split distributes the execution between a classical computer and the quantum computer. As mentioned above, a classical part within the workflow of a quantum algorithm is not unusual and in itself does not constitute an application of the pattern. The split between the classical and the quantum part cannot be set arbitrarily, because the quantum part contains superpositions and entanglements, which cannot be split. The reason is, that if the execution is split, the state needs to be measured to be saved for further execution, but doing so destroys the prepared state and therefore the measured state cannot be used equivalently. Hence, the abstract solution to split is not trivially applicable. A better idea of how to split can be found in the refinements of the Quantum-Classic Split pattern. One is the quantum kernel estimator where the quantum part helps to avoid calculating a high dimensional scalar product after using the kernel trick to separate data points. The kernel function equals the scalar product and is calculated with the help of the quantum computer. A whole category of quantum algorithms is described in the next section.

2.3 Variational Quantum Algorithms

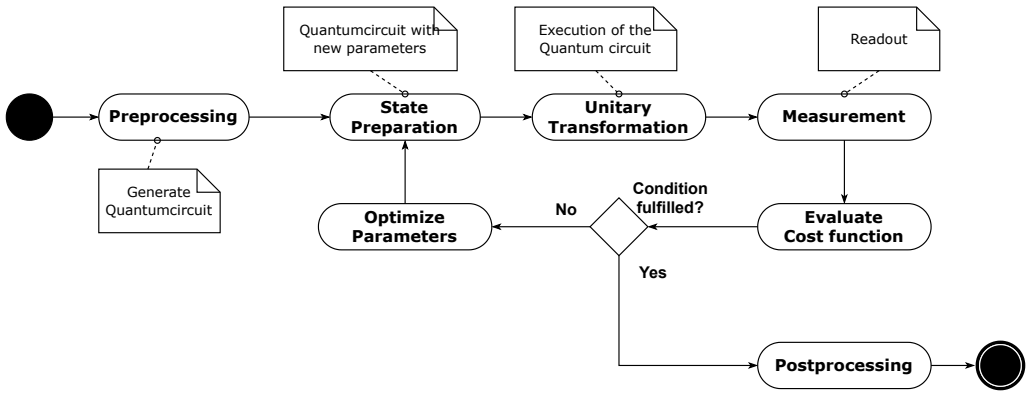


Figure 2.7: The workflow of a Variational Quantum Algorithm as UML Activity diagram, adapted from [CAB+20].

Variational Quantum Algorithms (VQA) are quantum algorithms which are hybrid but not in the same way other quantum algorithms are. VQAs are split but contain several classical parts [CAB+20].

The general structure is shown in Figure 2.7. The input is a cost function and a quantum circuit which is called ansatz. Optionally a set of training states can be used to help the optimization of the parameters. The focus lies on the box, where the quantum part and classical part are executed in a loop, alternating between classical and quantum parts.

2.3.1 Cost function

The cost function C encodes a problem with respect to parameters θ. A general definition [CAB+20] is given in the form of

$$C(\theta) = f(\{\rho_k\}, \{O_k\}, U(\vec{\theta})) \tag{2.11}$$

Here, ρ_k are test states which can be used to help in the optimization step, but they are optional. $\{O_k\}$ are a set of observables, which define how the quantum circuit will be measured. The cost function is also dependent on the parametrized unitary operators $U(\vec{\theta})$.

Cost functions which can be efficiently computed by a classical computer do not need to be calculated with a quantum computer. There must be improvements in runtime or accuracy possible compared to classical approaches. These benefits are often referred to as quantum advantage. If the cost function is not efficiently computable, the advantages of using a quantum computer can diminish or completely vanish. That includes the measurement which is not trivial because most quantum computers measure in the computational basis (Z basis) and the quantum circuit to transform in order to measure in the basis contribute to the depth of the quantum circuit. The same criteria apply to the parameters of the cost function which also need to be optimized efficiently, otherwise, the benefits can be lost in the classical optimization step. A cost function must be faithful [CAB+20]. That means, the optimal solution of the cost function, whether it is a minimum or a maximum, needs to approximate the solution of the encoded problem. When all the requirements are met, the ansatz and the optimizer must be chosen carefully as these components contribute heavily to the solution quality.

2.3.2 Ansatz

The quantum circuit used for a Variational Quantum Algorithm is called ansatz [CAB+20]. The difference to the quantum circuits examined in subsection 2.1.4 is, that the ansatz depends on the parameters $\vec{\theta}$. The unitary operations are therefore expressed as [CAB+20]

$$U(\vec{\theta}) = U_L(\theta_L) \cdots U_2(\theta_2) U_1(\theta_1), \tag{2.12}$$

where every unitary operation is [CAB+20]

$$U_k(\theta_k) = \prod_m e^{-i\theta_m H_m} W_m. \tag{2.13}$$

W_m is a unitary operator without a parameter. In this thesis, the focus lies on ansätze, that are described with H_m in the exponent which is a hermitian operator which is often built with Pauli-operators. Although the general structure of all Variational Quantum Algorithms is similar, the different ansätze utilize various strategies to keep the quantum circuit small. Due to their complexity in this thesis ansätze are shown in detail which are used in the examined algorithms, other ansätze are only mentioned. Many ansatz categories are discussed in more detail in [CAB+20].

2.3.3 Optimizer

In this section, optimizers are introduced and prominent examples are presented. A cost function depends on an array of Parameters $\vec{\theta}$. In order to minimize or maximize this function, optimal values for these parameters need to be found. The new parameters are found through the equation [CAB+20]

$$\vec{\theta}^* = arg \min_{\vec{\theta}} C(\vec{\theta}). \tag{2.14}$$

The cost function landscape can have many local minima. However, for an optimal solution, the optimizer navigates through the cost function landscape to find a global minimum [CAB+20]. Many local minima may exist where the gradient will stop as it found a minimum. To further optimize many iterations are necessary to have a chance for the global minimum. An optimizer is an essential component of machine learning algorithms and with the similarity also for Variational Quantum Algorithms. Many ideas for optimizers are taken from the classical world and applied to the measured expectation values of the quantum computer. Although they are useable in this context, experiments have shown huge differences in the quality of the result with various optimizers [SIKC19].

The stochastic gradient descend is a commonly used method for optimization [SIKC19]. The gradient of the cost function is calculated and with it, the optimizers steps in the direction of the steepest descend. However, there is a stochastic behaviour and therefore the steps may be too big to find a minimum or so small, that it the runtime is increased. Furthermore, it is not always feasible or possible to calculate the full gradient, as there are many parameters resulting in a high dimensional gradient. The step size of the optimizer is regulated by the learning rate α [KACC19].

A further developed variant of the gradient descend is ADAM which is also well-known from machine learning. ADAM uses 2 different momentums applied to every component of the gradient. This way, the step size is adjusted during the execution. With this strategy, ADAM avoids jumping over minima by lowering the step-size with respect to the momentums [KACC19].

Gradient-free optimizers are considered slower in most cases, but the simultaneous perturbation stochastic approximation (SPSA), presented in [Spa01], is very present also in the context of quantum computing despite not making use of the full gradient in the first place. It is related to gradient-based methods because it optimizes in a random direction and estimates a derivative in this direction [CAB+20].With α as learning rate and c_t for the step-size in the derivative, SPSA can be fine-tuned with hyperparameters. Both parameters get smaller during execution to avoid jumping over minima [KACC19].

The optimizers covered were not built with the idea of usage with quantum computers. Quantum aware optimizers [ACSC20; KACC19; SIKC19] specifically address quantum-related properties like quantum gates, the measurement or parts of the quantum circuits. The individual Coupled Number of Shots (iCANS) optimizers introduced in [KACC19] are based on CABS known in the context of machine learning optimizers. The approach is to adjust the number of shots for every partial derivative. The idea is, that the closer to the optimal value, the more shots are needed. This idea is refined with static and dynamic learning rates and then compared to state-of-the-art optimizers like Adam and SPSA. It is remarked in this paper, that the chosen hyperparameters influence the performance [KACC19].

Other quantum-aware optimizers, proposed in [OGB19] are Rotoselect and Rotosolve. Especially the Rotosolve optimizes the rotation angles and with it, the structure of the ansatz itself. These optimizers utilize the fact, that the rotations can be expressed as sinusoidal curves with the Pauli-operators. For a rotation, the Pauli-operators for the angle are calculated and the minimum of those in $[-\pi, \pi]$ is taken as the operator with the respective angle. This process is done for each rotation, while the others are kept static. A problem is, that there are 4 Pauli-operators resulting in a 4^n complexity. In [OGB19] it was pointed out, that only the native gate set of the quantum computer is used, which often does not consist of all Pauli-operators or their rotation respective.

Parameter-shift rule

When the gradient is not available, the parameter shift rule can be used to create one with not too much effort. This is useful for VQA, which often rely on gradients, to optimize the parameters. The partial derivative can be obtained by adding a parametrized single qubit quantum gate [Cro19]. The parameter of this gate is shifted in one direction, then the quantum circuit is executed a second time, with the parameter shifted in the other direction. This method is a simple and computational cheap alternative to the finite-differences method [SBG+18].

2.3.4 Barren Plateaus

The phenomenon of barren plateaus (BP) was observed in quantum computing. When a BP exists the magnitude of the derivations are vanishing, such that the landscape of the cost function is very flat. Then optimizers may need many optimization steps to reach a minimum. The steps need to be small because the slope is small and it could step over the minimum if the step size is too big. This effect can diminish if not completely destroy the anticipated quantum advantage. The problem scales with the number of qubits but also with the depth of the quantum circuit. The occurrence correlates with the depth of the quantum circuit and how many qubits a cost function manipulates. If a cost function manipulated all n qubits when the system size is n, it is called global. Otherwise, the cost function manipulated k out of the n qubits, then it is called k-local.

The following table provides an overview, when BP can occur with respect to the circuit depth and the property of the cost function in terms of qubits [MBS+18].

Global $C(\theta)$	Circuit depth	Barren plateau
yes	$O(log(n))$	BP possible
yes	$O(poly(n))$	BP possible
no	$O(log(n))$	No BP

BP can also occur, when the ansatz is initialized randomly chosen parameters. There are strategies to mitigate BP. The idea to use gradient-free optimizers is not a solution as they also suffer from BP. A strategy to mitigate proposed in [CAB+20] is to choose initial parameters not randomly.

2.4 Technology

Quantum computing was a mathematical concept, where algorithms like Shor's algorithm [Sho95] were introduced long before quantum computers were built. In this section a short introduction to already available systems to run quantum algorithms on real quantum hardware is given. These systems are NISQ-Machines, which were introduced in subsection 2.1.8. The services used are public offerings from IBM [Qis22e]. Other providers are Rigetti [Rig22] and Amazon with Braket [Ama22a]. In the IBM Quantum Lab, one can use Jupyter notebook [Jup22] to execute code with Python and the required libraries which also can be installed with pip. IBM offers various systems listed in [Qis22b], where the *Systems* tab shows actual quantum computers and some of their properties. The Simulator tab shows the available simulators in the cloud. In Qiskit [Qis22e] there are implementations, which can be used to run simulations locally.

2.4.1 Quantum computers

Here, a short explanation for quantum computers is given, because it is important to know how to interpret the different values one can see, when looking at the properties in the *Systems* tab in [Qis22b]. The importance for the user in order to use quantum computers lies in quantum-specific challenges like the awareness of the errors or the number of qubits required for the uploaded quantum circuit. The number of qubits must be taken into account every time a quantum circuit is sent to the quantum device. A quantum circuit can only be processed if the device offers enough qubits. Other factors which do not prevent the execution are important nonetheless. The CNOT Error and the Readout Error, as well as the decoherence time (T1) and the dephasing time (T2) are displayed when clicking on a system in [Qis22b]. These values can change over time, as qubits can interact with the environment and therefore need to be calibrated from time to time. The entry with total pending jobs in the top shows the number of jobs which are queued on the device. If no time slot is booked beforehand, a long job queue can result in long waiting times. An important part is the basic gates set, supported by the quantum device. The supported gates can be used "out of the box". When gates are used, which are not natively supported, they must be expressed as a sequence of basic gates. An example can be seen in Figure Figure 2.8.

To illustrate the importance of this, two different transpiled versions are done with the basic gates of "ibmq_Lima" and in comparison with "ibmq_qasm_simulator". Both transpilations are equivalent in terms of functionality but different in the gates used and in circuit depth. An important note is, that the circuits will be different with different error-rates of the qubits or CNOT-gate.

(a) A quantum circuit transpiled for real quantum hardware. In this case for the "ibmq_Lima".

(b) The same quantum circuit as in (a) transpiled for the "ibmq_qasm_simulator".

Figure 2.8

To apply 2-qubit gates, the qubit-map of the QPU must be taken into account. The lines between the qubits are connections as shown in Figure 2.9. The colouring indicates the error of a qubit or a connection between them. The CNOT-gate is used to entangle qubits, but only qubits with such a connection can be entangled. Whether a CNOT is applicable can be seen in the calibration data map and in this example, the CNOT cannot entangle qubit 0 and qubit 2 without swapping the content with an intermediate qubit.

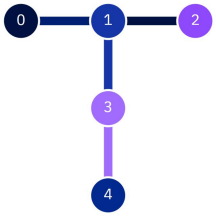


Figure 2.9: Qubit map of the “ibmq_lima”

If there is entanglement in a circuit, the transpiler needs to map the qubits in a way to make entanglement possible. This is achieved by swapping qubits to a position where a connection to another qubit can be established. The problem is, that swapping is expensive because it increases the depth of a quantum circuit as it requires 3 CNOT gates to achieve a swap shown in Figure 2.10.

(a) A swap gate **(b)** A swap gate decomposed

Figure 2.10: The swap gate has the depth = 3, when it is not in the gate set of the quantum computer

For teaching, learning or proof of concepts, simulators are a convenient way to execute quantum circuits without a real device with a limited number of qubits or long waiting times due to highly occupied job queues. An overview can be found in the simulators tab in [Qis22e]. Most simulators support more native quantum gates than real quantum computers, which can influence the depth of the quantum circuit. An example comparison of transpiled quantum circuits between a simulator and an actual quantum computer can be seen in Figure 2.8.

2.4.2 Hybrid Execution Environment

Hybrid execution environments are provided by IBM with the Qiskit Runtime [Qis22k] and by Amazon with Amazon Braket Hybrid Jobs [Ama22c]. In contrast to the regular approach where only the quantum circuit is sent to the cloud service, a runtime program contains both the classical and

the quantum part. The communication between the classical computer and the quantum computer during the execution is completely done in the cloud. Only intermediate results or the final result will be sent to the user.

For quantum algorithms, which only need a single execution of a quantum circuit, the regular approach is reasonable. If a quantum algorithm has a hybrid loop, every quantum circuit sent to the quantum computer is queued in the job queue. If the quantum device is occupied and without priority, even very small circuits, like the instances in this thesis can have hours of waiting time. Therefore, a regular execution is dependent on the occupation of the quantum devices. When using the hybrid environment, not only a quantum circuit is sent to the quantum device, but a package with classical and quantum code. This upload also is handled via job queue, but the whole package with all iterations is executed. Not only the quantum code but also the classical code can have long execution times. To regulate this, different user groups have different execution time limits. The open group, where access is free, has a stricter time limit than premium users.

Usage

A list of publicly available programs in the Qiskit Runtime can be found at [Qis22b] in the program tab. A provider has to be loaded to check which privileges the user has. The main group is free of charge and can be loaded after creating an IBM account. A list of usable programs can be printed, where a program can then be invoked with its ID. This includes own private programs, but the user needs to be aware of other time limitations of private programs. To invoke a runtime program, the program ID and a backend are necessary. More inputs are specified by the runtime program. In the context of Variational Quantum Algorithms, often the operator, the number of shots and an initial point need to be set. For a description in more detail, one can check [Qis22l], where a short introduction is given. A runtime program consists of a python file with the combined classical and quantum code and metadata as a JSON file. The metadata necessary can be created with a simple python script. When additional parameters are used, another tag with *spec* and the additional keys in the metadata needs to be specified. With the key “required”, data within the specified fields must be provided to invoke the program. A detailed description of different keys is very important to assist the usage of public programs.

3 Algorithm selection

In this chapter, we examine how the Quantum-Classic Split pattern is applied. To illustrate the split-points, some algorithms are covered in detail. For a consistent form in this chapter, all algorithms are presented as pseudo algorithms.

Standard literature was consulted to gather suitable algorithms for this research. Suitable algorithms are ones which show the quantum concepts and split-points can be identified. A popular first quantum circuit presented is a random number generator, because it introduces the capabilities of quantum computers. The simple circuit contains the initialization of the qubit to the $|0\rangle$ state, then a Hadamard gate is applied and finally measured. This algorithm tackles the problem of classical computers, which offer only pseudo-random number generators [Hom18]. The split-point between the quantum code and the classical code is right after the measurement where the result is a classical bit. Here the hybrid nature of quantum algorithms can be observed because the result after the measurement is saved and displayed by a classical computer, which is referred to as post-processing.

The quantum algorithms introduced in standard literature, have a quantum part and a classical part. However, the focus often lies on the quantum circuit, such that the classical part which is necessary for real quantum computing is neglected. Examples of quantum algorithms often mentioned in literature are the Deutsch-Jozsa algorithm, the Grover-Search algorithm [Hom18], Quantum-Fourier transformation [Hom18], Quantum Phase estimation [NC11], the HHL algorithm [HHL08] and Shor's algorithm [Sho95]. All these algorithms share, that they execute the part on the quantum computer, which has a high complexity in the classical world. These quantum algorithms are often based on the mathematical basics of quantum computing and make use of the quantum effects like entanglement and superposition. Quantum algorithms which implement mathematical concepts have the goal to improve the algorithmic properties or constraints of classical computers.

The Deutsch-Jozsa algorithm, for example, is used to determine whether a function is constant or balanced with only one execution of the unitary which serves as an oracle [Hom18]. A classical computer needs at least 2 function evaluations, therefore this quantum algorithm provides a runtime improvement compared with the classical version. A split-point is located at the measurement after the quantum circuit is executed. The function need to be encoded into the quantum circuit. The translation from a function to the respective quantum circuit requires quantum knowledge. It is a split-point in terms of quantum knowledge.

The HHL-algorithm is aimed to speed up solving linear equation systems. The runtime of the best-known classical algorithm for a sparse $N \times N$ Matrix can be improved by the HHL algorithm [HHL08]. The split-point after the execution of the quantum circuit is also the measurement. In this algorithm, one needs to know how to encode the input data. The encoding step is the pre-processing before the quantum circuit is executed. The split-point is located at the encoding step.

Shor's algorithm splits the part on the quantum computer, where the period of a function needs to be found to achieve the factorization of a given input. This also utilizes the Quantum-Fourier transform, which itself provides a runtime improvement over the classical Fourier-Transform [Hom18]. This algorithm utilizes computation in the pre-processing and the post-processing that means, there are classical parts before and after the execution of the quantum circuit. The split-points are set, such that the period finding can be made computationally efficient. All other parts are left on the classical computer. The finding here is, that algorithms not only encode problem instances on the quantum computer, but also compute intermediate results on classical computers.

The Quantum-Classic Split pattern describes that split-points are problem and implementation-dependent [Ley19]. For further investigation, differentiation is needed dependent on what the goal of the application of the Quantum-Classic Split pattern is. Wherever a calculation cannot be efficiently done on a classical computer, it is a potential target to set the split-point.

In the search on how split-points can be set, the paper [WBLV21] is used to find more algorithms which make use of the Quantum-Classic Split. A more concrete pattern which derives Quantum-Classic Split pattern is the Variational Quantum Algorithms pattern [WBLV21]. These algorithms are designed with not only the regular pre-processing and post-processing split-points. Within the hybrid loop, which is a loop alternating between the quantum part and classical part, there are multiple split-points. To get a more detailed view of the split-points of VQAs, concrete VQAs like the Variational Quantum Eigensolver (VQE) and the Quantum Approximation Optimization Algorithm (QAOA) which are refined patterns in [WBLV21] are examined. These quantum algorithms are taken as representatives, as they have a wide range of use-cases. With more domains and applications, VQAs are important for quantum computing especially in the near term, but they do not lose value, when quantum computers are noise resistant and have more qubits [CAB+20]. This potential provides a reason, to examine this category of algorithms specifically, to help the development and understanding of these algorithms.

3.1 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) can be an alternative to the Quantum Phase Estimation (QPE), if only an eigenvalue is searched. The QPE is used by many algorithms to find eigenvalues, but it is not suited for NISQ-machines because it needs many gates for practical usage. The VQE addresses this problem with the variational approach.

A problem can often be formulated in a way, that the eigenvalues of a matrix can be a solution itself or correspond to a solution. The basic version of the Variational Quantum Eigensolver seeks the minimum eigenvalue of a hermitian matrix. Such matrices are often used in quantum mechanics. Quantum systems evolve with Hamiltonians. In the scope of quantum computation, the Hamiltonian is used for the calculation which is a Hermitian matrix [NC11].

Following the general structure of a Variational Quantum Algorithm [CAB+20], the VQE consists of a cost function encoding the problem, an ansatz which is the quantum circuit for the QPU and an optimizer to calculate new parameters for the quantum circuit. The general structure of a Variational Quantum Algorithm presented in Section 2.3 is slightly adjusted in Figure 3.1. The state preparation is done with a suitable ansatz for the specific problem. After the measurement, the measured expectation values are calculated and then evaluated.

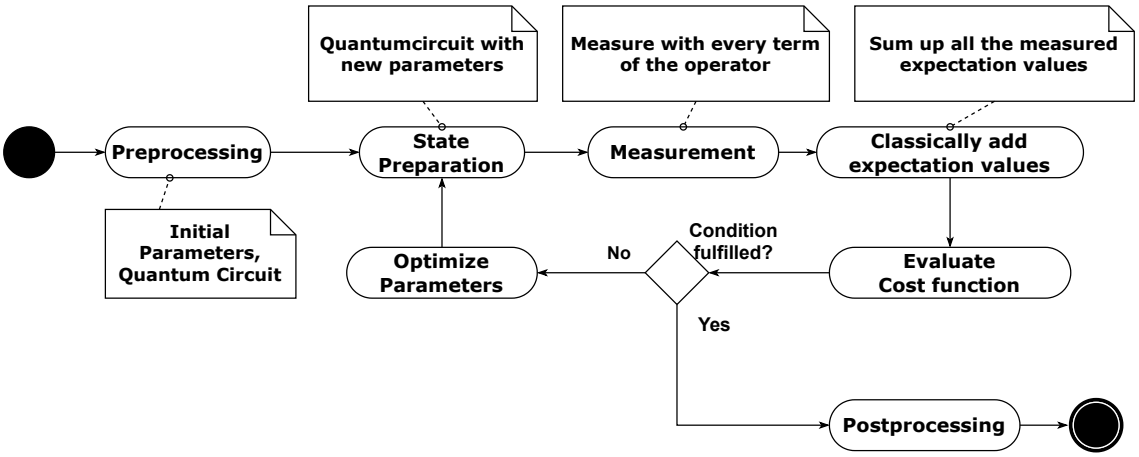


Figure 3.1: The workflow of the Variational Quantum Eigensolver as UML Activity diagram, adapted from [PMS+13]

The cost function can be different depending on the specific instance. Here a generic definition from [CAB+20] is used with

$$C(\vec{\theta}) = \langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle. \tag{3.1}$$

$\vec{\theta}$ are parameters which are used to prepare the state $|\psi(\vec{\theta})\rangle$ before measuring. The right-hand side of the cost function written in Braket notation is the expectation value of all the measurements of the state $|\psi(\vec{\theta})\rangle$. The Hamiltonian is often represented as linear combination of Pauli-operators σ with $c_i \in \mathbb{R}$ as

$$H = \sum_i c_i \sigma_i. \tag{3.2}$$

Every term of the sum yields an expectation value which will be added together to get the expectation value of the cost function. The algorithm without problem-specific post-processing is shown in Algorithm 3.1. The problem definition is encoded in the operator. The targeted quantum backend limits how the operator can be built. A prominent limitation is the number of qubits, where the terms in the summands of the operator must not exceed the available number of qubits. The parameter list $\vec{\theta}$ is initialized with random values. If better start parameters are known through testing or estimation from quantum experts, they can be used . Then the algorithm enters the hybrid loop. The preparation and the measurement are parts on the quantum computer, the summation and the optimization are classical parts.

To prepare the state, an ansatz is used. The VQE is presented with different ansätze that are suitable for different problems. The idea of the *Unitary Coupled Clustered* (UCC) ansatz [Qis22g] is based on quantum chemistry and therefore considered a problem-inspired ansatz. Molecules are described with the fermionic Hamiltonian with their excitations. These excitations are mapped to the qubits which are coupled and entangled by 2-qubit operators. The depth of this ansatz can be optimized with different variants. Available versions are the original *Unitary Coupled Clustered* (UCC), *Unitary Vibrational Coupled Clustered* (UVCC), *Unitary Coupled Clustered Single Double* (UCCSD) to

Algorithm 3.1 VQE algorithm

Input: Problem definition

1: Define operator $O = \sum_i c_i \sigma_i$ from the problem definition
2: Initialize $\vec{\theta} \leftarrow random(dim(O))$
3: **while** Not converged **do**
4: Prepare $|\psi\rangle$ with $\vec{\theta}$ by executing quantum circuit (Ansatz) // Split-point from classical to quantum
5: Measure with P_k to get $\langle O \rangle_k$ // Split-point from quantum to classical
6: Sum all expectation values: $\langle O \rangle = \sum_k c_k \langle O \rangle_k$
7: Evaluate C($\vec{\theta}$)
8: $\vec{\theta} \leftarrow$ Optimize($\vec{\theta}$)
9: **end while**
10: **return** $\langle O \rangle$ // Criteria met and $\lambda_{min} = \langle O \rangle$

name a few implemented ones. A comprehensive list of already implemented UCC-ansätze can be found in [Qis22j]. An ansatz where the idea of keeping the circuit small in depth is the Hardware efficient ansatz [CAB+20]. By using only natively supported quantum gates, the depth of the circuit can be controlled during the implementation. The connections between the qubits to entangle can also be taken into account to minimize the number of swap gates.

The first split-point is the entrance into the hybrid loop, which is done in *line 4* from the classical computer to the quantum computer. The description of a VQA contains a state preparation and an ansatz [WBLV21], which is combined in the description of the VQE and formulated as state preparation. The other split-point is inside the hybrid loop after measuring with the defined operator in *line 5* from the quantum computer to the classical computer. The sum for the expectation value is done classically. The general description of a VQA sets the split-point after the measurement and then evaluate and optimize the parameters with the cost function. In the VQE description, there is the classical adder specifically formulated in between. It is an example, that a VQA can described with more classical components inside the hybrid loop, as the measurement in most descriptions include the calculation of the expectation value. The VQE follows the structure of a VQA closely, with similar split-points.

3.2 Quantum Approximation Optimization Algorithm

The quantum approximation optimization algorithm (QAOA) refines the Quantum-Classic Split pattern and implements the quantum alternating operator ansatz pattern [WBLV21]. The patterns used are state preparation, quantum alternating operator ansatz and measurement. Prominent applications of QAOA are optimization problems like Max-Cut and 3-SAT which are NP-hard and can be approximated with QAOA [FGG14].

Following the structure of a VQA, the QAOA consists of a cost function, an ansatz and the classical optimizer. With the parameter vectors $\vec{\theta} = (\vec{\gamma}, \vec{\beta})$ a cost function can be written as [CAB+20]

$$C(\vec{\gamma}, \vec{\beta}) = \langle \psi_p(\vec{\gamma}, \vec{\beta}) | H_{Prob} | \psi_p(\vec{\gamma}, \vec{\beta}) \rangle. \tag{3.3}$$

H_{Prob} is the Hamiltonian operator encoding the problem definition. Together with the state $|\psi_p(\vec{\gamma},\vec{\beta})\rangle$, the expectation value can be calculated which approximates the solution. The p in the index is the number of repetitions of the ansatz. The first observation is, that QAOA requires more parameters than the single parameter vector $\vec{\theta}$ used for the VQE. In this case, the quantum circuit is built with a list of 2 parameter vectors $(\vec{\gamma},\vec{\beta})$, as it utilizes an ansatz with 2 different Hamiltonians. Another observation is, that p influences the depth of the quantum circuit with greater p resulting in deeper quantum circuits.

Algorithm 3.2 QAOA algorithm

Input: Cost function C encoded in H_{Prob}, Hyperparameter p
1: $(\vec{\gamma},\vec{\beta}) \leftarrow random(dim(H_{Prob}\cdot p), dim(H_M \cdot p)), \gamma \in [0,2\pi], \beta \in [0,\pi]$
2: **while** Not converged **do**
3: $|\psi\rangle \leftarrow |0\rangle^{\otimes n}$ // Split-point from classical to quantum
4: $|\psi\rangle \leftarrow Hadamard(|\psi\rangle)$
5: **for all** p **do**
6: $|\psi\rangle \leftarrow U(C,\gamma_i)(|\psi\rangle)$ // Unitary Operator
7: $|\psi\rangle \leftarrow U(B,\beta_i)(|\psi\rangle)$ // Unitary Mixer Operator
8: **end for**
9: Measure $|\vec{\gamma},\vec{\beta}\rangle$ in the computational basis // Split-point from quantum to classical
10: Calculate $\langle C\rangle_{|\vec{\gamma},\vec{\beta}\rangle}$
11: $(\vec{\gamma},\vec{\beta}) \leftarrow$ Optimize$(\vec{\gamma},\vec{\beta})$
12: **end while**
13: **return** $\langle C\rangle_{|\vec{\gamma},\vec{\beta}\rangle}$

The Quantum Alternating Operator ansatz must not be confused with the quantum approximate optimization algorithm (QAOA), although both share the same acronym. The Quantum Alternating Operator ansatz can be understood as a generalization of the QAOA [WBLV21], because the domain of possible solutions is not constrained. The ansatz of QAOA contains a state preparation, a phase separation operator that encodes the problem Hamiltonian and a mixer operator. This algorithm specifically is formulated as a pattern in [WBLV21]. The quantum register is initialized with a uniform superposition. The qubits are in state $|0\rangle$ initially and then Hadamard is applied on all qubits to create the state

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |q_k\rangle. \tag{3.4}$$

The unitary operator is described as [CAB+20]

$$U(\vec{\gamma},\vec{\beta}) = \prod_{l=1}^{p} e^{-i\beta_l H_M} e^{-i\gamma_l H_{Prob}}. \tag{3.5}$$

The parameters are described as $(\vec{\gamma},\vec{\beta})$. H_P is the Hamiltonian operator, which encodes the cost function. H_M is the mixer Hamiltonian. The unitary with γ rotates the qubits to change the phase. The more, the term contributes to the cost function, the higher the change in phase. But only the phase is changed with this operator, not the amplitude. In order to increase the amplitudes, a mixer operator is applied. The mixer operator $U(B,\beta)$ in the context of QAOA is realized with $R_X(2\beta)$

on every qubit individually [WBLV21]. These rotations around the X-axis are applied after each phase separation. The for loop with the hyperparameter p, is written as a pre-build quantum circuit with both unitary operators repeated p-times in succession. After the measurement, the calculation of the cost function with the measured state expectation value of $|\vec{\gamma}, \vec{\beta}\rangle$ and the optimization are classical post-processings. In Algorithm 3.2 the execution is modelled with the operator repetition as a loop inside the while loop. The split-point here is very similar to the VQE and can be found in the standard structure of a VQA with more than only one operator.

3.3 Warm-Start

In this thesis, the variant of warm-starting with classical preparation is covered. Other forms of warm-starting are not part of this thesis. Warm-start is defined as a refinement of the Quantum-Classic Split pattern in [WBLV21] and is introduced in [EMW20].

3.3.1 The general idea of warm-starting

Problems that are NP-hard are generally not believed to be efficiently computable on classical machines. For these problems, the exact result is not always needed and approximations can be good enough. These approximations are bounded by an approximation factor which means, how close can an approximated result of an algorithm get to the real solution. Under the assumption, that the unique games conjecture (UGC) holds many approximation factors can be estimated and no algorithm can get closer to the solution than the estimated interval. The UGC for quantum computers does not hold if entanglement is used within the computation [KRT07]. The conclusion is, that the boundaries for the classical world are not the boundaries in the quantum world. A quantum algorithm may get in the approximation interval and therefore closer to the optimal solution. To summarize, a classical pre-processing is performed to approximate a solution. This solution is computed efficiently, but the goal is to get a more precise solution. The classical solution serves as a starting point for the quantum algorithm.

3.3.2 Preparation of the Warm-Start Solution

Warm-starting is a general concept, but for practical purposes, problems are considered that can be approximated efficiently on classical computers. One class which fulfils that condition are Quadratic Unconstrained Binary Optimization problems (QUBO). A definition of a QUBO is given by the following equation [EMW20]

$$\min_{\vec{x} \in \{0,1\}^n} \vec{x}^T A \vec{x}, \tag{3.6}$$

with A as a $n \times n$ symmetric matrix. It is possible to add a linear term $\mu^T \vec{x}$, where it can simplify the system of equations [EMW20]. Warm-starting is not constrained to the formulation as minimization problems as in this section and can also be formulated as maximization problems. The classical way to approach such problems, as they are NP-hard is to relax them to

$$\min_{\vec{x}\in[0,1]^n} \vec{x}^T A\vec{x}, \tag{3.7}$$

where for x, all values between 0 and 1 are possible. This problem is also NP-hard but can be approximated to a certain extent. To achieve a binary solution a rounding scheme has to be used. But this solution is often not precise enough, as the approximation factor is a hard bound for the solution quality. Instead of applying a rounding scheme, the approximated solution can be used as a starting point. This approach is further described in the next subsection. Other variants of the warm-start QAOA using rounded results are called WS-RQAOA. In [EMW20] it was shown, that these variants can help the execution on quantum computers with a limited number of qubits and the authors also give a guideline on when and how to use relaxation. Other formulations are also possible, as long as the output is $c^* \in [0, 1]$ to directly encode the warm-started solution into the quantum circuit.

3.3.3 Code modfications

The first modification of the standard QAOA is at the state preparation step. Instead of the uniform superposition with Hadamard, the classical solution is encoded in parameterized rotations around the Y-axis [EMW20]. The approximated classical result $\vec{c^*}$ is encoded into initial parameters $\vec{\theta}$ in the following way [EMW20]

$$\theta_i = 2 \cdot arcsin(\sqrt{c_i^*}) \qquad \text{if } c_i^* \in [\epsilon, 1-\epsilon] \tag{3.8}$$

$$\theta_i = 2 \cdot arcsin(\sqrt{\epsilon}) \qquad \text{if } c_i^* \leq \epsilon \tag{3.9}$$

$$\theta_i = 2 \cdot arcsin(\sqrt{1-\epsilon}) \qquad \text{if } c_i^* \geq 1-\epsilon \tag{3.10}$$

In some online resources [Qis22i], only the first equation is used, omitting the hyperparameter ϵ. The original paper [EMW20] introduced the hyperparameter ϵ as transition to the cold started QAOA with Hadamard as state preparation and to address the case, when the classical result is 0 or 1. In this case, the qubits would be initialized with $|0\rangle$ and $|1\rangle$. When only ZZ and I gates are used, they remain in the state [EMW20]. This is unlikely in the context of the relaxed results from QUBOs, but common when the cost function is built in the context of the Max-Cut problem. The modified state preparation is defined as [EMW20]

$$|\phi\rangle = \bigotimes_{i=0}^{n-1} R_Y(\theta_i)|0\rangle_n. \tag{3.11}$$

The Hamiltonian operator is the same as in standard QAOA. The mixer Hamiltonian is modified, to also rotate with the angles from the pre-processing. To describe the mixer, the R_Z and R_Y gates as presented in Chapter 2 are used. With $\vec{\beta}$ defined as in the standard QAOA, the new mixer is given as

Algorithm 3.3 WS-QAOA algorithm

Input: Cost function C encoded in H_{Prob}, Hyperparameter p, ϵ, classical result $c^* \in \{0, 1\}$

for all c^* **do**
  if $c^* \leq \epsilon$ **then**
    $\theta_i \leftarrow 2arcsin\sqrt{\epsilon}$
  else if $c^* \geq 1 - \epsilon$ **then**
    $\theta_i \leftarrow 2arcsin\sqrt{1-\epsilon}$
  else
    $\theta_i \leftarrow 2arcsin\sqrt{c_i^*}$
  end if
end for
$(\vec{\gamma}, \vec{\beta}) \leftarrow random(dim(H_{Prob} \cdot p), dim(H_M \cdot p)), \gamma \in [0, 2\pi], \beta \in [0, \pi]$
while Not converged **do**
  $|\psi\rangle \leftarrow R_Y(\vec{\theta})(|\psi\rangle)$ // Split-point from classical to quantum
  for all p **do**
    $|\psi\rangle \leftarrow U(C, \gamma_i)(|\psi\rangle)$ // Unitary Operator
    $|\psi\rangle \leftarrow U(B, \beta_i)(|\psi\rangle)$ modified with θ_i // Unitary Mixer Operator
  end for
  Measure $|\vec{\gamma}, \vec{\beta}\rangle$ in the computational basis // Split-point from quantum to classical
  Calculate $\langle C \rangle_{|\vec{\gamma}, \vec{\beta}\rangle}$
  $(\vec{\gamma}, \vec{\beta}) \leftarrow$ Optimize$(\vec{\gamma}, \vec{\beta})$
end while
return $\langle C \rangle_{|\vec{\gamma}, \vec{\beta}\rangle}$

$$U(\theta_i, \beta_i) = R_Y(\theta_i) R_Z(2\beta_i) R_Y(-\theta_i). \tag{3.12}$$

The modified algorithm is shown in Algorithm 3.3. The role of the hyperparameter p stays the same as in the standard QAOA. The other hyperparameter is ϵ which controls how much influence the warm-started solution has on the execution on the quantum computer. The split-point has not changed in terms of the quantum circuit, but it should not be forgotten to count the preparation of the classical approximation into the running time of the algorithm. That means an approximated solution should be efficiently computable otherwise the advantage of using the quantum computer can diminish because of the classical pre-processing. It was pointed out in [BP21] that WS-QAOA was the only algorithm which found the solution, but the processing time of the WS-QAOA is longer than the VQE or QAOA due to the classical pre-processing.

3.4 Applications in other domains

The covered algorithms are very prominent in quantum computing as they can be applied to many problems, but the applications of VQA are much broader. In mathematics, the Harrow-Hassidim-Lloyd (HHL) algorithm is a quantum algorithm to solve systems of linear equations [HHL08]. The quantum circuit needed for the execution is very deep because it uses the quantum phase estimation. NISQ-machines do not provide qubits which are stable enough for a reasonable execution. This was

the motivation for the Variational Quantum Linear Solver (VQLS) [Qis22h]. VQAs are flexible and research is ongoing in many other domains. A prominent application is in the Machine Learning domain, as they share a similar structure with parametrized circuits and optimizers. Quantum Error Correction is another area where VQA algorithms could be of help. The resulting demand of qubits and the depth are too big, but also here, the Variational Quantum Error Corrector (QVECTOR) could potentially help even on NISQ machines [CAB+20].

4 Algorithms, usage with different components

Until now the Quantum-Classic Split pattern was applied in form of an abstract description with refinements for actual execution. In this chapter, case-studies are presented where different components are switched, while the other parts of the algorithm are kept constant. As modularized code is standard in classical software engineering, it is tested how quantum algorithms can be modularized, because these modules could be reusable. The split-points are identified with actual implementations of the Variational Quantum Algorithms identified in Chapter 3.

The intermediate questions for this chapter are the following.

1. Where are the split-points in concrete implementations of the examined algorithms?

2. How much quantum-related knowledge does a user need when using these implementations?

The split-point in the code is set, as soon as quantum related knowledge is needed. That is the case, when interacting with quantum circuits or providing information to build them. The knowledge needed for usage is tested with default settings. Every additional quantum-related input that is needed to use the algorithm is counted as quantum-related knowledge. The results of the executions are examined to decide whether the usage was successful. With these questions, it is investigated, how developers split the algorithms and make them usable.

4.1 Algorithms without hybrid loop

A short implementation for the random number generator explained in Chapter 3 and the resulting quantum circuit is shown. The code shows the implementation of the quantum circuit. It contains the coded logic, but not the exact quantum circuit executed on the quantum device. The classical pre-processing is the part where the quantum circuit is complied and sent to the cloud.

```
1 quantumcircuit = QuantumCircuit(1)
2 quantumcircuit.h(0)
3 quantumcircuit.measure_all()
```

Listing 4.1: Random number generator which acts like a coin flip. Programmed in Python with Qiskit.

It is a simple representation of a quantum algorithm, where the classical post-processing part is the interpretation of the measurement. One needs to be able to understand quantum gates to understand the functionality of the quantum circuit. However, a user of this algorithm does not need to know the internal behaviour or how quantum computers work. It could be seen as a black box acting as a real random number generator inside. Such a black box could be used for algorithms, which do not require inputs or only classical input.

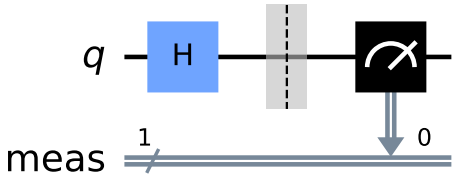


Figure 4.1: Random Generator circuit

4.2 Algorithms with hybrid loop

The execution of a Variational Quantum Algorithm differs from the execution of a single quantum circuit, as there may be multiple quantum circuit jobs and classical processing in between necessary. In Figure 4.2, the workflow of a general VQA is shown. One can see, that there are parts on the user side and the execution of the quantum circuit with the measurement in the cloud. The send and receive events however are still done on classical computers. The two pools do not indicate the split-point of the algorithm, only a split-point of the execution environments.

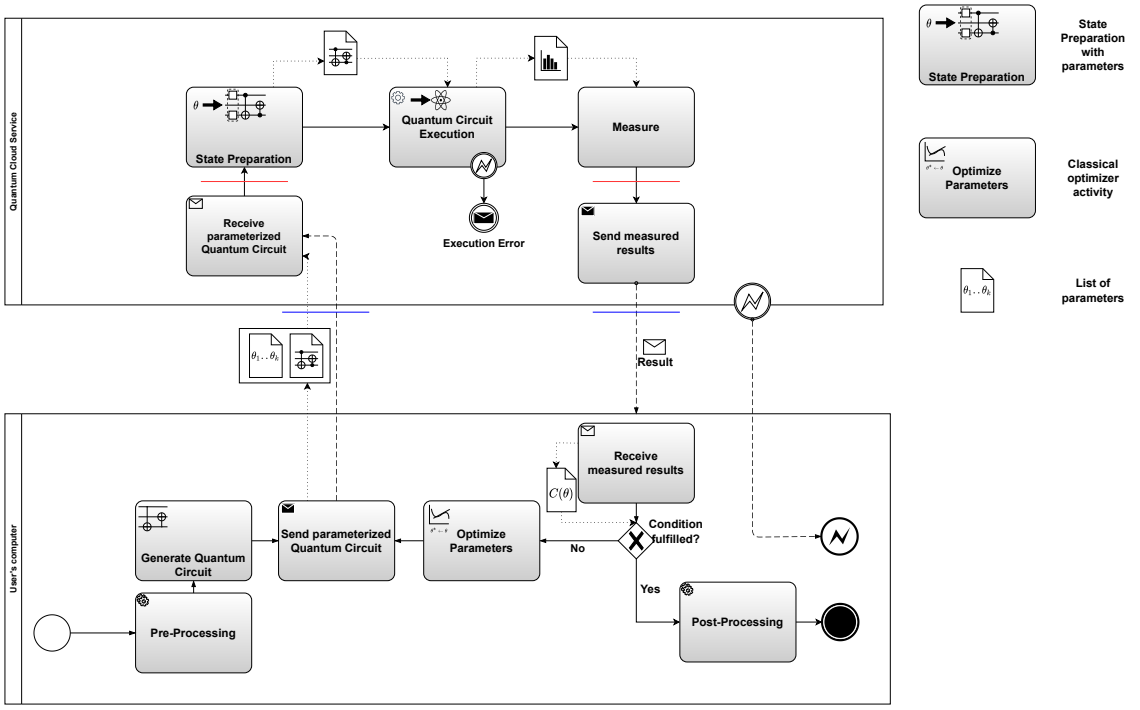


Figure 4.2: Workflow diagram for the execution of a VQA, adapted from [CAB+20] and illustrated with the QuantME [WBLW20] notation. Red lines indicate a split-points between classical and quantum. Blue lines indicate split-points between execution environments. The classical part in the cloud is necessary for the classical communication between the user and the cloud.

4.2.1 Variational Quantum Eigensolver

This section is an analysis of several ways to use the VQE with multiple implementations from [Kim20; Qis22f; Ros21]. Many of these guides are outdated and need to be adjusted to recent library changes. The biggest change is the integration of *qiskit_aqua* into *qiskit_nature*. Here, a use-case with a problem instance from the chemistry domain was picked, to test the VQE with different parts. With the concrete implementation acquired mostly from tutorials, split-points were identified and described. In this use-case, the *GroundStateEigensolver* [Qis22a] of Qiskit is used for computation. It calculates not only the minimum eigenvalue but the ground state energy with other values from the chemistry domain.

As problem instance, the H_2-Molecule is the simplest molecule, but after the pre-processing there are no more than 2 qubits involved. Therefore, to utilize the quantum computing resources with more than 2 qubits, LiH is chosen as problem instance, because it occupies 4 qubits and is also a well-established example in several guides. In this case, the VQE is used to determine the ground state of the LiH molecule.

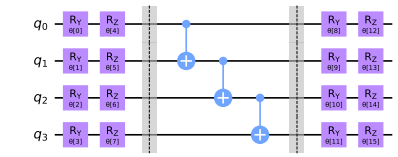
Specific transformations are needed to get the problem instance for the quantum computer. These steps are all classical steps, as they do not involve any quantum-related actions and are left out here for a better overview. The first split-point is, where the operator, a weighted Pauli-string, must

be built. To translate the instance into an operator, mappers are used. There are several mappers, e.g. the *JordanWignerMapper* and the *ParityMapper* are prominent ones. The *ParityMapper* was used for further classical optimization in the pre-processing. After the mapping on qubits with the mapper, an operator is obtained.

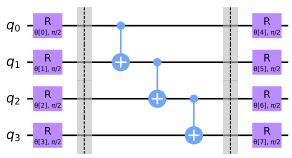
Here with the workflow of the VQE in mind, different ansätze are tested. Ansätze are often problem inspired but potentially usable in other contexts. For the VQE, different ansätze are possible. The basic VQE-module requires an ansatz which needs to be defined, selected or given by the user. Qiskit contains many ansätze that can be used if the required inputs are given [Qis22j].

The variants as depicted in Figure 4.4 are 2 paths when using the *GroundStateEigensolver*. One with the *VQEUCCFactory* and the other path with manual ansatz definition. The other path requires choosing and constructing an ansatz to build the *VQE*. In this test, the *EfficientSU2* ansatz is used, which requires more inputs from the user than the *VQEUCCFactory*. To build the ansatz, the user needs to know the number of qubits needed, how often the ansatz should repeat and which entanglement strategy should be used. The *RealAmplitudes* ansatz is also tested, as it has similarities with the *EfficientSU2* and should serve for another comparison. The differences of the used ansätze are depicted in Figure 4.3, wherein the UCC Ansatz only a part is shown, as it is too large to draw here completely. The circuit depth of the *RealAmplitudes* and *EfficientSU2* is very similar, with the UCC having the greatest depth by far.

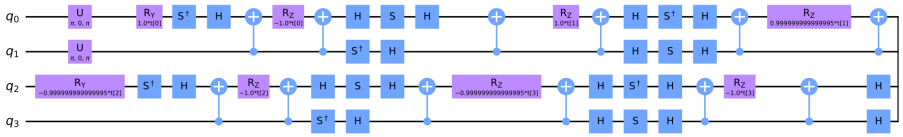
With these preparations, the *GroundstateEigensolver* [Qis22a] can be used. The whole hybrid loop is then executed by the *GroundstateEigensolver*. It sends the quantum circuit to the specified backend and also receives the results for further computation.



(a) EfficientSU2 Ansatz.



(b) RealAmplitudes Ansatz.



(c) UCC Ansatz, cropped for a better overview

Figure 4.3: The different ansätze used for the VQE.

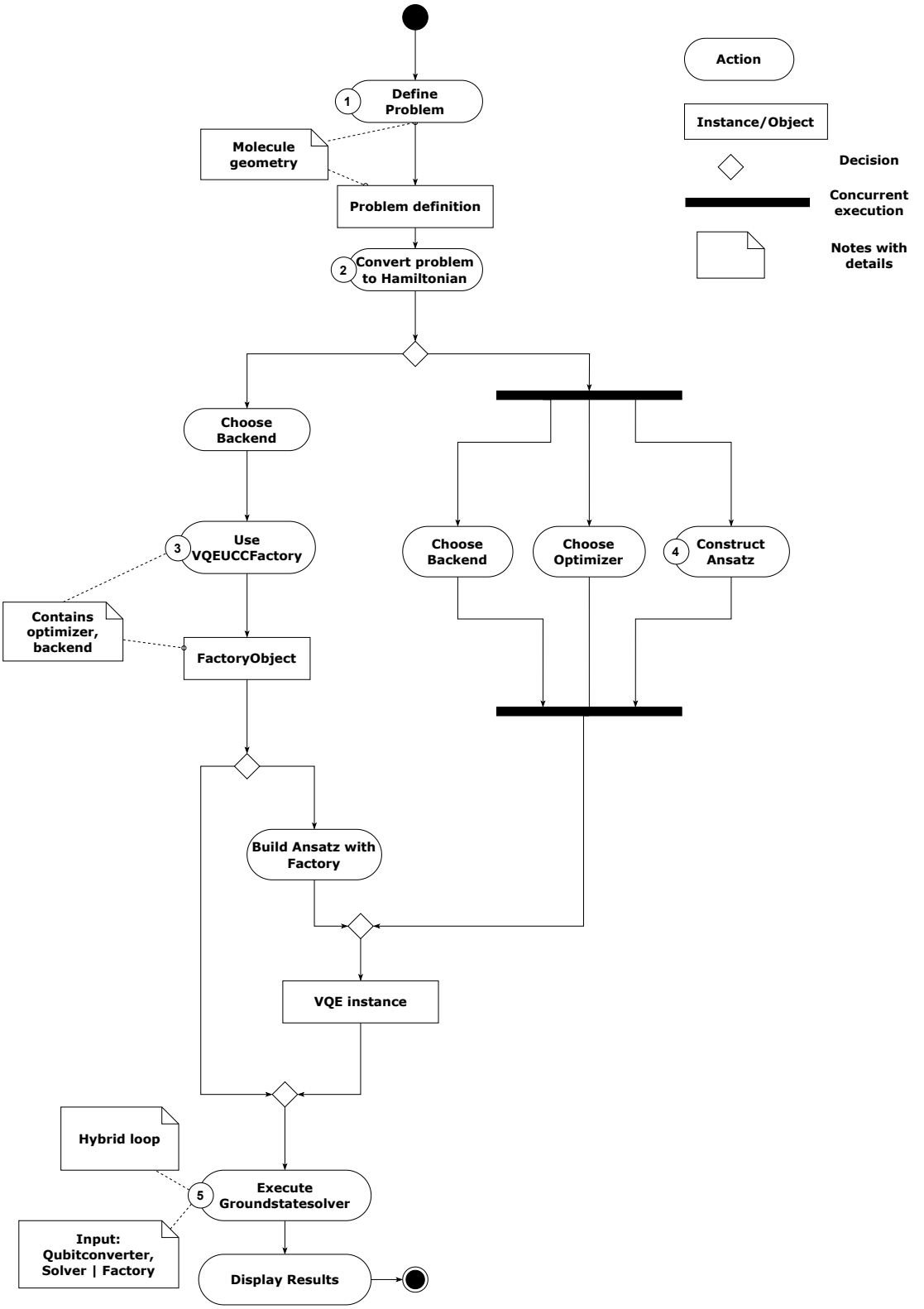


Figure 4.4: Activity diagram for the implementation decisions of the VQE when using the *GroudnStateEigensolver*. A split-point is located in step 2 from the classical problem definition to the quantum representation. The hybrid loop lies completely in step 5, with all the split-points between classical parts and quantum parts.

When following the steps in Figure 4.4, step (1) is a pure domain-specific part, because it defines the problem statement in the chemistry domain. The first examined split-point is in step (2), because the conversion to a Hamiltonian already requires expert knowledge from the quantum domain. In the tested case with the conversion from the chemistry problem statement to an Hamiltonian, the operator is mapped with the help of a mapper module without further inputs. The implementation split-point is set deliberately by Qiskit developers, so that the user can use the modules as black boxes. The path with step (3) automates the construction of the ansatz, while step (4) requires the construction to be done by the user. There are several ansätze which can be used for the VQE and this problem instance and some of them are tested in this case-study. The ansatz can be constructed manually or with the Qiskit cicruit library [Qis22j]. Step (5) contains the execution of the hybrid loop when a quantum instance such as the VQE or the Factory is provided as input. The *GroundStateEigensolver* is a module which is not purely implemented for use with a quantum computer and can be considered as a black box. It also could be used as a classical module with the *NumpyMinimumEigenSolver*. It does the required post-processing in the context of the chemistry domain with the results from the VQE which is used as a subroutine. The VQE only calculates the minimum eigenvalue but the result of the algorithm also includes problem specific values like the Hartree-Fock Energy. The split-points of the VQE examined in Chapter 3 is completely hidden behind the function call.

In Figure 4.5 results were calculated with the *QasmSimulator* and the backend noise-model from a real quantum device. The settings used in the test specifically kept on default to get an idea of which results to expect when using quantum algorithms as a simple function call to simulate the experience of a novice user without in depth quantum computing knowledge following tutorials. One finding is, that the SLSQP optimizer has the most inconsistent results. The guidelines in [Qis22m] recommend the SPSA optimizer for real quantum hardware or the *QasmSimulator*. The results of this test support the recommendation from the guidelines. Every point in Figure 4.5 corresponds to one execution of the algorithm. With the UCC-ansatz, results are consistent with all used optimizers, the average result quality is not ideal. The other 2 ansätze are similar to each other in structure and also in terms of the result quality. The SLSQP is used in the context as seen in guides, but it is not recommended for noisy environments, which could cause inconsistent results with the other ansätze. The COBYLA optimizer performed similarly to SPSA. The combination of the *RealAmplitudes* with SPSA yields the best results. The classical parts and the quantum parts need to work together, in order to get good results. When a quantum expert implements an ansatz which is a quantum circuit, the classical parts need to fit and be suited for the noisy environment. In this use-case, the optimizers can be pre-selected by a quantum expert and given as a hint for the user or another developer in the documentation.

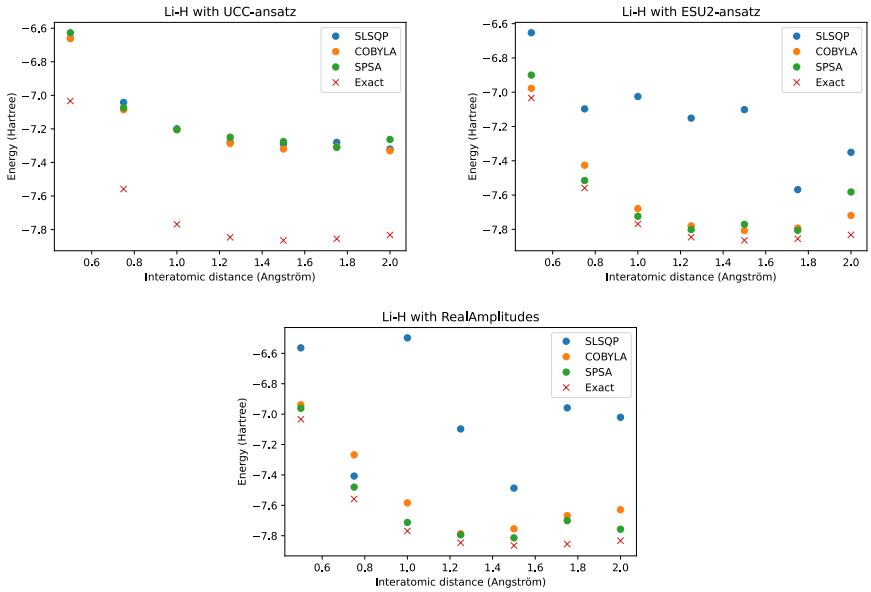


Figure 4.5: Results from the *GroundStateEigensolver* with different ansätze and optimizer. Shots=4000, noise-model and settings from ibm_nairobi (09/06/2022).

4.2.2 Quantum Approxiamtion Optimization Algorithm

The tested implantations of QAOA and WS-QAOA are based on [Qis22c; Qis22d; Qis22n]. To test the QAOA, a problem instance needs to be chosen. One of the most prominent problems in the context of QAOA is the Max-Cut problem. The input is a graph containing nodes and edges. The goal of the Max-Cut problem is to divide the nodes into two subsets which maximizes the number of edges between those subsets. The graph used for the tests is displayed in Figure 4.6. This problem instance has multiple equivalent solutions that means Max-Cut values with different chosen subsets. The subset assignment can be represented by a bit-string. One example of such a representation is 0100000, which means, that *Node* 1 is in one subset, while all other nodes are in the other subset. The counted number of edges between these subsets yields the maximum cut value 5. The inverted bit-string 1011111 also has the maximum cut value 5 as every inversion of a given bit-string.

If Variational Quantum Algorithms are executed on real quantum hardware or a simulator in the cloud, there are two possibilities for execution. The standard execution is shown in Figure 4.2, where the quantum circuit is sent to the quantum computer individually. Every quantum circuit is a circuit job and needs to be queued before it is executed. The queue-time can be mitigated, if time slots are booked. However, network overhead cannot be mitigated with timeslots. The other way to execute the algorithm is to use a hybrid execution environment. The main difference from the regular execution can be seen in Figure 4.7. The part on the users' computer is reduced to a single pre-processing and post-processing task. The loop between the quantum computer and the classical computer lies completely in the cloud.

With runtime programs, the developer can deliberately set the required parameters for running the program. For a quantum expert it is not only possible to achieve a split between the classical part and quantum part, but also to decide how much classical code runs on the users' side. In this

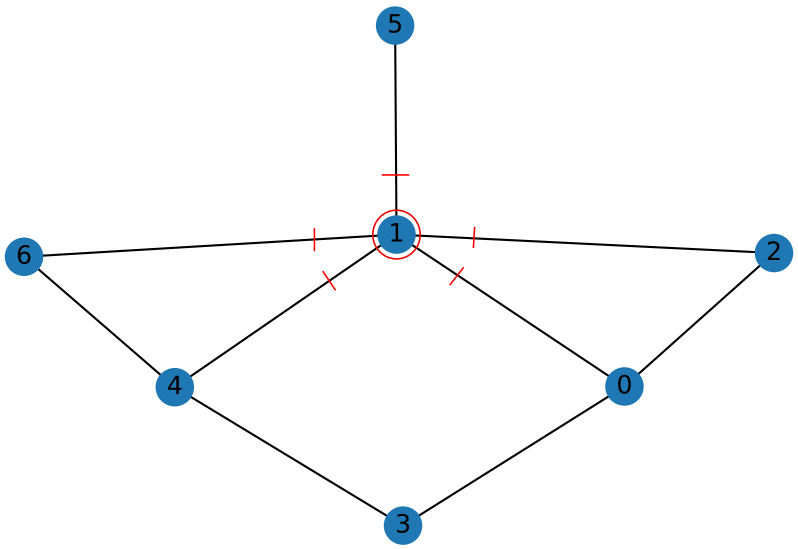


Figure 4.6: The problem instance used for QAOA and WS-QAOA with the example cut 0100000.

use-case, no manually built ansatz is required, as the public QAOA-runtime module implements the Quantum Approximate Optimization Algorithm (QAOA) automatically with some restrictions. The operator must only contain Pauli-I and Pauli-Z which limits the usage, but can also be an optimization for specific problem categories.

To get an operator, a Max-Cut problem instance needs to be translated to the respective operator. The problem is given as a graph with a set of nodes and edges which is shown in Figure 4.6. The operator is constructed such that every node is mapped to a qubit. The edges are implemented in the quantum circuit with entanglements done with RZZ-gates. The rotation angles for these RZZ-gates are the parameters in the cost Hamiltonian. The resulting quantum circuit is shown in Figure 4.8. The parts in the ansatz can be identified following the description of QAOA. First, the state-preparation is done with Hadamard-gates, decomposed to the U_3-gate, which creates the superposition via rotations as explained in Chapter 2. Next is the block with the Hamiltonian, which was built with the operator. The last black box is the mixer operator, realized with R_X-gates. With the hyperparameter p, these 2 black boxes can be repeated increasing the depth of the quantum circuit.

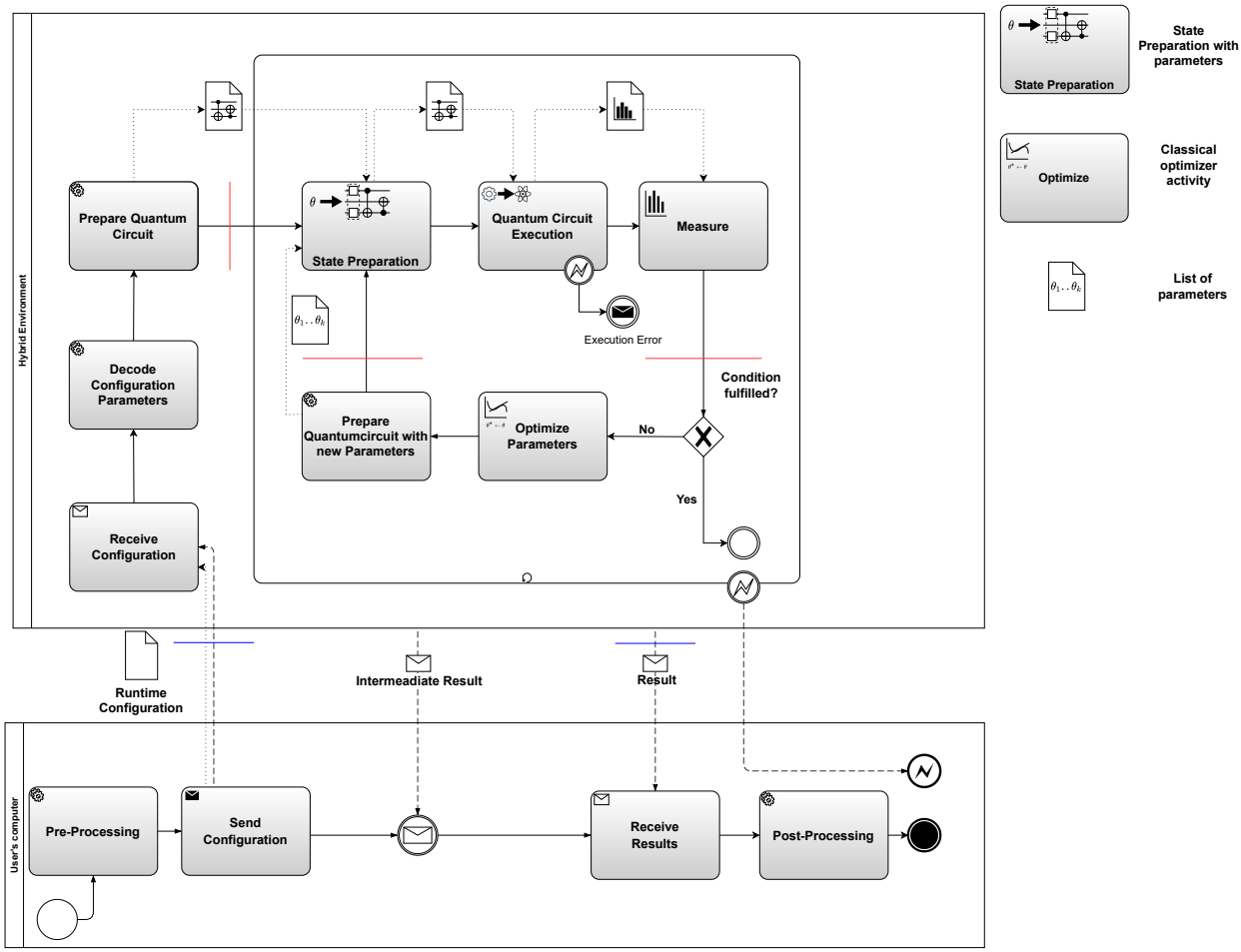


Figure 4.7: Workflow of a VQA in a hybrid environment, symbols are adapted and extended from [WBLW20]. Red lines indicate a split-points between classical and quantum. Blue lines indicate split-points between execution environments.

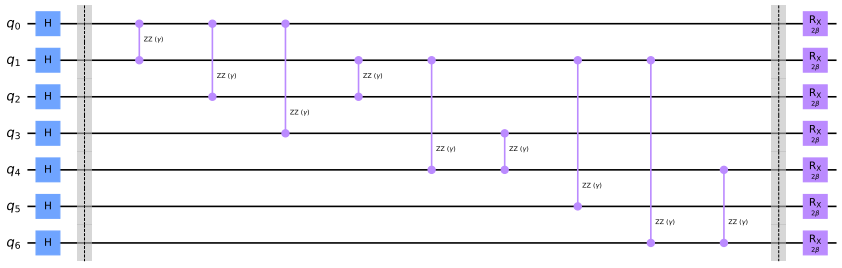


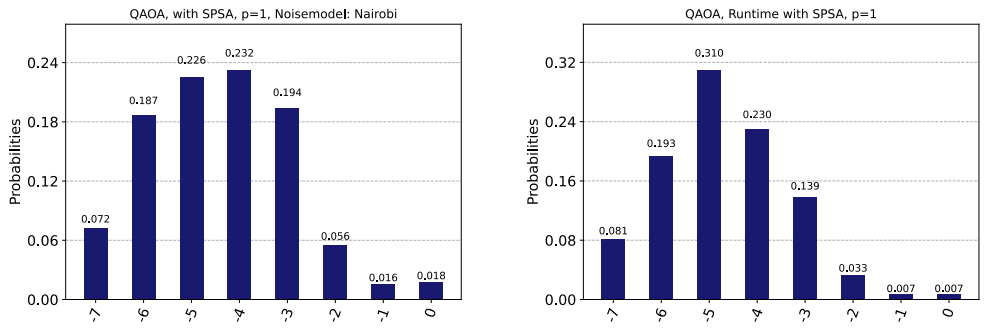
Figure 4.8: The QAOA circuit used in this use-case. The first box is implemented with the RZZ-gates and the second box are R_X-gates.

The QAOA-module in Qiskit is usable with minimal knowledge, it only needs an operator and a quantum backend. Other inputs are optional. The default optimizer is SPSA, the hyperparameter p is set to 1 and the default number of shots is 1024. With the backend configuration of the "ibm_nairobi" the number of shots is 4000. With this configuration, the Max-Cut was not found often.

Without explicit guidance on how to configure the parameters for the execution does not bring the desired results. To be reusable as software and usable for real-world applications, the default properties need to be tweaked. In this case, the goal was to improve the results from Figure 4.9 by choosing other parameters to modify depths and exchanging the optimizer. The observations were, that no reliable improvement of the results with the used noise-model could be achieved.

By transpiling the quantum circuit for the respective backend, 8 swaps were necessary which increases the depth and respectively the noise. Another point is, that qubits or connections between them must be used no matter how big the error rate is, as the encoded problem instance occupies all available qubits.

The best result was found by using the COBYLA optimizer with p=1. Other settings lead to no further improvement or the results got worse. Higher reps should lead to an improved result, but with NISQ-machines more depth also means more noise. The conclusion here is, that in practice with the used noise model, the choice of the hyperparameter is not trivial. The Qiskit runtime results with the real quantum computer are similar in terms of the probability of finding the Max-Cut of 7. Similar to the experiment in [BP21], the hybrid execution environment has no perceivable influence on the results. The differences can be explained by the different noise profiles of the quantum device and the used noise model, which can explain the existing variance in the results.



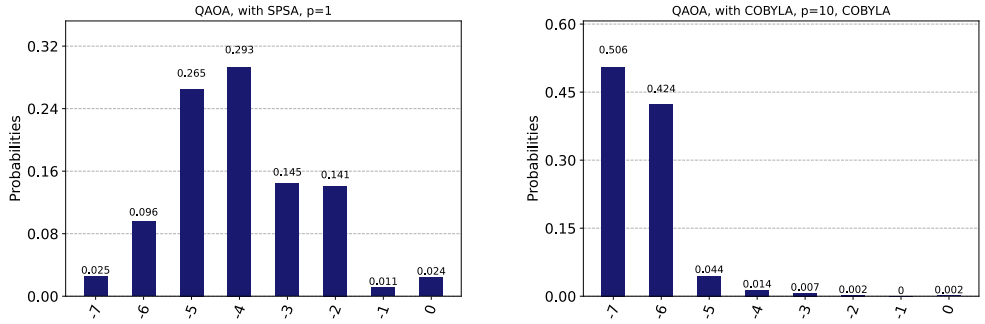
(a) QasmSimulator results.

(b) QAOA runtime results.

Figure 4.9: QAOA results with the same parameters. p=1, SPSA, noise-model and settings from ibm_nairobi, runtime program executed on the ibm_nairobi with swap strategies to reduce the cicruit depth.

Although better results were achieved with another optimizer, the concept and the importance of guidelines and well-set default parameters are shown. These were very simple modifications with the *QasmSimulator* without a noise model. With lower error rates and more qubits, more reliable results can be expected when these algorithms are used.

In Figure 4.10 the results with low depth and standard configuration yield only slightly improved results, than the execution with noise model or on real hardware. The results can be improved with the increased depth and the COBYLA optimizer, where the inspiration was obtained from other papers. These results are only possible because the noise does not increase with our increasing circuit depth as the noise-free *QasmSimulator* was used and the COBYLA does perform well in a setting without noise. It only shows the importance on the chosen components and parameters and need to be taken with care, as this result quality is only possible when error correction is possible or when the qubits do not suffer from the noise like on todays quantum devices.



(a) QAOA Max-Cut results with the same settings as Figure 4.9 without a noise-model.

(b) QAOA Max-Cut results with higher depth and COBYLA-optimizer without a noise-model.

Figure 4.10: QAOA results on the *QasmSimulator*.

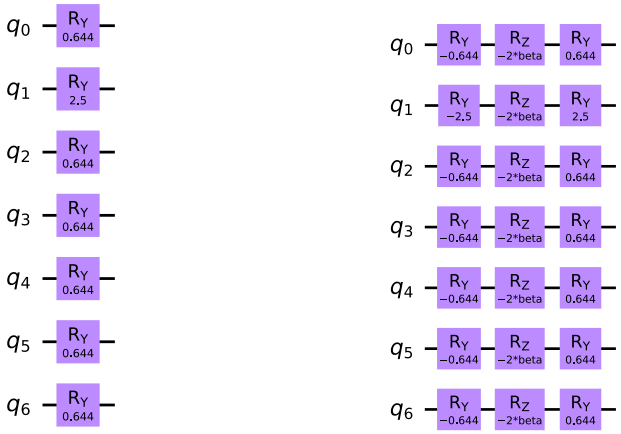
Warm-Start QAOA with classical pre-processing

Here the warm-start QAOA as explained in detail in Chapter 3 is tested. For the warm-starting example, the problem instance in Figure 4.6 from the standard QAOA was reused. One question in this context could be, how much classical pre-processing is needed for the algorithm to calculate the Max-Cut? With the additional pre-processing part of finding a good start solution, the classical part is computationally more expensive than in normal QAOA. That processing time needs to be considered when the processing time of the whole algorithm is analysed. The findings in [BP21] are, that the processing time from WS-QAOA is much longer than other Variational Quantum Algorithms, but WS-QAOA found the solution where the QAOA and the VQE did not. Many of the examples seen online use the GW-Algorithm like in [TBB+22].

Here the warm-start of the algorithm was manually done by providing a cut as a bit-string. The algorithm for finding a warm-start solution is purely classical, therefore the function call of the algorithm can be mocked by a manual input. The Max-Cut solution of this instance is 7 with different combinations. The inverse of the bit-string is always a solution too as in the QAOA use-case. In the context of approximation algorithms, a classical algorithm is bounded in terms of solution quality. That means, how close is the approximated solution to the actual solution. The goal here is to improve the classicaly calculated approximation. An approximation, that can be achieved by choosing node 1 leading to a ratio of $\frac{5}{7} \approx 0,714$, which the WS-QAOA should improve. The approximations in [TBB+22] are better with a minimum of 0.8835. Besides the warm-start solution, the other inputs are kept constant, like in the QAOA case.

It is variant of the QAOA is when the state preparation and the mixer Hamiltonian are modified. The state preparation of the standard QAOA is implemented with Hadamard to create a uniform superposition. The WS-QAOA performs the state preparation with $R_Y(\theta)$ where $\vec{\theta}$ encodes solutions obtained by a classical optimizer. To decide, which angles to use, the approximated solution with a hyperparameter ϵ is used to calculate.

In Figure 4.11 on the left, the state preparation is shown. The angles of this circuit encode a solution, where qubit 1 rotates with a different angle than all the other qubits. The modified mixer uses the same angles for the R_Y-rotations. The parameter list $\vec{\beta}$ are angles as used in the QAOA which are often initialized randomly.

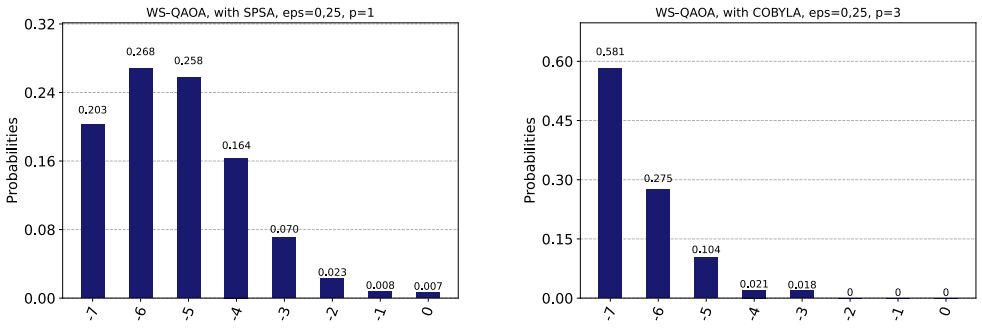


(a) WS-QAOA State preparation **(b)** WS-QAOA Mixer

Figure 4.11: Circuit modifications for the warm-starting QAOA, with the altered angles. The warm-start solution was 0100000, which can be seen with the rotations done to q_1.

The results in Figure 4.12 in the left diagram on how often the Max-Cut is found have significantly improved and the warm-started solution was measured more often. The improvement was achieved with usage of a user with minimal quantum knowledge, as every parameter kept on default.

The Max-Cut solutions have different sets and by warm-starting in one direction, the probability to measure another solution is lowered. With the hyperparameter ϵ, one can decide how much weight the warm-start solution should have. It was chosen with 0.25, which is balanced within the allowed range $[0, 0.5]$. A value close to zero weighs the warm-started result heavily, while 0.5 results in a non warm-started QAOA execution. The choice of p=3 was done through knowledge from previous experiments and with the circuit depth in mind. With this choice, the WS-QAOA produced slightly better results, than QAOA, but by using a quantum circuit with smaller depth.



(a) WS-QAOA Max-Cut Results without noise **(b)** WS-QAOA Max-Cut Results without noise

Figure 4.12: WS-QAOA Max-Cut Results without noise and warm-started with 0100000 as depicted in Figure 4.6.

The warm-start introduces another hyperparameter to consider because now the weight of the warm-start solution needs to be chosen. Here the classical part plays a larger role in the overall algorithm. Classical approximation algorithms, which can be done without any quantum-related knowledge. A split between classical software developers and quantum experts can be set at the point of translation from the classical solution to quantum circuit parameters. In our example, the classical developer could provide the implementation of the GW-algorithm, which was omitted, as the warm-starting solution was chosen manually. The circuits need to be implemented with the chosen hyperparameter to build the circuit with the respective rotations.

The takeaway here is, that not only does the split-point of the algorithm need to be set, during the design, but also the implementation split-point which were different in all examined implementations. More precisely with respect to the hyperparameters. That means, every choosable hyperparameter is a split-point, where the quantum expert can decide how these are set. In the case of QAOA, the hyperparameter p influences the depth, which means, that as soon as the quantum circuit is transpiled for the respective device, it could be checked, whether the decoherence time allows extending the depth of the circuit by another repetition.

5 Evaluation and Findings

In this chapter the split-points found in the examined algorithms are evaluated. In Chapter 3 the split-points were examined from the theoretical descriptions of the algorithms. The examined Variational Quantum Algorithms (VQA) were the Variational Quantum Eigensolver (VQE), the Quantum Approximation Optimization Algorithm (QAOA) and the warm-started QAOA (WS-QAOA). Split-points are found at the entrance of the hybrid loop from classical to quantum and inside the hybrid loop after every measurement. To verify the theoretical findings in Chapter 3 some example use-case scenarios of the algorithms mentioned above were implemented manually in Chapter 4 to examine how the split-points are set. A split-point in implementations is set, where the code manipulates quantum circuits. The findings are, that these split-points are set differently throughout the tested implementations. The split-points found in Chapter 4 in particular differ from the split-points found in the theoretical descriptions.

In order to encapsulate the ideas of how to set split-points, the pattern format is used. The pattern finding process is used as method to structure the pattern identification. The phases of the pattern finding process from [FLR+14] are depicted in Figure 5.1. In the *Pattern Identification* phase data is collected and prepared in the steps printed below the phase. These steps filter and homogenize information, such that a team of pattern researchers can work efficiently together. In the *Pattern Authoring* phase, the patterns are created based on the information of the previous phase. The patterns are often related and can form a pattern language. The *Pattern Application* phase describes the process of how to use patterns once there is a pattern language. This phase is independent of the other phases and not directly addressed here.

The Quantum-Classic Split pattern and the related patterns provided the domain to search for new candidates. These patterns influenced the algorithm selection and which algorithms are examined in more detail. With this information selection, findings are written as pattern-candidates in the *Pattern Identification* phase. Pattern-candidates are structured information collections which could help in the *Pattern Authoring* phase. In Figure 5.1 the pattern-candidates in this section are located in the *Information Collection*. The related patterns and the suggested way to follow a solution path [Ley19] inspired the way to format the information and link the findings to other patterns.

The findings are also related to the quantum software lifecycle where the descriptions of the phases are augmented to accommodate the findings. The additions are done with the pattern-candidates and a short explanation of the relation to the respective phase.

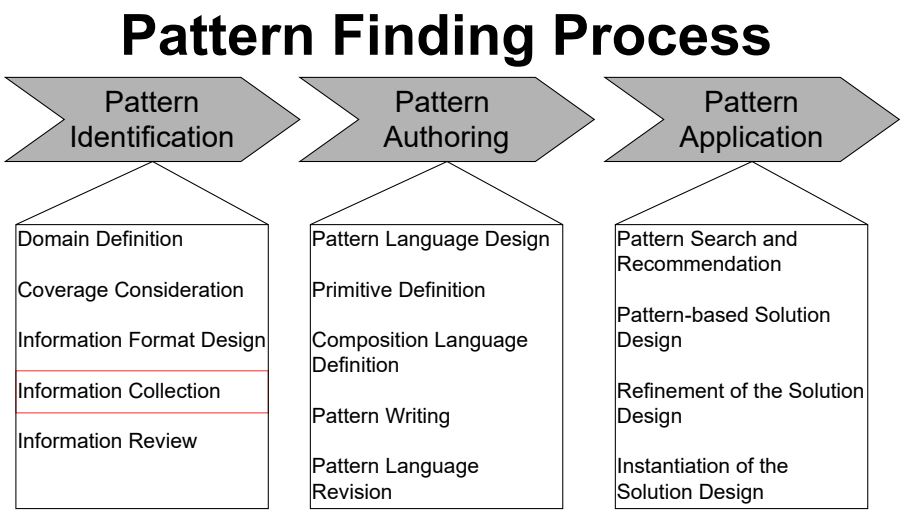
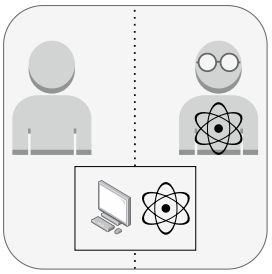


Figure 5.1: Pattern Finding Process, adapted from [FLR+14]. The pattern-candidates are information collections for further pattern research.

5.1 Quantum-Expert Split

The *Quantum-Expert Split* states, that every expert has a specific domain of expertise and thus should be working in that domain. The problem with quantum computing is, that the quantum expert is not necessarily an expert in the explicit use-case for a quantum algorithm. Furthermore, quantum computing experts are not as common as experts in established fields because quantum computing is a new research field compared to the other domains. The abstract idea of this pattern-candidate is that the quantum expert needs to design the implementation in a way, that non quantum experts can use it minimizing the chances of mistakes that are caused by a lack of understanding of quantum algorithms by choosing syntactically correct but bad inputs in the context of the algorithm.



How can the development of a quantum algorithm be distributed efficiently in a team of developers?

Context

Quantum algorithms, specifically Variational Quantum Algorithms (VQA), contain classical parts and quantum parts. When such algorithms are developed, quantum knowledge and a deep understanding of the problem is necessary. Both parts need to work together, where the classical part is mainly there to enable the execution of the quantum part. With the Quantum-Classic Split,

the idea arises, that similar to the split of the algorithm, the development could be split in order to utilize the available resources. Classical software engineering has many tools which can aid the development, but they are only available for classical parts. For the quantum parts, quantum circuits need to be implemented with the given libraries [Qis22e]. For this implementation, a quantum expert is indispensable, because there must be a split between the quantum part and the classical part as it is stated in the Quantum-Classic Split pattern. Implementations of the quantum parts can influence the implementation of classical parts, therefore quantum knowledge can be necessary in classical parts. The split-points need to be set to divide the domains in order to distribute the implementation tasks. A major force is, that setting these split-points is not trivial and a quantum expert is not necessarily as experienced with classical software engineering, therefore the trivial solution to only let quantum experts work on the quantum parts.

Solution

The solution is to distribute the development of the algorithm between the domain experts in such a way that the developers working in their domain of expertise. In particular, the quantum circuit should be implemented by the quantum expert. The implemented quantum part can potentially be provided as a black box function, which is usable with minimal quantum computing knowledge. Classical parts in the pre-processing or post-processing can be done by a classical software engineer with the required domain knowledge for the problem. Before implementing an algorithm, a domain experts needs to identify hard to calculate parts that can potentially benefit from quantum computing. These parts then must be reviewed together with a quantum expert, to decide whether or how they can be implemented on a quantum computer and where in the classical pre-processing the split-point should be located. This can be done, e.g., by marking split-points in flow diagrams or the pseudo code of these algorithms. The benefit of this process is, that it avoids time-consuming actions from the quantum expert to review every detail of the algorithm because the domain expert already preselected interesting parts.

Known Uses

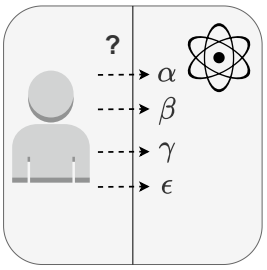
The first example in Chapter 4 was the random number generator, where the quantum circuit for the random number generator can be implemented by a quantum expert and supplied as a black box function with classical in- and outputs. No quantum computing knowledge is required to interact with the packaged algorithm. Shors' algorithm as described in [Hom18], marked the computational complex part, which was split in two parts and the quantum part executed on the quantum computer. Examined use-cases of VQA are the VQE and the QAOA, where the modules act as a black box, where the user only needs to encode the problem statement. In the VQE example in Chapter 4, most of the code that needed to be implemented was classical. The implementation of the quantum computing part is hidden inside modules which also include the classical the post-processing. The publicly available QAOA runtime program also limits the input for a specific use-case.

Related Patterns

The *Quantum-Expert Split* can be applied whenever a Quantum-Classic Split is applied when an algorithm is implemented. It is also related to the VQA pattern and the Warm-Start pattern. When the *Quantum-Expert Split* is applied, developers get a more precise description of how to design implementations of these algorithms.

5.2 Hyperparameter-Split

The *Hyperparameter-split* pattern-candidate is a refinement of the *Quantum-Expert Split* pattern-candidate. As already established quantum experts need to design a quantum module in a way, that users avoid input mistakes. When a quantum circuit is involved it is pretty clear, that all inputs or parameters altering the quantum circuit requires expert knowledge. In this case, other parameters are examined too. As a Variational Quantum Algorithm consists of quantum parts and classical parts in the hybrid loop and there is room for non-quantum experts to implement modules.The goal of the *Hyperparameter-split* pattern-candidate is to provide further instructions on how to set the split-point oriented on hyperparameters.



How can a quantum algorithm which can be controlled by setting parameters be used with minimal quantum computing experience?

Context

Quantum algorithms, where the user input is only the problem statement itself can be checked with, e.g., the NISQ-analyzer [SBB+20] to determine whether it can be executed on a given quantum computer. Some of the most relevant factors are the number of qubits or the decoherence time. An implemented VQA often needs fine-tuning of parameters. Whether a VQA succeeds in its execution and outputs a useful result relies on the choice of one or often more parameter. The analysis in terms of depth and width of the quantum circuit is not enough to consistently produce reasonable results. Choosing a hyperparameter big as in the QAOA-context to increase the chance to obtain a good result may result in a noisy measurement and an optimizer with a smaller learn rate. The red lines in Figure 5.2 are possible split-points, where the left line indicates, that it is quantum expert domain and the right line indicates, that it can be a classical developers task.

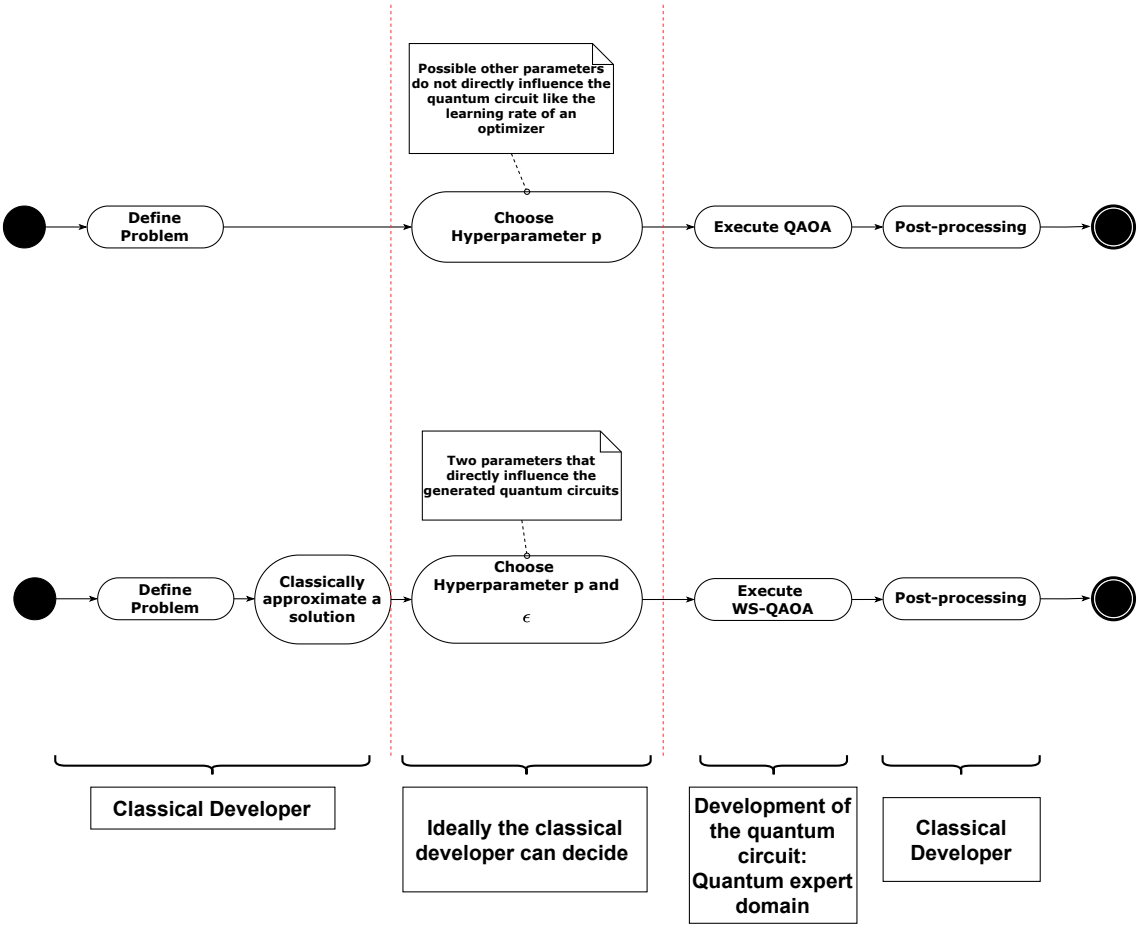


Figure 5.2: Hyperparameter-Split sketch with QAOA (top) and WS-QAOA (bottom).

Solution

When implementing a quantum algorithm as a usable module, provide the best possible default value. If that works every time hide the parameter completely otherwise expose the parameter within the documentation. A way is to give precise instructions on how to choose the parameters. If guidelines for how to avoid bad values are known, then they should be included in the documentation of that parameter. Additionally, the module that receives the parameters can provide helpful information to the user of what went wrong, e.g. depth exceeded limits because p was set too high. Whenever hyperparameters are involved, it can be considered as a split-point. When estimations about these hyperparameters are done, it can be decided whether it is in the quantum domain or can be placed in the classical domain to make it reusable.

Known Uses

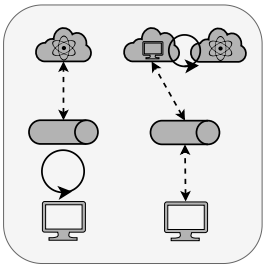
The QAOA-runtime program, examined in Chapter 4 has default optimizer settings, when using the algorithm. The importance of a well-chosen hyperparameter in the context of WS-QAOA can be found in [TBB+22], where the problem instance and the cost function influence the result quality.

Related Patterns

The *Hyperparameter-Split* pattern-candidate refines the *Quantum-Expert Split* with a more concrete description on how to set split-points. The VQA pattern is a use-case of the *Hyperparameter-Split*, as VQAs often rely on hyperparameters.

5.3 Hybrid Execution Environment

The execution of quantum algorithms on real hardware uses concepts of the cloud computing domain. Here the communication and the cloud symbols are inspired by cloud computing patterns [FLR+14]. The hybrid execution environment provides an alternative execution variant for quantum algorithms. Both, classical part and quantum part are executed in the cloud instead of a distributed execution between the users computer and the cloud. It is an already existing technology in form of the Qiskit Runtime [Qis22k] or Amazon Hybrid Jobs [Ama22c]. The information condensed in this pattern-candidate was identified with use-cases in Chapter 4 and tests from [BP21; TBB+22]. As the pattern-candidate references the VQA-pattern and the cloud computing patterns it connects those two languages.



How can a quantum algorithm with a hybrid loop be executed efficiently on a quantum computer?

Context

Quantum computers today and in the foreseeable future are still accessed via cloud. The way the users' computer interacts with the quantum computer is through a service from the provider over a job queue. This is the reason, that quantum computing should not be seen as a purely mathematical and physical phenomenon. It is also the reason, that classical code is necessary to send quantum circuit to the cloud. The overhead because of networking can negatively impact the execution time. With too many jobs for too few quantum computers, the waiting time in the queue can diminish the return of a quantum advantage in terms of execution time.

Solution

If a quantum algorithm relies on an execution with many circuit jobs, a hybrid execution environment is recommended. A hybrid execution environment is an environment which can execute both classical code and quantum code. The only network activity during the execution between the user and the cloud service is the transmission of the input and the results. This leads to a lower network

overhead, as not every single quantum circuit must be sent to the cloud. The major time save is, that every quantum circuit job is executed inside of one program job. Especially the waiting time in occupied queues can lead to a much higher process time.

It must be mentioned, that the implementation can differ, especially when setting a split-points between classical parts to pre-process something on the users side and then send the serialized results as input to the hybrid execution environment when invoking the program in the cloud. The classical parts of the hybrid loop of Variational Quantum Algorithms should be placed in the hybrid execution environment. Beware, that this pattern cannot be applied if the classical code must not be executed off premise for, e.g., confidentiality reasons.

Known Uses

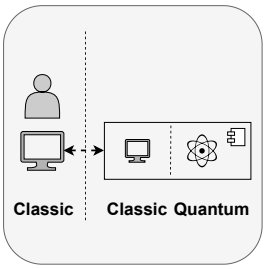
IBM offers a hybrid execution environment under the name Qiskit Runtime [Qis22k]. The Qiskit Runtime was used in Section 2.4 for a QAOA implementation and explained in Section 2.4. Another hybrid execution environment service is provided by Amazon with Braket Hybrid-Jobs [Ama22b].

Related Patterns

The pattern-candidate is very useful in combination with the VQA pattern. The *Two-Tier Cloud Application* from the cloud computing pattern language [FLR+14] relates to the pattern-candidate. Here the two tiers can be seen as the classical computer and the quantum computer, with both parties in the cloud.

5.4 Classic-Classic Split in Quantum Algorithms

In the test cases in Chapter 4, the problem is converted to an operator, which is sent serialized with the other runtime configuration. This is the use-case, when the problem definition is converted on the users machine. More general, the preparation does not necessarily need to be done on the users side. It could be done by invoking the program with serialized data of the graph in our test. Then the operator could be built in the cloud. Setting the split-point deliberately could result in a better way to organize reusable code fragments.



Are there other ways to split a quantum algorithm, besides the split between a quantum part and a classic part?

Context

With the standard workflow of a quantum algorithm, the classical part is executed on the user's side and the quantum circuit on a quantum computer, which is accessible via cloud service. With the hybrid execution Environment, the classical code is sent as a package together with the quantum code for execution in the cloud. As the classical code also runs in the cloud, the execution of the classical code must also be paid for. In the case of classical warm-starting, the execution runtime can be very long [BP21].

Solution

When a new quantum algorithm is designed and the *Quantum-Expert Split* is applied to decide which parts are necessary to calculate on the quantum computer, the remaining parts are classical software. As in the context of classical software engineering, these parts can be provided as modules or services. Quantum computing is provided as a cloud service and specifically, the hybrid execution environments can be used. They are hybrid, because they execute user-provided classical code and quantum code. Splitting the classical parts, in order to execute not the entire algorithm in the cloud, but pre-process to a certain extent and send the results as serialized data to the hybrid execution environment, where the other part is calculated, followed by the interaction with the quantum hardware. A well-set split-point can give a user guidance in itself or restrict the usage on a selected set of problem instances.

In [BP21], the pre-processing of the classical warm-start leads to an increased execution runtime. Execution runtime is either limited [Qis22k] or can be calculated with price/time [Ama22c]. The classical pre-processing time can be estimated and then calculated whether it is beneficial to process parts of the pre-processing locally.

Besides the prominent examples, the abstract description here is, that the split-point is often set to get reusable classical code and classical code, which is near the quantum part, which cannot be easily implemented reusable. Another aspect is the privacy aspect. The classic-classic split does not fully solve the privacy problem as data is still transferred, but it can be argued, that Python code with the problem instance as input can be considered more problematic than pre-computed solutions and a parametrized quantum circuit. In the latter case, the information provided to the service provider is very limited.

Related Patterns

The *Classic-Classic Split* abstracts the *Hybrid Execution Environment* and the *Quantum-Expert Split*, as it splits the execution at a deliberately set split-point. It is related to the Quantum-Classic Split, as it splits code of a quantum algorithm into different classical parts.

5.5 Proposed Pattern-Candidates in the Pattern Landscape

The pattern-candidates need to be placed in a pattern language in order to be included in a solution path, described with the idea of a graph like in [LB21]. The pattern-candidates are found through the usage and research of Variational Quantum Algorithms. With that starting point, the pattern-candidates are placed in the quantum computing landscape according to the graph from [WBLV21]. In Figure 5.3 the pattern-candidates are inserted with connections to already existing patterns.

The *Classic-Classic split* is very abstract and states the possibility to split classic parts to be reusable by deliberately splitting the software and putting them into modules, as shown in Chapter 4, where variants of different split-points are shown. In the VQE, the ansatz needs to be built with a module, where in the case of the QAOA this ansatz is already part of the module.

The *Hybrid Execution Environment* provides another way to split the classical parts. The reusable modules on the local machine and the classical parts in the cloud need to work together with the quantum part. This concept has a connection to the Cloud Computing domain, as it uses a modified two-tier cloud application. A connection to the VQA pattern is also there because it is advised to execute these algorithms in the hybrid execution environment. The *Quantum-Expert split* and the refinement, the *Hyperparameter-Split* is directly connected to the *Quantum-Classic Split* pattern.

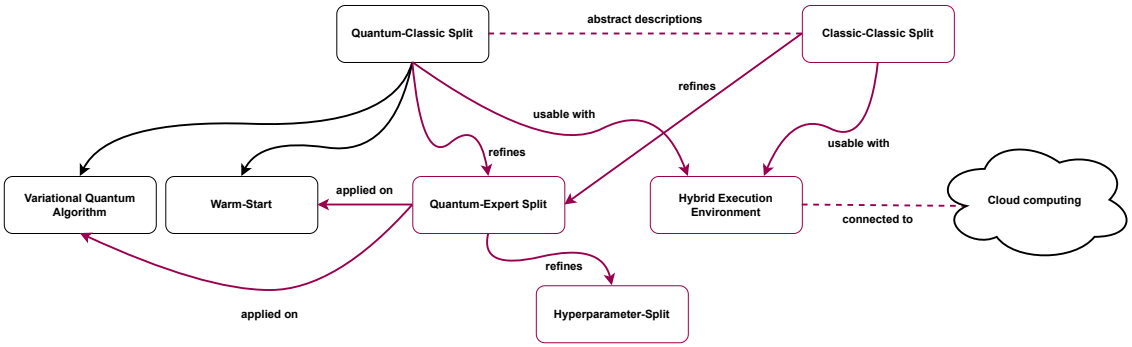


Figure 5.3: The pattern landscape of quantum computing pattern related to the Quantum-Classic Split pattern. New pattern-candidates are printed in red. The connection to the Cloud Computing domain is also shown as in the pattern atlas where domains are connected through related patterns. Adapted and extended from [WBLV21].

5.6 Additions on the Quantum Software Lifecycle

In this section, the phases of the quantum software lifecycle proposed by Weder et. al. [WBL+20] are examined. The quantum software lifecycle is a structured description of the development of a quantum algorithm. As it is suggested in the discussion part of [WBL+20] the quantum software lifecycle is used as a baseline, hence, in this section, the pattern-candidates are added to some phases while other phases are left as they were in the original paper and just explained briefly.

Some phases already mention VQAs, but the descriptions are very abstract. The phases *Hardware-independent Optimization* and *Readout-Error Mitigation* can be used for VQA similar to pure quantum algorithms. They are part of the quantum software lifecycle but are not addressed in this context. The importance of these phases is unchanged and needs to be followed in the process.

Quantum-Classical Splitting

This phase splits the quantum algorithm into classical parts and quantum parts. The addition to this phase are in the pattern-candidates which also describe how to set split-points in order to distribute the implementation task between quantum experts and classical software engineers. For quantum algorithms, without a hybrid loop, the *Quantum-Classical Splitting* step is already difficult and should be supported by automatic recommendations [WBL+20]. For hybrid algorithms, not only the classical pre-processing and post-processing needs to be considered. The *Quantum-Expert Split* presented in this work encapsulates the criteria for the split, but when designing the hybrid loop, the *Hyperparameter-Split* gives further criteria to follow for better usability later on.

For Variational Quantum Algorithms, this phase is divided into 2 steps. The first is to divide the problem into a classical part and a quantum part. Then in the second step, the structure of a VQA is taken into account. The classical pre-processing is not only used to prepare the data, but to compress the problem into smaller or executable problems. A reference is to formulate a problem as a QUBO or an Ising-operator shown in Chapter 3 and Chapter 4.

The *Hybrid Execution Environment* opens the possibility of further splitting the execution. The reasons could be to split up the execution time. For fairness but also in an economic aspect, IBM limits the execution time, dependent on the user privileges, therefore it could be better, to execute parts locally. The payment for Amazon Hybrid Jobs, differs from the regular circuit jobs. A quantum circuit job is paid per execution and shots, while in the hybrid execution environment, the price is charged per minute. The *Classic-Classic Split* can be applied to set split-points in the classical code to decide which parts are executed locally and which parts in the cloud.

Hardware independent implementation

This phase makes use of the *Quantum-Expert Split*. The fragments of the program are divided in phase 1, therefore the implementation can be distributed accordingly. In the original formulation of this phase, the quantum expert should verify the quantum circuits, since quantum experts implement them in the first place and therefore verification is performed.

The extension to this phase is that the classical parts can be split deliberately, due to the fact, that Variational Quantum Algorithms can be implemented with the goal to use them in a hybrid environment. For quantum algorithms without a hybrid loop, this is also possible, especially when they have a *Classic-Classic Split*. However, they do not profit from the reduced network overhead as much as algorithms with many quantum circuit jobs.

Quantum circuit enrichment

This phase remains similar to its original formulation in the paper. The addition in this phase is done with the *Quantum-Expert Split* and the *Hyperparameter-Split*. The data preparation in this phase is defined in the classical part which is close to the quantum part. The Oracle, which is also described as a pattern in [Ley19] needs to be implemented with the respective function of the problem and then translated. The translation from the problem definition to the encoding scheme can be seen as a split-point between the classical domain and the quantum domain.

Quantum hardware selection

The choice of a suitable quantum computer heavily depends on the number of qubits and the error rates which are basically the metrics for limitations in the NISQ-era. For an optimized circuit from the last phase, quantum experts, who know with which quantum gates the ansatz was implemented, can consider choosing a quantum device with a suited quantum gate set. A quantum expert is needed to choose a suitable quantum computer because the depth of a quantum circuit might grow after the transpilation as shown in Chapter 2.

Integration

This phase opens one of the largest changes in the quantum software lifecycle. In the original paper [WBL+20], the integration and the execution step are the same due to the job-based system and the execution which is directly bounded to the deployment. With the *Hybrid Execution Environment*, the deployment of the code package is decoupled from the execution. The runtime program is executed, when it is invoked, not when it is deployed to the cloud. Furthermore, applying the *Classic-Classic Split* provides more freedom, but also more responsibilities for the developer.

Execution

In this phase, Variational Quantum Algorithms are already mentioned, because they execute the classical and quantum parts multiple times. The execution phase can be iterated several times, with new parameters. It can be more efficient to extensively execute with different parameters before completely going through every phase of the quantum software lifecycle.

This phase can be a cycle itself, together with the result analysis, where the result of the other phases is executed, analysed and then executed with other parameters or optimizers. This further enables developers to gain experience with the different components and how they work together. Additionally, the results can serve as examples for further research. Examples are [BP21; TBB+22] where the warm-start QAOA is tested, without extensively going through each phase.

Result analysis

This phase cannot be automated in most cases for several reasons. Therefore, applying the ideas of the *Quantum-Expert Split*, the quantum expert is not dispensable. First of all, Variational Quantum Algorithms are not usually used to find an exact result, but rather an approximated result. The best measure for this case is an approximation factor. However, a proof of concept execution is not enough to judge the quality of the algorithm because other instances might lead to other approximation factors. Another reason is, that multiple components of the algorithm can lead to incorrect or bad quality results. In Chapter 4, the results of the VQE were influenced by the optimizer and the ansatz, while [TBB+22] also showed an influence of the used cost function.

6 Related Work

The Quantum-Classic split is an open research topic and many aspects can be examined. Some related work presented here examine various Variational Quantum Algorithms and execute them in different ways. Other papers present quantum computing patterns and their relation to each other. Quantum software engineering uses the ideas of the Quantum-Classical split pattern and is also shown here as related work.

In [TBB+22] the components of Variational Quantum Algorithms are examined and tested in different ways. Different cost functions for the same problem instances, but also different hyperparameters and optimizing strategies. The examined algorithm was the WS-QAOA. The WS-QAOA applies the idea of the Quantum-Classic Split and is one of the prime use-cases for the development of further refined methods discussed in this paper. The Max-Cut problem was used as problem statement in the tests. Truger et. al. [TBB+22] used cost functions inspired by other domains. One example is the C_{CVaR} function where the critical value at risk (CVaR) is well-known in the finance domain and is not directly related to the quantum domain. With alternative cost functions, results were better with the tested Max-Cut instances. In particular, Truger et. al. tested the regulation hyperparameter ϵ with different cost functions. The hyperparameter depends on the cost function and the problem instance. Another aspect is the optimization strategy. There was a partial optimization strategy, where only a subset of the parameters are optimized and a regular optimization strategy where all parameters are optimized.

In [BP21] different Variational Quantum Algorithms are examined in terms of runtime and how accurate the results are. All tested algorithms are Variational Quantum Algorithms and make use of a Quantum-Classic Split. The problem statement was a classification problem solved with Max-Cut clustering. The motivation was to test the capabilities of WS-QAOA compared to the VQE and normal QAOA. The Qiskit runtime was used to test, whether the results are influenced by a hybrid environment and how it influences the execution time of an algorithm. The results of WS-QAOA are equally well in either execution, but a significant improvement in process time has been shown when using real quantum hardware. VQE and QAOA did not produce correct results. The conclusion is, that the Qiskit-Runtime does not influence results negatively and provides a significant speed-up in processing time with all examined algorithms.

In a larger scope, patterns for quantum computing presented in [Ley19; WBLS21; WBLV21]. The patterns in [Ley19] describe fundamental ingredients of a quantum algorithm. For example, it is described why and how state preparation works or how to create the often used uniform superposition. In [WBLS21] the focus lies on data encoding and preparation patterns for quantum computing. In [WBLV21] Weigold et. al. focus on the relations to the Quantum-Classic Split pattern, for example, Variational Quantum Algorithms and warm-start. With the patterns proposed, one can search for an entry pattern which fits the problem and then follow the path of the related patterns for a complete solution.

The quantum software lifecycle presented in [WBL+20] summarizes the process of the development of quantum algorithms. The Quantum-Classic Split pattern [WBLV21] is part of some phases inside the quantum software lifecycle. As mentioned in the paper, there are other software lifecycles for quantum software engineering, but the phases of the quantum software lifecycle [WBL+20] explicitly target forces of the implementation of quantum algorithms. In the NISQ-era some phases are very important, which are left out in other lifecycles mentioned in the paper. Specifically, the *Quantum-Classical Splitting* together with the *Readout-Error Mitigation* are phases, that can decide whether an algorithm is usable or not.

7 Conclusions and Outlook

The importance of the Quantum-Classic Split pattern is often overlooked, because it just states that a quantum algorithm consists of a classic part and a quantum part. If the split-point is near the quantum part, then the user has much freedom to optimize the quantum circuit with parameters or choose different suitable quantum circuits. However, working with quantum circuits requires knowledge about quantum computing. It may be better to provide a black box, for example as a Python module. The quantum expert is advised to keep hyperparameters and quantum circuits hidden, to make the module usable for users with minimal quantum knowledge. In the case of a Variational Quantum Algorithm, the optimizer should be chosen based on prior knowledge of the problem or through extensive testing. This way, users with no quantum computing experience can benefit from a quantum algorithm without making crucial mistakes, as there is a risk of choosing bad parameters or optimizers for the problem instance.

Variational Quantum Algorithms are used in many domains and have great potential for applications in the near term, but also beyond the NISQ-era. This is a major reason for the heavy focus on VQAs in this work. With the analysis of the split-points and concrete implementations, formalized ideas, guidelines and graphical representations are provided for a better understanding of the algorithm category. The examined algorithms are presented as workflows and as simple UML-activity diagrams. With these representations a developer gets an idea of how hybrid algorithms work.

Weder et. al. [WBLW20] propose QuantME, a conversion from a quantum workflow into an executable workflow. Further, well-crafted workflows for prominent algorithms could help create workflows for other Variational Quantum Algorithms and may give further hints on how to implement them. These approaches, which already aid classical software engineering can also help quantum-related implementations.

Especially the classical parts have proven to be reusable but need to be handled with care, for example optimizers have to deal with noise from the quantum computers and need to be adjusted with parameters. In the tests simulators were used which influences the choice of parameters and optimizers. During the research, some additional concrete recommendations were found beyond the goal to find split-points. Tested optimizers lead to good results, but these cannot be ensured when used with real quantum hardware. Guidelines need to be given in the context of the used machines. It needs to be discussed how big this gap between real quantum hardware and simulators is, but the noise or the available number of qubits influences the decision towards lower depth ansätze and noise resistant optimizers.

The pattern-candidates presented in Chapter 5 are the main findings of this thesis and could be further explored with more iterations in the pattern finding process. The idea from the Quantum-Classic Split is further refined in the *Quantum-Expert Split*, where the condensed idea is, that the algorithmic split-point does not generally equal the ideal split-point from a software engineering perspective.

The distribution of the implementation task between software engineers and quantum experts must be set as its own split-point where the quantum expert must decide how much of the implementation is tightly coupled to the quantum domain and which parts can be implemented classically.

A more concrete instruction, where such a split-point can be set, is the *Hyperparameter-Split*. With this instruction, the point in the code where a circuit-defining hyperparameter is chosen can be considered as a split-point, which dictates how the software can be used, tested or re-used. The *Hybrid Execution Environment* pattern-candidate tackles the problem of many iterations of quantum circuit jobs and additionally provides another way to split the code. The execution of the classical code can be split between the classical computer and the hybrid execution environment in the cloud. The *Classic-Classic Split* pattern-candidate is on the same abstraction level as the Quantum-Classic Split and states that for re-usability, privacy and the distribution of the execution, a split between the classical parts can be considered. A typical strategy is to set the split-point close to the quantum circuit implementation and divide the rest into general purpose implementations and problem and quantum-related parts.

The presented pattern-candidates are related to existing quantum computing patterns and even patterns of the Cloud Computing domain. The pattern candidates are described in Chapter 5 and relations are shown graphically in Figure 5.3. In quantum computing, one must address a wide range of forces when making use of theoretical concepts. More patterns with relation to quantum computing patterns with patterns which tackle economic, technological or privacy aspects could help developers or quantum experts to get a better idea of how to design algorithms and their implementation. Specifically, with the improvement of real quantum hardware in the future, the pattern-candidates can be evaluated with other use-cases for more precise instructions.

Bibliography

[ACSC20] A. Arrasmith, L. Cincio, R. D. Somma, P. J. Coles. "Operator Sampling for Shot-frugal Optimization in Variational Algorithms". In: arXiv, 2020. DOI: 10.48550/ARXIV.2004.06252 (cit. on p. 24).

[AIS77] C. Alexander, S. Ishikawa, M. Silverstein. "A Pattern Language: Towns, Buildings, Construction". In: New York: Oxford University Press, 1977 (cit. on p. 20).

[Ama22a] Amazon. *Amazon Braket*. 2022. URL: https://aws.amazon.com/de/braket/ (cit. on pp. 11, 26).

[Ama22b] Amazon. *Amazon Hybrid Jobs*. 2022. URL: https://docs.aws.amazon.com/de%5C_de/braket/latest/developerguide/braket-jobs.html (cit. on p. 59).

[Ama22c] Amazon. *Amazon Hybrid Jobs Overview*. 2022. URL: https://aws.amazon.com/de/about-aws/whats-new/2021/11/amazon-braket-hybrid-jobs-quantum-classical-workloads/ (cit. on pp. 12, 27, 58, 60).

[BP21] D. Beaulieu, A. Pham. "Max-cut Clustering Utilizing Warm-Start QAOA and IBM Runtime". In: arXiv, 2021. DOI: 10.48550/ARXIV.2108.13464 (cit. on pp. 36, 48, 49, 58, 60, 63, 65).

[BPM22] BPMN. *BPMN Overview*. 2022. URL: https://www.omg.org/spec/BPMN/ (cit. on p. 18).

[CAB+20] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, P. J. Coles. "Variational Quantum Algorithms". In: Nature Reviews Physics, 2020. DOI: 10.48550/ARXIV.2012.09265 (cit. on pp. 11, 22–25, 30–33, 37, 41).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. "Open Quantum Assembly Language". In: arXiv, 2017. DOI: 10.48550/ARXIV.1707.03429 (cit. on p. 16).

[Cro19] G. E. Crooks. "Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition". In: arXiv, 2019. DOI: 10.48550/ARXIV.1905.13311 (cit. on p. 25).

[EMW20] D. J. Egger, J. Marecek, S. Woerner. "Warm-starting quantum optimization". In: arXiv, 2020. DOI: 10.48550/ARXIV.2009.10095 (cit. on pp. 34, 35).

[FBBL14] C. Fehling, J. Barzen, U. Breitenbücher, F. Leymann. "A process for pattern identification, authoring, and application". In: *Proceedings of the 19th European Conference on Pattern Languages of Programs - EuroPLoP '14*. ACM Press, 2014. DOI: 10.1145/2721956.2721976 (cit. on p. 21).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. "A Quantum Approximate Optimization Algorithm". In: arXiv, 2014. DOI: 10.48550/ARXIV.1411.4028 (cit. on p. 32).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. “Cloud Computing Patterns”. In: Springer Vienna, 2014. DOI: 10.1007/978-3-7091-1568-8 (cit. on pp. 20, 53, 54, 58, 59).

[GAD+19] P. Gokhale, O. Angiuli, Y. Ding, K. Gui, T. Tomesh, M. Suchara, M. Martonosi, F. T. Chong. “Minimizing State Preparations in Variational Quantum Eigensolver by Partitioning into Commuting Families”. In: arXiv, 2019. DOI: 10.48550/ARXIV.1907.13623 (cit. on p. 17).

[HHL08] A. W. Harrow, A. Hassidim, S. Lloyd. “Quantum algorithm for solving linear systems of equations”. In: Physical review letters, 2008. DOI: 10.48550/ARXIV.0811.3171 (cit. on pp. 29, 36).

[Hom18] M. Homeister. “Quantum Computing verstehen”. In: Springer Fachmedien Wiesbaden, 2018. DOI: 10.1007/978-3-658-22884-2 (cit. on pp. 13, 14, 16, 17, 29, 30, 55).

[Jup22] Jupyter.org. 2022. URL: https://jupyter.org (cit. on p. 26).

[KACC19] J. M. Kübler, A. Arrasmith, L. Cincio, P. J. Coles. “An Adaptive Optimizer for Measurement-Frugal Variational Algorithms”. In: arXiv, 2019. DOI: 10.48550/ARXIV.1909.09083 (cit. on p. 24).

[Kim20] J.-S. Kim. *VQE Tutorial*. 2020. URL: https://www.youtube.com/watch?v=Z-A6G0WVI9w (cit. on p. 41).

[KRT07] J. Kempe, O. Regev, B. Toner. “Unique Games with Entangled Provers are Easy”. In: arXiv, 2007. DOI: 10.48550/ARXIV.0710.0655 (cit. on p. 34).

[LB20] F. Leymann, J. Barzen. “The bitter truth about gate-based quantum algorithms in the NISQ era”. In: vol. 5. 4. IOP Publishing, Sept. 2020, p. 044007. DOI: 10.1088/2058-9565/abae7d (cit. on p. 19).

[LB21] F. Leymann, J. Barzen. “Pattern Atlas”. In: *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future*. Springer International Publishing, 2021, pp. 67–76. DOI: 10.1007/978-3-030-73203-5_5 (cit. on pp. 20, 61).

[Ley19] F. Leymann. “Towards a Pattern Language for Quantum Algorithms”. In: *Quantum Technology and Optimization Problems*. Springer International Publishing, 2019, pp. 218–230. DOI: 10.1007/978-3-030-14082-3_19 (cit. on pp. 11, 20, 30, 53, 63, 65).

[MBS+18] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, H. Neven. “Barren plateaus in quantum neural network training landscapes”. In: arXiv, 2018. DOI: 10.48550/ARXIV.1803.11173 (cit. on p. 25).

[Mic22] Microsoft. *MS Azure Overview*. 2022. URL: https://azure.microsoft.com/de-de/services/quantum/#overview (cit. on p. 11).

[NC11] M. A. Nielsen, I. L. Chuang. “Quantum Computation and Quantum Information: 10th Anniversary Edition”. In: Cambridge University Press, 2011. ISBN: 9781107002173 (cit. on pp. 11, 13–15, 17, 29, 30).

[OGB19] M. Ostaszewski, E. Grant, M. Benedetti. “Structure optimization for parameterized quantum circuits”. In: *Quantum* (2019). DOI: 10.48550/ARXIV.1905.09692 (cit. on p. 25).

[Pat22] Q. Patterns. 2022. URL: https://quantumcomputingpatterns.org/#/ (cit. on p. 22).

[PMS+13] 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 quantum processor". In: arXiv, 2013. DOI: 10.48550/ARXIV.1304.3061 (cit. on p. 31).

[Pre18] J. Preskill. "Quantum Computing in the NISQ era and beyond". In: Quantum, 2018. DOI: 10.48550/ARXIV.1801.00862 (cit. on pp. 11, 19).

[Qis22a] Qiskit. *GroundStateEigensolver*. 2022. URL: https://qiskit.org/documentation/nature/stubs/qiskit_nature.algorithms.GroundStateEigensolver.html (cit. on pp. 41, 42).

[Qis22b] Qiskit. *IBM Systems overview*. 2022. URL: https://quantum-computing.ibm.com/services?services=systems (cit. on pp. 26, 28).

[Qis22c] Qiskit. *QAOA Runtime Tutorial*. 2022. URL: https://qiskit.org/documentation/optimization/tutorials/12_qaoa_runtime.html (cit. on p. 45).

[Qis22d] Qiskit. *QAOA Tutorial*. 2022. URL: https://qiskit.org/documentation/tutorials/algorithms/05_qaoa.html (cit. on p. 45).

[Qis22e] Qiskit. *Qiskit documenation*. https://qiskit.org/documentation/, 2022. URL: https://qiskit.org/documentation/ (cit. on pp. 11, 26, 27, 55).

[Qis22f] Qiskit. *Qiskit Groundstate Tutorial*. 2022. URL: https://github.com/Qiskit/qiskit-nature/blob/stable/0.4/docs/tutorials/03%5C_ground%5C_state%5C_solvers.ipynb (cit. on p. 41).

[Qis22g] Qiskit. *Qiskit UCC Ansatz*. 2022. URL: https://qiskit.org/documentation/nature/stubs/qiskit_nature.circuit.library.UCC.html#qiskit_nature.circuit.library.UCC (cit. on p. 31).

[Qis22h] Qiskit. *Qiskit VQLS*. 2022. URL: https://qiskit.org/textbook/ch-paper-implementations/vqls.html (cit. on p. 37).

[Qis22i] Qiskit. *Qiskit Warm Start Tutorial*. 2022. URL: https://qiskit.org/documentation/optimization/tutorials/10_warm_start_qaoa.html (cit. on p. 35).

[Qis22j] Qiskit. *Qiskit-Nature Ansätze*. 2022. URL: https://qiskit.org/documentation/nature/apidocs/qiskit%5C_nature.circuit.library.html (cit. on pp. 32, 42, 44).

[Qis22k] Qiskit. *Qiskit-Runtime*. 2022. URL: https://qiskit.org/documentation/partners/qiskit_ibm_runtime/ (cit. on pp. 12, 27, 58–60).

[Qis22l] Qiskit. *Qiskit-Runtime Tutorial*. 2022. URL: https://github.com/Qiskit-Partners/qiskit-runtime/blob/main/tutorials/02_uploading_program.ipynb (cit. on p. 28).

[Qis22m] Qiskit. *SPSA Documentation*. 2022. URL: https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.SPSA.html (cit. on p. 44).

[Qis22n] Qiskit. *WS-QAOA Tutorial*. 2022. URL: https://qiskit.org/documentation/optimization/tutorials/10_warm_start_qaoa.html (cit. on p. 45).

[Rig22] Rigetti. *Rigetti Homepage*. 2022. URL: https://www.rigetti.com (cit. on pp. 11, 26).

[Ros21] M. Rossmannek. *VQE presentation*. 2021. URL: https://www.youtube.com/watch?v=UtMVoGXlz04 (cit. on p. 41).

[SBB+20] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild. "The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms". In: *Service-Oriented Computing*. Springer International Publishing, 2020, pp. 66–85. DOI: 10.1007/978-3-030-64846-6_5 (cit. on p. 56).

[SBG+18] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran. "Evaluating analytic gradients on quantum hardware". In: Physical Review A, 2018. DOI: 10.48550/ARXIV.1811.11184 (cit. on p. 25).

[SBLW20] M. Salm, J. Barzen, F. Leymann, B. Weder. "About a Criterion of Successfully Executing a Circuit in the NISQ Era: What wd $\ll 1/\epsilon_{Eff}$ Really Means". In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*. ACM, Nov. 2020. DOI: 10.1145/3412451.3428498 (cit. on pp. 11, 19, 20).

[Sho95] P. W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: STAM review, 1995. DOI: 10.48550/ARXIV.QUANT-PH/9508027 (cit. on pp. 26, 29).

[SIKC19] J. Stokes, J. Izaac, N. Killoran, G. Carleo. "Quantum Natural Gradient". In: Quantum, 2019. DOI: 10.48550/ARXIV.1909.02108 (cit. on p. 24).

[Spa01] J. C. Spall. *SPSA Overview*. 2001. URL: https://www.jhuapl.edu/SPSA/ (cit. on p. 24).

[TBB+22] F. Truger, M. Beisel, J. Barzen, F. Leymann, V. Yussupov. "Selection and Optimization of Hyperparameters in Warm-Started Quantum Optimization for the MaxCut Problem". In: *Electronics* 11.7 (Mar. 2022), p. 1033. DOI: 10.3390/electronics11071033 (cit. on pp. 49, 57, 58, 63–65).

[UML22] UML. *UML specification*. 2022. URL: https://www.omg.org/spec/UML/ (cit. on p. 18).

[WBL+20] B. Weder, J. Barzen, F. Leymann, M. Salm, D. Vietz. "The Quantum software lifecycle". In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*. ACM, Nov. 2020. DOI: 10.1145/3412451.3428497 (cit. on pp. 11, 12, 61–63, 66).

[WBLS21] M. Weigold, J. Barzen, F. Leymann, M. Salm. "Encoding patterns for quantum algorithms". In: vol. 2. 4. Institution of Engineering and Technology (IET), Dec. 2021, pp. 141–152. DOI: 10.1049/qtc2.12032 (cit. on pp. 20, 65).

[WBLV21] M. Weigold, J. Barzen, F. Leymann, D. Vietz. "Patterns for Hybrid Quantum Algorithms". In: *Service-Oriented Computing*. Springer International Publishing, 2021, pp. 34–51. DOI: 10.1007/978-3-030-87568-8_2 (cit. on pp. 11, 20, 22, 30, 32–34, 61, 65, 66).

[WBLW20] B. Weder, U. Breitenbucher, F. Leymann, K. Wild. "Integrating Quantum Computing into Workflow Modeling and Execution". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, Dec. 2020. DOI: 10.1109/ucc48980.2020.00046 (cit. on pp. 18, 41, 47, 67).

All links were last followed on September 19, 2022

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature