DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

A Novel Approach for Solving Constrained Optimization Problems with QAOA using Encoders

Burak Mete

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master′s Thesis in Informatics

A Novel Approach for Solving Constrained Optimization Problems with QAOA using Encoders

Ein neuartiger Ansatz zur Lösung von eingeschränkten Optimierungsproblemen mit QAOA unter Verwendung von Encodern

Author:	Burak Mete
Supervisor:	Irene Lopez Gutierrez
Advisor:	Prof. Dr. Christian Mendl
Submission Date:	15.05.2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.05.2022

Burak Mete

[illegible]

Abstract

The Quantum Approximate Optimization Algorithm (QAOA) [1] is a meta-heuristic gate-model quantum algorithm, mainly used for solving combinatorial optimization problems, such as Max-Cut, or the Travelling Salesman Problem. The main goal of the algorithm is to find an upper bound to the ground state energy, using the *cost Hamiltonian*, which is based on the objective function of the optimization problem. Usually, the solutions to such an optimization problem are provided as a binary string, therefore any computational basis state can be considered as one solution to the optimization problem. In the algorithm, the application of cost Hamiltonian achieves *phase separation*, in relation to the costs of each solution. The algorithm also includes another Hamiltonian called the *mixer*, and it is defined as a Hermitian matrix that does not commute with the cost Hamiltonian. The mixer Hamiltonian tries to *mix* the probabilities of all possible solutions. The aim of applying such a mixer Hamiltonian is to be able to explore the whole *solution space* and avoid potential local minima.

In recent work, Hadfield [2] showed that the role of mixer Hamiltonians can be expanded when there are constraints involved in the optimization problem. Preparing according to the hard constraints of a problem, the mixer is ensured, not only to explore the whole solution space but to also restrict it according to the corresponding constraints. However, preparing a mixer can be cumbersome, since it requires distinct ansatz for different problems and constraints.

This paper proposes a different approach for solving such constrained optimization problems. Instead of using a problem-specific mixer Hamiltonian, the model uses an encoding scheme, that shrinks the search space into the feasible subspace. After applying the cost Hamiltonian and the encoder in a cascaded manner, the aim is to find an approximate solution to the optimization problem, that satisfies the constraints, using the Travelling Salesman Problem as an example.

Therefore, the overall goal is to provide an optimization scheme for a constrained optimization problem, along with comparing the convergence of the algorithm with the state-of-art approaches.

Contents

1 Introduction

Over the past few decades, there has been far-reaching research, dedicated to promising quantum algorithms, to outperform their classical counterparts. Accompanying the advances in quantum software, the progress in quantum hardware laid the foundations for displaying the advantages of quantum software in the forthcoming years. The concept of a quantum computer being able to outperform a classical computer on any application problem, such that classical devices are unable to solve the problem in a reasonable amount of time even with future improvements to classical software and hardware, is called *Quantum Supremacy*.

There are several realizations of quantum computers with different approaches, such as Gate-Model Quantum Computing [3, 4], or one of its alternatives, Adiabatic Quantum computing [5]. Quantum Annealers, which follow the adiabatic process, have been one of the early instances of physical implementations of a quantum computer, implemented by D-Wave [6]. Using the D-Wave2X Quantum Processing Unit (QPU), it was demonstrated that [7, 8] a quantum algorithm can outperform its classical analogues, in various non-trivial optimization problems. Furthermore, in 2016, IBM launched the first cloud-based quantum computer with public access, using the Gate-Model approach [9], however the limits on the number of qubits and the computation being prone to errors, have been the primary obstacles to implementing promising quantum algorithms. More recently, Google's Sycamore superconducting circuit has demonstrated quantum supremacy, on a certain sampling problem [10].

Although it is not trivial to showcase that any quantum algorithm is capable of beating the performance of its equivalent classical algorithms while maintaining a high solution accuracy, several instances are promising candidates to do so, especially in the Noisy Intermediate-Scale Quantum (NISQ) era. One of the auspicious examples of such an algorithm is the Quantum Approximate Optimization Algorithm (QAOA). QAOA is a *meta-heuristic*, *gate-model variational* quantum algorithm, that is mainly used for solving combinatorial optimization problems. One of several advantages of QAOA is that it works with a relatively small number of qubits while keeping the circuit sufficiently shallow. Furthermore, it has a very high representative capability, since it has a lot of use-case problems to tackle upon.

There are several important aspects of preparing a successful QAOA ansatz. One of the most eminent factors to consider is, to formulate a *cost Hamiltonian* and a *mixer Hamiltonian*, in accordance with the problem structure, which is further discussed in Section 2.3.1.1. Moreover, the selection of a classical optimization scheme is also another crucial aspect of learning success. Since the objective function that the algorithm tries to maximize or minimize over, could include a large number of *local minimum* and *local maximum*, the optimization method should also be selected accordingly and carefully, in order not to get stuck while trying to reach the global maxima/minima.

Constrained optimization problems, which are a subset of general combinatorial optimization problems, pose even more difficulties since their optimal solutions lie in a feasible subspace, which is a subset of the general solution space. There are classical approaches implemented for solving various instances of constrained optimization problems that also emerge frequently in real-life problems. However, for a large portion of problems, the classical methods fail to provide an exact, or an approximate solution in reasonable complexity.

Furthermore, even though there are several quantum algorithms that are dedicated to solving constrained optimization problems, they often do not propose a general-purpose ansatz that can be used for various problem structures, or that can scale efficiently as the problem size grows. In order to remedy these problems, this thesis proposes a novel QAOA ansatz interleaved with an encoder that is designated for solving combinatorial optimization problems with *hard constraints*. The encoder circuit presents a general-purpose solution by projecting the states onto the feasible subspace, which guarantees that the output signifies a solution that satisfies the hard constraints, while the optimization task then becomes to find the most optimal solution among them. Thus, the overall goal of this work is to introduce a general-purpose ansatz for solving constrained optimization problems, that incorporates a more efficient solution that can also be conveniently applied for larger problem scales.

In the remaining chapters, the discussion is organized as follows. Chapter 2, gives a theoretical background, on general concepts in quantum computing and classical ways to solve combinatorial optimization problems, along with the QAOA process, its practical uses, and how to formalize a QAOA ansatz depending on the problem structure. Chapter 3 describes the problem statement of the Travelling Salesman Problem (TSP) and proposes a novel approach for solving problems that are in the same class. Chapter 4 presents the results of the ansatz and displays the comparison between similar approaches in terms of performance and accuracy. Chapter 5 presents the conclusions, along with final remarks on the algorithm. Finally, Chapter 6, gives a general overview of the problem statement and the methods that have been used, and an analysis of how to further improve the precision and the efficiency of the method.

2 Theoretical Background

2.1 Quantum Computing

In this section, before going into detail about various solutions based on quantum algorithms, the fundamentals of quantum computing are discussed, along with the used notation in the following chapters.

A quantum wave function is a mathematical object that describes the *quantum state* of a system. It is mostly represented by the Greek Letters psi (ψ) and phi (ϕ). Furthermore, it can be used to describe different properties of a particle, depending on the context. For instance, it can be used to describe a particle's position in a position space, a polarization of a photon, an energy level of an electron, or the spin of an elementary particle, which is an intrinsic angular momentum. Moreover, The Hilbert space is a vector space, that is also equipped with an *inner product*, that describes the distance between two elements of that space. It is a generalization of the Euclidean space, such that it can also be infinite-dimensional, and it is defined over the set of complex numbers.

Whenever the quantum state of a system can only take up values that are a superposition of d independent and distinguishable states, this system is called a d-level system, that lives on a d-dimensional *Hilbert space*, $\mathcal{H}^d$. The superposition principle states that, a d-level quantum state can be in a linear combination of possible basis states in such a form,

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \cdots + \alpha_d |d\rangle , \tag{2.1}$$

that each coefficient α_i, represents the probability of that particle, collapsing into their corresponding basis state $|i\rangle$. The probability of measuring the state $|i\rangle = |\alpha_i|^2$, such that $\sum_i^d |\alpha_i|^2 = 1$, meaning that the sum of probabilities of measuring each basis state is a valid probability distribution.

Qubits are the smallest information unit, that can be stored in a quantum processor, that can be used to describe a state of a 2-level quantum system. Even though it has 2 distinct *states* that it can take after being measured, in contrast to its classical counterpart 'bits', a qubit can be in any state that is a *superposition* of 0 and 1 states, before such a measurement occurs. The more general term, *qudit*, is used to describe the states in a general d-qubit quantum system.

In the upcoming chapters, Dirac (bra-ket) notation [11] is being used to describe quantum systems and operations. A quantum state vector ψ is represented with a coloumn vector *ket*

$$|\psi\rangle ,$$

and the *bra* vector shows its adjoint, or the conjugate transpose

$$\langle\psi| .$$

U is generally used for a unitary matrix. The symbol H appears frequently, and there are several use cases, that differ according to the context. When the symbol H also includes a *hat*, $\hat{H}$ describes a *Hamiltonian operator*, which is a Hermitian matrix. If it appears in the context of a quantum gate in a circuit, it represents the *Hadamard* gate. Sometimes, it is also used to represent the *Hilbert Space*, which is demonstrated as $\mathcal{H}$

The $|0\rangle$ and $|1\rangle$ states are also known as the computational basis states, or the bases in the Z-basis. They are the eigenvalues of the Pauli-Z operator, such that,

$$\sigma_z |0\rangle = 1 \cdot |0\rangle \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{2.2}$$

The $|+\rangle$ and $|-\rangle$ states are the basis states in the X-basis, and they refer to the equal superposition of the computational basis states with alternating phases, such that,

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{2.3}$$

One useful tool, to analyze and also visualize the state of a single qubit is called *Bloch sphere*, which is shown in Figure 2.1.

One constantly mentioned operator set is called the *Pauli Gates*, which includes Pauli-X, Pauli-Y, and Pauli-Z operators, which are both Unitary and Hermitian matrices. They are commonly represented with a sigma symbol σ.

A subscript after an operator defines at which qubit line does that operator act upon. For instance, the string $Z_1Z_2X_3X_4$ describes a system with 4 qubits, where Z gates act on the first 2 qubits, and X gates act on the other 2. The string of different operators represents the *Kronecker Product* of those operators. Similarly, quantum state vectors that are demonstrated as a string, also illustrate a Kronecker product. For instance the expressions

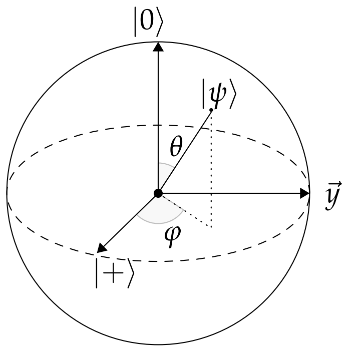


Figure 2.1: Bloch Sphere and representation of a single state.

$$|01\rangle = |0\rangle\,|1\rangle = |0\rangle \otimes |1\rangle\,,$$

all represent the same quantum state. The inner product of two quantum states is being shown as $\langle v|w\rangle$, and the outer product is $|w\rangle\,\langle v|$.

Quantum computing, is essentially built upon making calculations regarding the changes occurring in a quantum state [3]. There are different realizations of a quantum computer, such as Gate-Model Quantum Computation, Adiabatic Quantum Computation, and Topological Quantum Computation [12]. Being one of the prevailing models, Gate-Model Quantum Computing utilizes *Quantum Logic Gates* to manipulate and make computations on quantum systems.

One of the strengths of the gate-model quantum computation is its ability to provide a framework for simulating physical systems. The main objective of this task is to find a solution to the question of how the final state of a system will evolve over time given initial conditions. Furthermore, simulating quantum systems with a classical computer is rather a non-trivial and inefficient task [3]. The key obstacle for a classical computer while simulating a quantum system, is that the complexity of a quantum system grows exponentially.

The dynamics of a quantum system over time is governed by the Schrödinger's Equation,

$$\frac{\partial}{\partial t}\psi(x) = \Big[-\frac{1}{2m}\frac{\partial^2}{\partial x^2} + V(x)\Big]\psi(x). \tag{2.4}$$

The above equation utilizes continuous quantum wave function $\psi(x)$ to represent a position of a particle. In the context of quantum states that have a discrete number of possible states (qudits), the same equation is being used as

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle, \tag{2.5}$$

where $\hat{H}$ is the *Hamiltonian* operator. Hamiltonian is a Hermitian operator, and operated on a wave function, it describes the total energy of that particular quantum system. At the same time, it governs the time evolution of a quantum wave function. The time evolution is described by *unitary maps*, that have the form,

$$|\psi(t)\rangle = e^{-i\hat{H}t/\hbar}|\psi(0)\rangle = U(t)|\psi(0)\rangle. \tag{2.6}$$

Therefore, a unitary map (similarly, unitary gate, unitary transformation), which is a bounded linear transformation on Hilbert space, is essential to explain how a quantum state evolves in time. Therefore, the quantum gates in the gate-model computation are implemented as unitary transformations. A quantum circuit is reversible since an inverse of a unitary always exists and it is equal to its adjoint operator $U^\dagger$.

Formulated first by Feynman [13], figuring out how Hamiltonian acts on a system, is called the *Hamiltonian Simulation* problem, and it is an essential problem in various domains in quantum physics and quantum chemistry. Since a Hamiltonian is a $2^n \times 2^n$ Hermitian matrix, exponentiating that matrix is exponentially costly with respect to the number of qubits in the system. Therefore, as the system size grows, finding the exact time evolution becomes an intractable task. However, there are different sets of Hamiltonians, in which calculating the exact time evolution is less costly. One example of that is, Hamiltonians that are diagonal in some basis $|u\rangle$, that can be generalized as

$$\hat{H} = \sum_i \alpha_i |i\rangle\langle i|. \tag{2.7}$$

The matrix exponential of a matrix is generally in the form

$$e^H = \sum_{k=0}^{\infty}\frac{1}{k!}H^k. \tag{2.8}$$

However, the exponential is trivial to compute while working with the diagonal matrices, such that,

$$\hat{H} = \sum_i \alpha_i |i\rangle\langle i| \tag{2.9}$$

$$e^H = \sum_i e^{\alpha_i}|i\rangle\langle i|. \tag{2.10}$$

Similarly, if A is a diagonalizable matrix, then the exponential is

$$A = UDU^{-1}$$
$$e^{A} = Ue^{D}U^{-1}.$$

Furthermore, the *commutation* relation is an important property that also utilizes diagonalization. *The Commutator* between two operators A and B is defined as,

$$[A, B] = -[B, A] = AB - BA. \tag{2.11}$$

If $[A, B] = 0$, then $AB = BA$, and it can be said that the matrices A and B commute. In the context of unitary transformations, two operators can be applied to a system interchangeably (either applying A first, then B, or vice versa), if they commute, and the resulting quantum state will be the same. Another significant subset of Hamiltonians is k-local Hamiltonians, where there are a set of different operators H_j that compose the overall Hamiltonian H, such that each component acts non-trivially only on at most k-qubits of the system, where $k < n$. Therefore, the summation of those components is equal to

$$\hat{H} = \sum_{j=1}^{m} H_j. \tag{2.12}$$

Such Hamiltonians that have the form of H_j are called *local* Hamiltonians while finding its simulation is called the *Local Hamiltonian Simulation*.

If each pair of individual operators H_j and H_i that act on k-qubits, commute among themselves, then the simulation becomes easier, due to the fact that the exponential of each local term can also be computed separately. Equation 2.13 demonstrates such a calculation, and it only holds for the cases where each pair of the local terms in the summation commute.

$$e^{-i\hat{H}t} = e^{-i(\sum_{j=1}^{m} H_j)t} = e^{-iH_1 t}e^{-iH_2 t} \dots e^{-iH_m t} \tag{2.13}$$

However, it is not always the case, that the terms in the summation commute independently. One simple example can be given as $H = \sigma_x + \sigma_z$, since σ_x and σ_z anti-commute In these settings, since it is intractable to find the exact unitary transform that corresponds to a Hamiltonian and a time parameter t, various approximation algorithms can be employed, in order to find a unitary map U_a, such that $||U - e^{-iHt}| < \epsilon|$, where e^{-iHt} describes the exact time evolution, $||.||$ is the spectral norm and ϵ is a hyper-parameter that describes the maximum simulation error. There are several algorithms to tackle this problem. 3 of these algorithms and their gate and query complexities are listed in Table 2.1.

Algorithm	Gate Complexity	Query Complexity
Taylor Series	$O\left(\frac{tlog^2(\frac{t}{\epsilon})}{loglog\frac{t}{\epsilon}}\right)$	$\mathrm{O}\left(\frac{d^2\|\|H\|\|_{max}log\frac{d^2\|\|H\|\|max}{\epsilon}}{loglog\frac{d^2\|\|H\|\|max}{\epsilon}}\right)$
Trotter-Suzuki	$O\left(\frac{t^2}{\sqrt{\epsilon}}\right)$	$O\left(d^3t(\frac{dt}{\epsilon})^{\frac{1}{2}}k\right)$
Quantum Random Walk	$O\left(\frac{t}{\sqrt{\epsilon}}\right)$	$O\left(d^2\|\|H\|\|_{max}\frac{t}{\sqrt{\epsilon}}\right)$

Table 2.1: Gate and Query complexities of different Hamiltonian Simulation methods [14, 15]

One approximate algorithm for Hamiltonian Simulation is the Trotter-Suzuki Decomposition (also known as Trotterization or Product Formulas). Trotter-Suzuki Decomposition is the process of simulating each term that appears in the sum-of-terms of the Hamiltonian solely for a small portion of time [16]. Suppose that a Hamiltonian has the form $H = \sum_i H_i$, then the exact unitary transformation is equal to $U(t) = e^{-i(\sum_i H_i)t}$. The approximation is done as

$$U(t) \approx \left(e^{-iH_1t/r} \dots \ e^{-iH_nt/r}\right)^r, \tag{2.14}$$

where r describes the number of time steps of the simulation.

2.2 Combinatorial Optimization Problems

Combinatorial optimization problems, that arise in many fields such as computer vision, communications network design, database queries, AI reasoning, or computational biology [17], is a sub-field of mathematical optimization. Many different research areas in operations research, computer sciences, and mathematics have been working on the theory and better performing algorithms for solving combinatorial optimization problems. One possible way to tackle such problems is called *Linear Programming (LP)*, and they perform in polynomial time for a small subset of combinatorial optimization problems, called LP problems. However, this set does not generalize over other combinatorial optimization problems, which can also be NP-Hard or NP-Complete. For a general NP-Complete problem, there is also a reasonable amount of research, dedicated to finding *approximate solutions* in polynomial time, such that, the final output is relatively close to the global optimum. Another obstacle in solving combinatorial optimization problems is, that the problem setting is almost always non-convex therefore there might be local minima, where the optimization algorithm might get stuck, instead of constantly improving [18, 19].

By definition, combinatorial optimization problems, are specific types of optimization problems where the goal is to find the optimal solution that is in a finite set, where the optimality of a solution is decided by an objective function. Then, the optimization over that finite is done via either maximizing, or minimizing that objective function, which is also commonly known as the *cost function*, or the *loss function*. The binary combinatorial optimization problems are a subset of combinatorial optimization problems, where each discrete variable can take up to 2 distinct values. Even a deeper subset, called the Quadratic Unconstrained Binary Optimization (QUBO), which is an NP-hard problem, has been thoroughly investigated in the quantum computing field [20], due to its natural close connection with the Ising model, which can be seen in Equation 2.15, and it's suitableness to be solved through Quantum Annealars (QA) using Adiabatic Quantum Computing (AQC) approach [21].

$$H(\sigma) = -\sum_{\langle i,j\rangle} J_{i,j}\sigma_i\sigma_j - \mu\sum_i h_i\sigma_i \tag{2.15}$$

Moreover, various well-known problems in computational theory, such as Max-Cut, Graph Coloring, Graph Partitioning, or the Travelling Salesman Problem(TSP) have been formulated as QUBO problems [22].

As a general setting of a combinatorial optimization problem, the solution space is the set of n-bit binary strings, the cost function maps n-bit binary strings into a real number and the structure of the optimization routine is given as

$$\begin{aligned} &min.\ C(\boldsymbol{x}) \\ &s.t.\ \boldsymbol{x} \in S, \end{aligned}$$

where S refers to the solution space, or the *domain* of the problem, $\boldsymbol{x}$ is a random discrete variable, $C : F \rightarrow \mathbb{R}$ is the cost function, and $F \subseteq S$.denotes the *feasible subspace*, especially defined for the *combinatorial optimization problems with constraints*. Sometimes, it is also useful to think of every violation of cost function, as a violation of *soft constraints*, which does not necessarily have to be fulfilled, but adds a penalty term whenever they are not satisfied by an example solution. Hence, the *hard constraints*, refers to the clauses of the problem, which *must* be satisfied.

General cost functions of combinatorial optimization problems have the form of

$$C(\boldsymbol{x}) = \sum_{(N,\tilde{N})} h_{(N,\tilde{N})} \prod_{i\in N} \boldsymbol{x}_i \prod_{j\in\tilde{N}} (1-\boldsymbol{x}_j)\ , \tag{2.16}$$

where N and $\tilde{N}$ are subsets of the solution space. An exemplary cost function which is formulated as,

$$C(\boldsymbol{x}) = \sum_{\langle i,j\rangle\in S'} h_{i,j}\ \boldsymbol{x}_i, (1-\boldsymbol{x}_j) = \sum_{\langle i,j\rangle\in S'} -h_{i,j}\ (\boldsymbol{x}_i, \boldsymbol{x}_j)\ , \tag{2.17}$$

can be used to describe the problem, where two neighbouring members of the same set S' are tried to be assigned differently. For instance, the cost function in Equation 2.17, can be utilized to formulate the Maximum-Cut problem, where the aim is to find two cuts in a connected graph, such that the number of edges between two cuts are maximized. This formulation, has a very close connection with a simplified version of the Classical Ising Model that is shown in Equation 2.15. The representative difference between those two equations, is that in the objective function, the discrete random variables refer to an assignment value, whereas in the Ising Model, they refer to a *spin* value along the z-axis.

Following its similarity to the Classical Ising Model, a cost function, that is derived from the general form in Equation 2.17, can also be mapped to a *Hamiltonian*, that encapsulates all the soft constraints as different interactions occurring in the system. That Hamiltonian can be used as a baseline for the learning criteria to be used in the Quantum Algorithm later on, which is discussed in detail in the following Section 2.3. Moreover, several Hamiltonian formulations for different cost functions are also discussed in Section 2.3.1.1. Furthermore, in this mapping, the *spin-z values* that show up in the Hamiltonian formulation as a discrete random variable, can also be thought of as assignments of eigenvalues of Pauli Z. The Pauli-Z operator is a diagonal operator, that has the eigenvalues -1 and 1. Since the Pauli-Z is a diagonal matrix, any tensor product of Pauli-Z gates will also be diagonal in the computational basis states. Therefore, finally, the cost function can be converted as a Hamiltonian, which is diagonal in the computational basis states.

2.3 Solving Combinatorial Optimization Problems with Quantum Computing

There are several ways to implement a solution for combinatorial optimization problems in the setting of quantum computing. One of the most common approaches is to formulate the objective function of the problem as a *Hamiltonian operator* and try to find the *highest energy eigenstate* of the Hamiltonian, which is going to encode the configuration that is an approximate solution for the optimization problem. Therefore, even though every energy eigenstate of the Hamiltonian is a valid solution to the optimization problem, finding the highest energy eigenstate is going to be the aim of the quantum algorithm.

As explained in Section 2.1, Adiabatic Quantum Computation is one of the methods for doing quantum computations, and it has a very close relation with the QAOA. Suppose that a Hamiltonian H_0 is given that is acting on n number of qubits, and the quantum system $|\psi_0\rangle$ that consists of those n qubits, is a *ground state* of the Hamiltonian H_0. Then, the Hamiltonian is gradually transformed into some other Hamiltonian H_f, whose ground state is $|\psi_f\rangle$. The transform can be given as a convex combination of H_0 and H_f, such as:

$$H_t = (1-t)H_0 + tH_f \tag{2.18}$$

where H_t is a time-dependent Hamiltonian that mixes two Hamiltonians H_0 and H_f. *The Adiabatic Theorem*, states that doing the transformation slow enough achieves successful tracking of the instantaneous ground state. Therefore, starting from a known ground state of the Hamiltonian H_0, by doing a transformation that is sufficiently slow, the system can end up in the ground state of H_f. Therefore, there is a close relation between the Adiabatic Quantum Computing and QAOA, since while trying to find the optimal solution for a combinatorial optimization problem, the goal is to find the ground state of the Hamiltonian, which encapsulates the problem structure. However, there is a limit T on how "slow" the transformation should be:

$$T = \frac{1}{min_t g(t)^2} \tag{2.19}$$

where the function g gives the *gap* between the two lowest energy eigenvalues of the Hamiltonian H_f. Therefore, if the energy gap between the two lowest eigenstates is too small, then the speed limit becomes too restrictive and it prohibits achieving an exponential speedup while trying to find the desired solution, even for trivial problems [23, 24]. Since in most combinatorial optimization problems, the costs between feasible solutions might be too small, AQC is not always the ideal way to solve these kinds of problems. In order to overcome the constraint on the speed limit, Quantum Annealers (QA) has been used to tackle these problems [25, 26], however, it has also drawbacks, since the optimization can get stuck on a local-minima while searching for a solution. Also, similar to the principles in Adiabatic Computation, the problem Hamiltonian has to be a *gapped Hamiltonian*. The term *gapped Hamiltonian*, can be summarized as a Hamiltonian in an infinitely large Hilbert Space, that has a finite and non-zero energy gap between its ground state and its first excited state.

The Quantum Approximate Optimization Algorithm (QAOA), proposed by Farhi [1], is a meta-heuristic gate-model quantum algorithm. It is a well-studied algorithm, mainly used for solving such combinatorial optimization problems. It is also a *variational* method, meaning that the goal of the algorithm is to find the set of parameters of certain gates in the circuit, that either maximize or minimize the objective function.

One advantage of QAOA is, that they have a very shallow architecture with a trivial form, mainly consisting of a switching mechanism between the cost function based operators and the *mixer* operators. Since it requires only a low number of qubits, and unitaries that most of them have similar forms, QAOA is thought of as one of the best algorithms for noisy intermediate-scale quantum (NISQ) computation, which can also prove the *quantum supremacy* over its classical equivalent counterparts. Another benefit of using QAOA is that since it is a gate-model-based approach, contrary to the AQO, it can also work with *gapless Hamiltonians*.

2.3.1 The Quantum Approximate Optimization Algorithm (QAOA)

There are many interesting real-world problems that can be formulated as combinatorial optimization problems. Given a classical objective function

$$C(z) : \{+1, -1\}^N \rightarrow R_{\geq 0} \tag{2.20}$$

of a problem with n binary variables. Examples for such a binary variable can be given as the assignment of a binary value for each variable that appears in a clause, for the 3-SAT problem.

The full set of n-bit strings denote the *search space*. Furthermore, any n-bit string assignment, that also satisfies all the hard constraints of a problem, can be considered as the *feasible solution*, while the set that only includes such solutions, is called the *feasible subspace* of the *solution space* f n-bit strings that also satisfy all the hard constraints are called the *feasible space*.

As opposed to the *exact optimization problem*, where the goal is to find the solution that either maximizes or minimizes the objective function, in an *r-approximate optimization* problem, the goal is to find the solution, within a certain approximation ratio r. Therefore, even though getting the most optimal solution is not certain, it is guaranteed to find the solution which at least is a factor of r of the most optimal assignment, such that:

$$\frac{C(z)}{C_{max}} \geq r. \tag{2.21}$$

The algorithm can be classified as an r-optimization problem, if any instance of the problem (same objective function definition with alternating values) yields a solution that is at least a factor of r of the optimal solution. Additionally, following the structure of the Adiabatic Theorem, it is proven that the QAOA is able to capture the most optimal solution with the correct setting [1], which is going to be explained later in the Section 2.3.1.1.

Therefore, QAOA can be thought of as a Gate-Model implementation for approximating the *adiabatic pathway*. The way to achieve that is by preparing two Hamiltonians, called the *mixer Hamiltonian* and the *cost Hamiltonian* (Also called the problem Hamiltonian). Starting in the ground state of the mixer Hamiltonian, which has a very simple form so that the ground state of it is easy to prepare, the aim is to gradually acquire the ground state of the cost Hamiltonian by following an approximation of the adiabatic pathway. Similar to AQC, in QAOA, the transition happens gradually, by *Trotterization* (see Section 2.1). Having the same transition with Equation 2.18, the Trotterization breaks the time evolution of the Hamiltonian into discrete chunks, with parameters β and γ, and by alternating application of the mixer and the cost Hamiltonians with p number of different iterations, the approximation for the ground state of the cost Hamiltonian is finally achieved. In contrast to the QA approach, this method also includes a classical optimization, for finding the parameters β and γ in the unitaries. The Trotterization process in QAOA is given in Equation 2.22

$$\left|\psi_{\vec{\beta},\vec{\gamma}}\right\rangle = e^{-i\gamma_p H_M}\ e^{-i\beta_p H_C}\ e^{-i\gamma_1 H_M}\ e^{-i\beta_1 H_C}\ |\psi_{H_M}\rangle \tag{2.22}$$

where $|\psi_{H_M}\rangle$ stands for the ground state of the mixer Hamiltonian.

2.3.1.1 Preparing Cost and Mixer Hamiltonians

The very first step for a QAOA is to convert the classical problem structure into the Quantum setting. Therefore, it is required to formalize the objective function, as a Hamiltonian operator, which contains a ground state that is an optimal solution for the problem. In the state vector representation of that ground state wave function, every qubit corresponds to a binary variable.

Consider an objective function that has the form $C(x_1, x_2 \dots x_n)$. The cost Hamiltonian can be constructed by substituting each binary variable with a quantum spin σ_i^z. The Pauli-Z operator σ^z has an eigenvector $|0\rangle$ with the eigenvalue 1, and an eigenvector $|1\rangle$ with the eigenvalue -1. Therefore, since such a *local* Hamiltonian H_C contains only local Hamiltonians that include a Pauli-Z that individually commute among themselves the H_C is considered to be a *classical* Hamiltonian. Moreover, the eigenspace of H_C consists of *computational basis states*. Hence, the eigenvectors $|x\rangle$ of H_C correspond to a solution, with the "energy", or the cost term is given as the eigenvalue of that eigenvector, which makes every computational basis state a solution.

The design of the cost Hamiltonian varies as the problem structure changes. There are several propositions [27, 28] for the mappings from varying Boolean functions into cost Hamiltonians, which is also shown in Table 2.2.

Having a cost Hamiltonian, dedicated to encapsulating the objective function, results in separation of phases, according to the cost attached to the specific basis states. For instance, suppose a state vector that is initially in an equal superposition. After being

$f(x)$	H_C
x	$\frac{1}{2}I - Z$
$\tilde{x}$	$\frac{1}{2}I + Z$
$\wedge_{j=1}^{n} x_j$	$\prod_{j=1}^{n} \frac{1}{2}(I - Z_j)$
$\vee_{j=1}^{n} x_j$	$I - \prod_{j=1}^{n} \frac{1}{2}(I + Z_j)$

Table 2.2: Exemplary mappings from Boolean clauses into cost Hamiltonians [27] [28]

applied to the cost function based Hamiltonian, the states that correspond to a solution with a high cost get bigger phases and therefore have a bigger probability of being measured, with respect to the cost Hamiltonian, since the probability of measuring the state is attached with its coefficient $|a_x|^2$.

QAOA also makes use of the *mixer* Hamiltonians while solving combinatorial optimization problems. The mixer Hamiltonian resembles the *driver Hamiltonian* in the Quantum Annealing setting, in which the system is initialized as its ground state, while gradually trying to acquire the ground state of a problem Hamiltonian. There, driver Hamiltonian is specifically selected such that it does not commute with the problem Hamiltonian. Similarly, the mixer Hamiltonian in QAOA is also selected as a Hermitian matrix which does not commute with the cost Hamiltonian, therefore it does not converse the energies of H_C. The mixer Hamiltonian is also required because simply applying HC can lead the algorithm to become stuck in the same state throughout the algorithm run-time. That is because, after acquiring a state which is an eigenstate of H_C, such that, $H_C |\psi(\theta)\rangle = E |\psi(\theta)\rangle$, the system will remain in the same state $|\psi(\theta)\rangle$, after applying the time evolution $e^{-i\alpha H_C}$. The overall goal of the mixer Hamiltonian can be summarized as amplitude mixing for the computational basis states.

As it is explained later in Section 2.3.1.2, the unitaries, which happen to be the time evolutions of H_C and H_M with a different set of parameters, achieve phase separation interleaved with amplitude mixing, that eventually leads to a solution.

2.3.1.2 Defining the Unitary Operators

As it was explained in Section 2.1, the Hamiltonian is the operator that yields the total energy that the system has. Moreover, the application of a Hamiltonian on a quantum system with a certain period of time is described by a *unitary*, which is given from the time evolution of the Hamiltonian. The equation that describes the time evolution of a quantum system over a specified time t, is called the *The time-dependent Schrödinger Equation*. The Time-dependent Schrödinger Equation is given as

$$i\hbar\frac{d}{dt}\left|\psi(t)\right\rangle = \hat{H}\left|\psi(t)\right\rangle \tag{2.23}$$

Utilizing the Schrödinger's Equation, the time dependent unitary transformations (or, similarly the time evolution operators) have the form of,

$$U(t) = e^{-i\hat{H}t}. \tag{2.24}$$

such that,

$$\left|\psi(t)\right\rangle = U(t)\left|\psi(0)\right\rangle = e^{-i\hat{H}t}\left|\psi(0)\right\rangle \tag{2.25}$$

As discussed in Section 2.3.1, the overall goal of the algorithm is to find the maximum energy eigenvector of the cost Hamiltonian using a classical optimization routine, which can be formulated as

$$\max_{\beta,\gamma}\left\langle \psi_{\vec{\beta},\vec{\gamma}}\middle| H_C \middle|\psi_{\vec{\beta},\vec{\gamma}}\right\rangle . \tag{2.26}$$

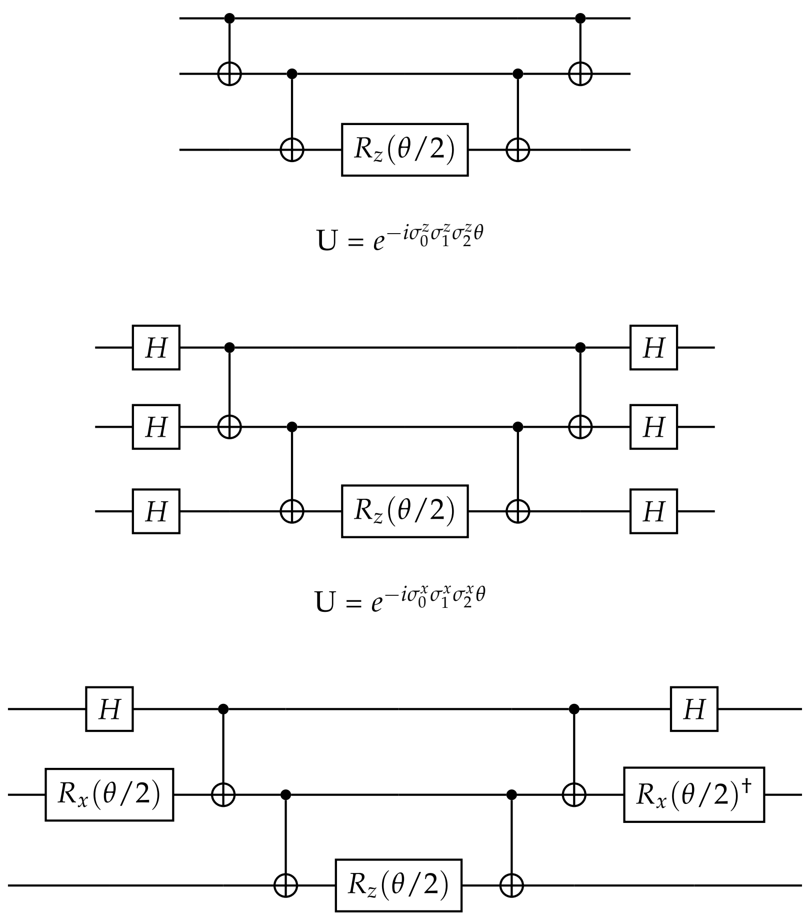
.

Making a close connection to the Adiabatic Method, in QAOA, the final ground state is achieved by applying the cost Hamiltonian, in an alternating fashion with the mixer Hamiltonian several times in a cascading manner. Even though the number of alternating iterations increases the depth of the circuit, it is also shown to be increasing the approximation accuracy [1]. The number of iterations parameter is p, is an *hyper-parameter*, which can also be optimized depending on the problem structure. Therefore the applications of the Hamiltonians in a fixed period of time can be represented as unitary transformations, such as

$$H_C \xrightarrow{\beta} e^{-iH_C\beta} \;\; , \;\; H_M \xrightarrow{\gamma} e^{-iH_M\gamma}. \tag{2.27}$$

Thus, the backbone of the QAOA ansatz is the alternating application of parameterized unitary transformations $U_M^p(\gamma)U_C^p(\beta)$, while p represents the index of the unitaries.

Creating the unitary transformations via time evolution of the mixer or the cost Hamiltonians are rather trivial, since the Hamiltonians are either diagonal in the computational basis, or the individual terms in the Hamiltonian's commute among themselves. Therefore, it is possible to create the unitary from local gates. For instance, if the Hamiltonian has the form $H_{C_a} = Z_1Z_2$, then the unitary transform becomes $e^{-iH_{C_a}\beta} = e^{-iZ_1Z_2\beta}$ which equals to the Ising ZZ Coupling Gate, parameterized with β. Different examples of the Hamiltonian simulation are illustrated in Figure 2.2.



$$U = e^{-i\sigma_0^z\sigma_1^z\sigma_2^z\theta}$$

$$U = e^{-i\sigma_0^x\sigma_1^x\sigma_2^x\theta}$$

$$U = e^{-i\sigma_0^x\sigma_1^y\sigma_2^z\theta}$$

Figure 2.2: Circuit representation of the exponential of the Hamiltonians in a 3 qubit setting

In this figure, the Hamiltonians are expressed as *Pauli Strings*, or similarly as linear combinations of tensor products of Pauli matrices. The final form of the unitaries is called the *Ising Coupling Gates*, which can be decomposed further into CNOTs, Hadamard gates, and single qubit rotation gates. The circuits on the figure display a 3 qubit version, which can also be generalized to n qubits. Whenever the unitary involves a σ_x or σ_y, the application of Hadamard gate or the X Rotation gate is crucial, which serves the purpose of a basis change. After the parity is calculated on the last

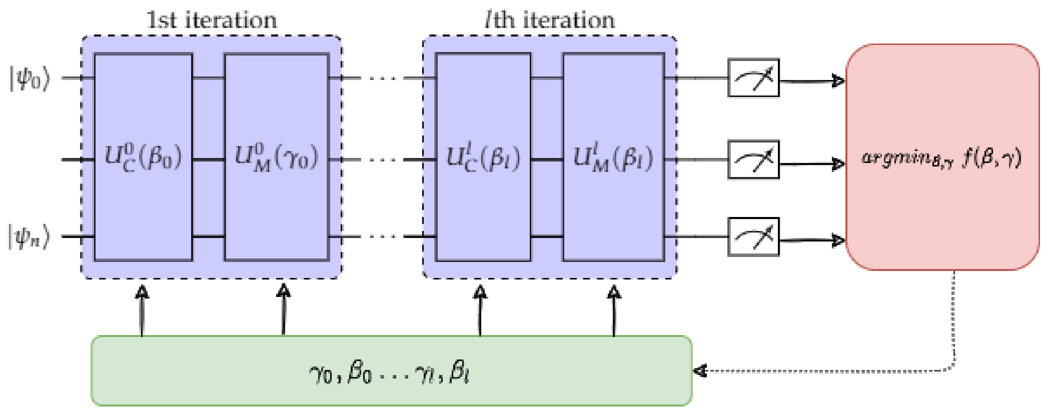


Figure 2.3: QAOA Ansatz with l iterations, equipped with a classical optimization

qubit via the application of a series of CNOTs, a single-qubit phase rotation is applied, depending on the time variable of the Hamiltonian. Finally, the reverse operation of parity calculation and the change of basis is applied.

Finally, commonly referred to as QAOA energy,

$$E_p(\overrightarrow{\beta}, \overrightarrow{\gamma}) = \left\langle \psi_{\overrightarrow{\beta},\overrightarrow{\gamma}} \middle| H_C \middle| \psi_{\overrightarrow{\beta},\overrightarrow{\gamma}} \right\rangle$$

can be acquired at the end of circuit iteration either via measuring repeatedly and sampling from them, or classically [29].

To sum up, the QAOA pipeline, which is also visualized at the Figure 2.3 can be summarized as follows:

1. Preparing the initial state $|\psi\rangle$
2. Initializing $2p$ parameters for β and γ
3. Preparing the cost Hamiltonian and the mixer Hamiltonian depending on the problem structure
4. Finding the time evolutions of the cost and the mixer Hamiltonians $U_C^p(\beta)U_M^p(\gamma)$
5. Run the circuit, which has the form of $U_M^p(\beta)U_C^p(\gamma)\dots U_M^1(\beta)U_C^1(\gamma)$
6. Sample S number of final states from the circuit or calculate the expectation value $\left\langle \psi_{\overrightarrow{\beta},\overrightarrow{\gamma}} \middle| H_C \middle| \psi_{\overrightarrow{\beta},\overrightarrow{\gamma}} \right\rangle$

7. If sampling is used, then provide a classical loss function, depending on the problem structure
8. Optimize over the parameters β and γ with a suitable method
9. Iterate until convergence

Furthermore, it is also proven that [1] while $p \to \infty$, the QAOA gives the most optimal result as a solution, instead of an approximation.

2.3.2 Specific QAOA Models for Constraint Problems

Along with the default QAOA ansatz, there are also other methods dedicated to solving different types of problems with an alternating optimization scheme. In the default setting of the QAOA, as it is discussed in the Sections 2.3.1.1 and 2.3.1.2, the role of the cost Hamiltonian is to separate the phases of the computational basis states such that

$$H_C \left|\boldsymbol{x}\right\rangle = f(x) \left|\boldsymbol{x}\right\rangle, \tag{2.28}$$

where $\boldsymbol{x}$ is a computational basis state and f is the cost function. Therefore, the family of phase separation unitaries have the form of

$$\sum_{\boldsymbol{x}} e^{-if(x)\beta} \left|\boldsymbol{x}\right\rangle \left\langle\boldsymbol{x}\right|. \tag{2.29}$$

On the contrary, mixer Hamiltonians have the role of mixing the probabilities of measuring each computational basis state, which helps to explore the whole solution space.

However, with that general approach, it is not straightforward, how the QAOA ansatz can find a solution for a combinatorial optimization problem with hard constraints. As it was discussed in the Section 2.2, while soft constraints define how optimal is the setting for some solution s, hard constraints decide the *validity* of a solution, which means if some constraints are violated in a solution s, that means it is not a feasible solution, therefore it must be discarded.

Two similar classical approaches for solving optimization problems with constraints are called the Augmented Lagrangian Method [30–32] and the Penalty Method [33, 34]. In both of these methods, the goal is to divide the problem into multiple unconstrained sub-problems, while the constraints are added as a penalty term λ in the original objective function. Therefore, these methods can be considered as a relaxation of a constraint problem, where the solution is only an approximation to the global optimum. In the case of $\lambda = 0$, the solution would disregard the constraints, and for the case $\lambda = \infty$, the solution will be determined, solely based on constraints and not the objective function itself. Therefore, the choice of a hyper-parameter penalty term is crucial in these types of methods.

Therefore, similar to its classical counterpart, in quantum optimization, one approach for solving constrained optimization problems is to specify a penalty term in the objective function, corresponding to each constraint. In such approaches [35, 36], the cost Hamiltonian is reformulated, such that it also encapsulates the constraints with a penalty coefficient. The new cost Hamiltonian then is defined as

$$H_C^{pen} = H_C + \sum_i \lambda_i \hat{C}_i, \tag{2.30}$$

where $\hat{C}$ is defined as the constraint operator.

However, penalizing the cost Hamiltonian is not a very efficient technique for solving constrained optimization problems [37, 38], due to the fact that the penalty terms complicate the cost Hamiltonian by adding multi-level interactions, the realization of such methods on physical hardware becomes cumbersome [39].

In order to address these issues, Hen, Spedalieri and Sarandy [37, 38] proposed an approach in the context of Quantum Annealing, by embodying the hard constraints into the mixer Hamiltonian, instead of having a trivial mixer that has the same structure across different problems. The main justification for that is, that by using general purpose mixers, the search occurs over the full solution space rather than a restricted feasible space. On the contrary, in this method, mixers have the responsibility for both restricting the search space to a feasible subspace, and also providing a mixing mechanism for exploring that feasible space, starting from any state which is in the feasible subspace.

In that work, it is proposed that a Hamiltonian H_A that embodies the hard constraints can be prepared that has the ground subspace that spans the feasible subspace. Similar to the default approach, H_A must not commute with the cost Hamiltonian H_C such that, $[H_A, H_C] \neq 0$ and must commute with the mixer Hamiltonian H_M, therefore $[H_A, H_M] = 0$. In addition to that, it should also be ensured that H_M is generalized enough to explore the entire feasible subspace.

More recently, Hadfield, Wang, and Rieffel [2] explored this approach even beyond the Quantum Annealing context. Moreover, the authors also further investigate the application of the method on several different combinatorial optimization problems with hard constraints, such as Graph Coloring Problem, Travelling Salesman Problem (TSP), and Single Machine Scheduling Problem, where the first one is NP-Complete and the last two are NP-Hard problems.

One exemplary problem that they have proposed a solution for, is the Travelling Salesman Problem, with the constraints,

$$\sum_{j=1}^{n} x_{vj} = 1 \quad \forall v \in V, \tag{2.31}$$

$$\sum_{v \in V} x_{vj} = 1 \quad \forall j \in E', \tag{2.32}$$

stating that each city must be visited once, and at each step of the Hamiltonian cycle, only one city must be visited. The proposed mixer Hamiltonians have the form

$$B_{uv} = \prod_{i=1}^{n} (X_{uj} X_{vj} + Y_{uj} Y_{vj}), \tag{2.33}$$

$$H_M = \sum_{u,v \in E} B_{uv} \tag{2.34}$$

where u and v are different vertices that share an edge. The proposed mixer structure is also known as the XY-Model on a ring, which is a specified version of the Heisenberg spin model. H_M, and each B_{uv} term that constructs the H_M, commutes with a general cost Hamiltonian which is diagonal in the computational basis. The proposed mixer structure assures, that the state remains in the feasible subspace while creating a mechanism to explore that fully.

The problem formulation for the Travelling Salesman Problem is explained in more detail in Section 3.1.1, along with the proposal of the novel ansatz for solving combinatorial optimization problems, without having to formulate problem-specific mixer Hamiltonians.

In the following paper, Hadfield et.al. [40], also propose different design criteria for preparing mixer Hamiltonians, and naturally, mixer Unitaries for a variety of constraint optimization problems.

Even though, this approach necessitates providing *ad-hoc* mixer Hamiltonians according to the problem structure which can be non-trivial for various examples, it generates a framework for solving the optimization problems with varying hard constraints.

Another approach [41] designed specifically for solving the TSP problem, uses phase estimation and quantum search together in order to find a global optimum. In that work, the edge weights between two vertices are encoded as phases, such that the eigenstates of the unitaries in the ansatz provides various phase configurations that correspond to a route. Their experiments involve 4-vertex connected graphs with alternating edge weights.

2.4 Quantum Variational Algorithms

Quantum Variational Algorithms (Methods) is another branch of quantum computing, that can also be classified as a subset of Quantum Machine Learning (QML), in which the researchers are trying to achieve quantum advantage on NISQ devices. In the setting of gate-model quantum computation, variational methods are defined as, learning the parameters θ that act on some set of gates in the circuit, such that, the objective function f is minimized. The ansatz includes gates that can be parameterized or not.

The optimization cycle begins with the initialization of the quantum state, before doing any manipulation on the system. The initial state can be a trivial state that is easy to prepare, such as a computational basis state. Furthermore, the initial state can also be formed, such that it encodes the information of the training inputs. If the input features of the learning task are classical, then the encoding operation is called *data embedding* or *quantum feature map* [42]. There are different embedding types [43], for various data structures, namely linear embedding, phase embedding, or amplitude embedding. Their primary aim is to do a mapping from the classical domain into a quantum state, such that

$$\chi \rightarrow \mathcal{H} \tag{2.35}$$

$$x \rightarrow |\psi\rangle \tag{2.36}$$

where $x \in \chi$ is an n-dimensional vector in the feature space [44]. This approach is very similar to the kernel method since the mapping is done from a classical vector space into a higher dimensional Hilbert space [45].

Following the initial state preparation, the second step is to arrange an ansatz, that is general and broad enough to capture the data correlation one wishes to obtain. Since a $2^n \times 2^n$ unitary is able to transform any arbitrary state in $\mathbb{C}$ into another arbitrary state in the same space, a large enough unitary is able to capture the desired mapping. However, that approach necessitates exponentially many parameters, which makes

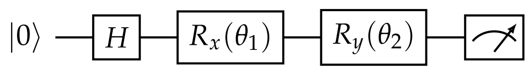


Figure 2.4: Exemplary variational circuit with 2 parameters. As shown in the figure, it is also possible to only have part of the gates on the circuit that are parameterized, since the H gate does not have any parameters.

the learning scheme intractable. Moreover, realizing an arbitrary unitary operation on quantum hardware is a cumbersome task [46].

Nevertheless, it is also possible to represent any quantum operation, by only using gates from a small set, called the *universal set of gates* [3]. Thus, instead of having an exponentially large unitary, the variational ansatz can only include universal gates, that can also be parameterized, such as a phase rotation gate. Furthermore, instead of parameterizing each element in the unitary matrix, a certain class of gates can also be parameterized with a significantly low amount of variables. For instance, a controlled rotation gate can be parameterized only with 1 variable even though the gate has 16 elements in total, since,

$$R_x(\theta) = e^{-i\theta\sigma_x/2} = \begin{bmatrix} \cos\theta/2 & -i\sin\theta/2 \\ -i\sin\theta/2 & \cos\theta/2 \end{bmatrix}. \tag{2.37}$$

Moreover, it is also possible to decompose an arbitrary unitary into a set of universal gates [47].

A very trivial variational circuit is shown in Figure 2.4.

After the ansatz is created, the final step is to prepare a cost function, mapping the measurement outcomes into a real number. Subsequently, a classical optimization routine is being used to find the set of parameters that either minimize or maximize the objective function, such that

$$\theta^* = argmin_\theta \; C(\boldsymbol{\theta}, \boldsymbol{x}). \tag{2.38}$$

The general structure of the variational methods is summarized in Figure 2.5.

2.5 Optimization

Optimization lies at the heart of quantum variational methods, and it constitutes an essential design choice while generating hybrid quantum-classical algorithms. Further-

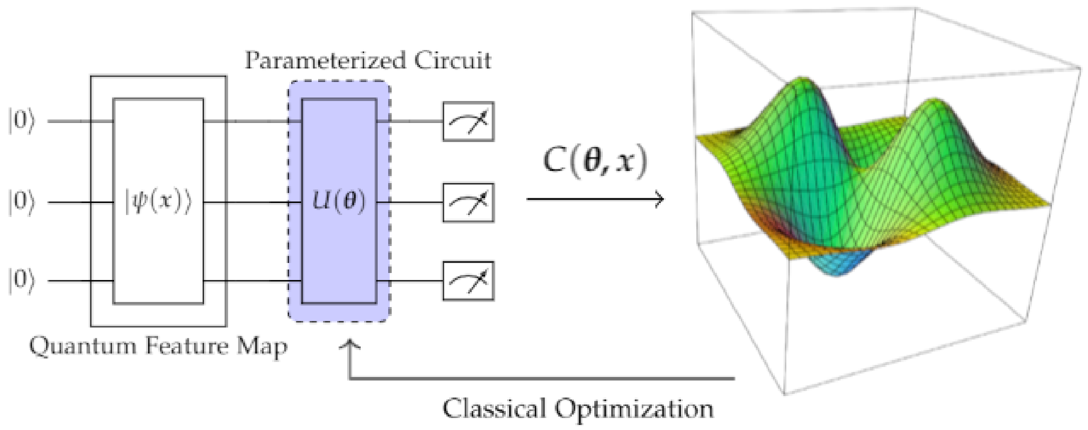


Figure 2.5: Schematic diagram for a quantum variational method.

more, along with ansatz selection, classical optimization influences the exactness of a solution heavily [48].

Initially, a quantum circuit produces output through its measurement device, which yields a *real value*, since all eigenvalues of Hermitian matrices are real scalars [49]. If the goal of a quantum algorithm, is to minimize or maximize the expectation value of an operator, in order to find an approximation of its ground state, this problem is called Variational Quantum Eigensolver (VQE) [50, 51], in which one does not need to prepare a non-trivial classical cost function that utilizes the measurement outcomes. However, while tackling classical problems with quantum variational algorithms, sometimes it is needed to provide an extra cost function, that manipulates the measurement outcomes and produces another real scalar. Either way, as an analogy to classical machine learning and deep learning methods, the quantum circuit itself can be interpreted as a parameterized function, that produces a real scalar output. Hence, in order to generate a learning routine, a classical optimization mechanism must be declared.

There are two main categories of the optimization problems, namely *gradient-based methods* and *derivative-free methods*. One important variable in optimization is the complexity of the objective function evaluation, and the number of function evaluations needed to run the optimizer. The gradient-based methods rely heavily on the objective function evaluation in order to calculate or approximate the derivatives. Therefore, there are several cases where a gradient-based method would not be favoured [52]. For instance, for the cases where the measurement on a function evaluation is either very noisy or time-consuming, the gradient-based methods might be intractable to implement. In such cases, derivative-free optimization methods can be applied. One subset of these approaches is called *simplex methods*, which relies on manipulating the vertices of a simplex, representing function evaluations on different points.[53]. Nelder-Mead method is a simplex methods [54], that is also being used to optimize quantum circuits [48, 55] .

Similar to classical machine learning, the optimization of quantum variational algorithms is generally done via a gradient-based algorithm. The differentiation methods for quantum circuits can be categorized into two, namely *analytic differentiation* and *numeric differentiation* [56].

Even though numerical methods for differentiation quantum circuits were the predominant choice for variational circuits, it is not always a scalable and feasible method for larger circuits with a significant amount of parameters, due to low error tolerance in near-term quantum devices [57].

In the recent past, researchers have proposed methods for calculating the analytic gradients of quantum circuits [58, 59].

Analytic differentiations, make use of the internal dynamics of the gates individually while calculating the derivatives. For the gates that are generated from a Pauli String G, such that $U = e^{-i\mu G}$, its gradient can be found as

$$\partial_\mu U = -iGe^{-i\mu G}, \tag{2.39}$$

which allows calculating analytic gradients, by evaluating the quantum device with the same quantum gate, only with additional phase gates. [57]

Numeric differentiation is done via evaluating the quantum circuit with varying parameters by small margins around the current parameter values. For instance, if a function $f(\boldsymbol{\theta})$ is being used to describe a quantum circuit, then the numerical derivative of that function with respect to the parameters θ is going to be

$$\partial_\mu f(\theta) = \frac{f(\theta + \Delta\mu - f(\theta - \Delta\mu)}{2\Delta\mu} \tag{2.40}$$

The numerical differentiation does not require information on the internal workings of a function, and it only requires evaluating that function with certain parameters several times in order to calculate its gradient. However, it is only an approximation of the derivative of that function over a given point. One notable example of numerical differentiation in the context of quantum computing is the *parameter shift rule* [42]. The parameter shift rule, which has a very natural connection to the finite-differences method, is applied by evaluating the same quantum gate with a small shift in parameters, in order to calculate the gradient. Hence, given an observable $\hat{B}$, the gradient can be calculated [56] via finding the difference between the expectation value of the measured observable $\hat{B}$, such that the derivative of the expectation value

$$\langle \hat{B} \rangle(\boldsymbol{\theta}) = \langle 0| \, U(\boldsymbol{\theta})^{\dagger} \hat{B} U(\boldsymbol{\theta}) \, |0\rangle \tag{2.41}$$

is equal to

$$\partial_{\theta} \langle \hat{B} \rangle(\boldsymbol{\theta}) = \frac{\langle \hat{B} \rangle(\boldsymbol{\theta} + \Delta\omega) - \langle \hat{B} \rangle(\boldsymbol{\theta} - \Delta\omega)}{2\Delta\omega}$$

However, since the parameter-shift rule, is applied for every free parameter, the cost of computation scales linearly and for larger circuits with more parameters, the optimization can become intractable. Nonetheless, there are different optimization methods for quantum devices, that have a similar approach to the parameter shift rule. One exemplary algorithm for that is called the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm [60], which only requires two evaluations, even with a larger number of parameters.

Similar to the gradient recipe of the parameter-shift rule, SPSA gradients have the form

$$\partial_{\theta} \langle \hat{B} \rangle(\boldsymbol{\theta}) = \frac{\langle \hat{B} \rangle(\theta + \boldsymbol{\Delta_{\theta}}) - \langle \hat{B} \rangle(\theta - \boldsymbol{\Delta_{\theta}})}{2\boldsymbol{\Delta_{\theta}}},$$

where Δ_{θ} is a perturbation vector, that has differing perturbation values for each free parameter θ_i. Usually, Δ_{θ} is created via sampling from a distribution (e.g. Bernoulli Distribution), so that the algorithm is not deterministic but rather stochastic.

Furthermore, the following work [61] gives instructions on how to fine-tune hyper-parameters, which might drastically affect the accuracy of the optimization.

For both of these classes of optimizations, it is possible to update the parameters dynamically with an automatic differentiation algorithm [62], such as Stochastic Gradient Descent (SGD) [63]. Furthermore, since both of these methods only make use of evaluations of the gate to be differentiated, with different parameters, they are also quantum-hardware compatible, such that these evaluations are not needed to be done on a classical computer, which can slow down the computation greatly.

3 Methodology

In this section, a novel ansatz is proposed for solving combinatorial optimization problems with constraints. As an example of showcasing the benefits of the proposed method, the Travelling Salesman Problem (TSP) is selected. The very first step of preparing such a solution is to formulate the problem structure to be used in the context of quantum computing. Therefore, the set of possible binary strings, which is going to be the solution space, is mapped to the computational basis states, which are on the quantum system with the same number of qubits as the length of that bit-string. Following the problem formulation, the novel ansatz and the optimization method that is being used are defined and explained in detail.

The implementation is done using Python, and the Qiskit Library [64] for quantum circuit simulation. Same implementation is also done with Pennylane [56] for benchmarking reasons. Simultaneous Perturbation Stochastic Approximation (SPSA) [60] algorithm is used for classical optimization. The code-base is publicly available at: https://github.com/Bmete7/QAOA-E

3.1 Travelling Salesman Problem

Being a well-known optimization problem, the Travelling Salesman Problem (TSP) has commonly been a research subject in various fields of science and engineering. It is categorized as an $NP - Hard$ problem [65], therefore there is no known classical algorithm that can solve this problem exactly in polynomial time.

The problem is formulated as follows: In the TSP, there is a 'map', which is a connected graph, and it includes n number of cities and m total number of different paths between any pair of two cities. The problem also includes a travelling salesman, whose goal is to visit each city on the map exactly once, and finally come back to the origin city as soon as all the cities have been visited. Any route that obeys these criteria, is also known as the *Hamiltonian Cycle* [66]. Each city in this graph is represented by a *vertex*, while each path is represented with an *edge*. Therefore, the cost of travelling along a path is embedded in the problem structure as an *edge weight*.

Furthermore, the overall goal of the salesman is to find the optimal route r_{opt}, such that the travelling cost is minimized, while the described criteria are satisfied. For such a problem, a *divide-and-conquer* approach would not be sufficient, because, each chunk of a sub-problem, is at least as complex as the main problem. Therefore, it belongs to the set of NP-Hard problems. The most naive way to solve this problem, is by a brute force approach, by checking all the possible combinations and finding the minimum cost route. Considering a graph where each vertex is directly connected to all the other nodes, the complexity becomes $\mathcal{O}(n!)$ for n number of vertices. Therefore, a solution is not scalable as it becomes intractable as the graph grows. Figure 3.1 shows an exemplary graph for the Travelling Salesman Problem, that includes 6 vertices and 9 total number of edges. Also, this figure presents a generalized version of a Travelling Salesman Problem structure, since, in some instances, the path to travel between two cities differ in terms of the direction. Here, because this is an undirected graph, the costs for the same path are always the same.

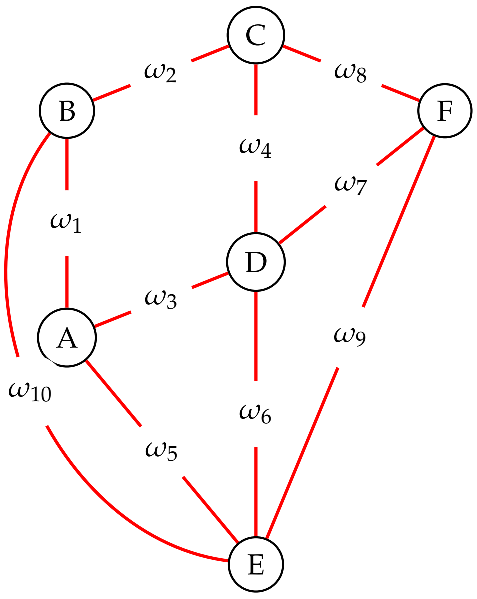


Figure 3.1: An exemplary graph for illustrating the Travelling Salesman Problem using 6 cities. ω, represents the *edge weights*, or the cost to travel from one city to another.

Hence, TSP looks for a solution in a graph $G = (V, E)$, that creates a *vertex tour* G', such that, $G' = (V, E')$ and $E' \subset E$ where each vertex in the tour has a degree of 2 (one incoming and one outgoing edge during the visit). The optimal solution should minimize the tour length, where d_{uv} describes the weight of a specific edge. For such a vertex tour G' that creates a route r, the cost is given as,

$$\sum_{\langle u,v\rangle \in E'} d_{u,v}. \tag{3.1}$$

3.1.1 Problem Formulation

The problem is represented with n^2 binary variables, n being the number of vertices. Each binary variable x_{vj}, displays a certain condition, where a node v is being visited at a step j. For instance, if the binary variable $x_{23} = 1$, that means the $2nd$ node is being visited at the $3rd$ step of the route. Therefore, there are n different binary sub-strings x_v corresponding to a vertex v, that shows when the node v is being visited. Each sub-string x_v has the length of n, where each binary variable refers to a certain step index. For instance, in each binary string s, the sub-string that includes the first n bits refers to the binary random variables related to the first vertex, where each binary random variable at the position j attributes that node being visited in the corresponding step.

For the bit-string representation, the Most Significant Bit 0 (MSB 0) approach [67] is being used, meaning that the most significant bit, represents the 0'th index. For example, if only the first random variable of a sub-string v is 1, whereas all the other $n-1$ bits are 0, then the node v is being visited in the first step, which would make v the origin of the route.

Figure 3.2, displays a graph with 4 nodes, where each node is labelled in the alphabetical order, such that $A = 0, \ldots, D = 3$.

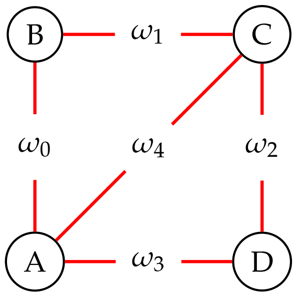


Figure 3.2: Exemplary graph with 4 nodes and 5 edges

For illustrative purposes, a potential solution s as a bit-string,

$$s = 0001\ 1000\ 0100\ 0010, \tag{3.2}$$

exhibits the path, $1 \rightarrow 2 \rightarrow 3 \rightarrow 0$, or $B \rightarrow C \rightarrow D \rightarrow A$ for the graph in Figure 3.2. The cost of the solution s is equal to $\omega_1 + \omega_2 + \omega_3 + \omega_0$.

There are 2 hard constraints of the TSP: Each city should be visited exactly once, meaning that each vertex should appear exactly once in a vertex tour such that,

$$\sum_{j=1}^{n} x_{vj} = 1 \quad \forall v \in V, \tag{3.3}$$

and at the each step of a vertex tour, only one city must be visited,

$$\sum_{v \in V} x_{vj} = 1 \quad \forall j \in E', \tag{3.4}$$

which can be explicitly from the problem statement, but should be defined as the problem is being converted as an optimization problem.

While all possible bit-strings comprising the solution space are represented with the computational basis states in C^{2^N} the feasible subspace is being restricted according to the constraints. Solutions that belong to the feasible subspace are called *valid routes (solutions)*, whereas the other bit-strings that do not satisfy the constraints are called *non-valid routes*. For instance, for a graph with 3 vertices, the binary string $r_1 = 100010010$ represents a non-valid route, as it assigns two different vertices for the same time step. Similarly, the bit-string $r_2 = 000100010$ also showcases a non-valid route, since it fails to cover all the vertices in the graph. On the contrary, the bit-string $r_3 = 010001100$ displays a valid solution, since it obeys both of the constraints. All of the routes mentioned above belong to the solution space for the 3-vertex setting, whereas, only r_3 belongs to the feasible subspace.

An exemplary diagram of the feasible subspace for the 4 qubit setting is shown in Figure 3.3, which can be used for solving the TSP with 2 vertices. In the figure, the outer circle represents the whole Hilbert space for 4 qubits, and the inner blue set S is the solution space, which has 2^4 elements with unique 4-bit binary strings. It is also a subset of the Hilbert space, since it only includes discrete bit-strings, while the Hilbert space has infinitely many elements, with superpositions of basis states. Since the solution space does not have linearity, it is a *non-convex set*. The yellow set F is also non-convex, and it represents the feasible subspace. It is a subset of the whole solution space. It only includes elements, where the constraints of the problem are satisfied. The examples shown on the diagram as elements of the feasible subspace, satisfy the criteria, while $|0000\rangle$, which is not a member of the feasible subspace, clearly does not satisfy the criteria mentioned in Equation 3.3 and Equation 3.4.

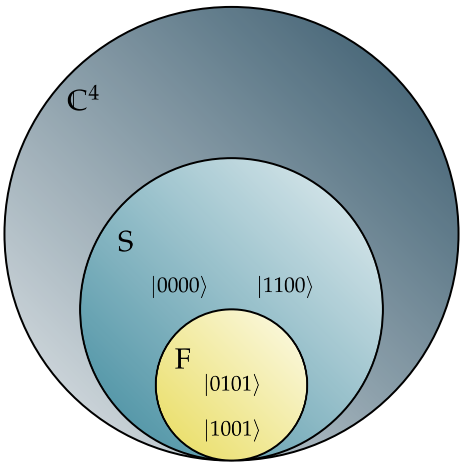


Figure 3.3: An illustration for showing the difference between the solution space and the feasible subspace.

The first step for preparing a QAOA Ansatz is to formulate the cost Hamiltonian and find the cost unitary accordingly. Since the soft constraints consist of the cost of a route, the classical objective function can be formulated as,

$$C = \sum_{\langle u,v \rangle \in E'} d_{u,v}, \tag{3.5}$$

where E' is the set of edges that are covered in the route.

Considering the set of solutions S including bit-strings with n^2 binary random variables, the objective function $C : S \to \mathbb{R}$ is defined as,

$$C = \sum_{v=0}^{n} \sum_{u \neq v} \sum_{j=0}^{n-1} \sum_{i=j+1}^{n} d_{u,v} \;\; \delta_{v \times n+j} \;\; \delta_{u \times n+i} \tag{3.6}$$

where,

$$\delta_x \begin{cases} 1 & \text{if } S[x] = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

Since there are n^2 different binary random variables, each variable can be represented with a random variable $x_{u,j}$, which means, "the node u, being visited at the step j". Throughout the following sections, the binary setting of the random variable x_{vj} has been used. Equivalently, the same random variable can also have a binary form, $x_{u \times n+j}$.

The objective function can be converted into a cost Hamiltonian, as it is shown on Equation 3.8.

$$C = (2 - \frac{n}{2}) \sum_{\langle u,v\rangle \in E} d_{u,v} + \sum_{\langle u,v\rangle \in E'} \sum_{j=0}^{n-1} d_{u,v}(Z_{u\times n+j} \ \ Z_{v\times n+j+1} + Z_{u\times n+j+1} \ \ Z_{v\times n+j}). \quad (3.8)$$

Each edge E, that connects two vertices u and v, can be used to describe two different paths in a route with alternating orders $u \rightarrow v$ and $v \rightarrow u$. Therefore, there are two σ_z terms in the cost function as a sum, describing alternating path formations.

3.2 The Novel QAOA Ansatz with an Encoder

Since the problem structure that we are trying to tackle is a constrained problem, the desired output should be an element of the feasible space F, and not the more generalized solution space S. Instead of preparing ad-hoc mixer unitaries for projecting the solution onto the feasible subspace, another non-variational ansatz can be created for the same purpose.

The ansatz is prepared as an *encoder*, which separates the solution space into two subspaces A and B, that have the dimensions k and $(N-k)$ respectively, where $N >> k$. If the input of the encoder is a valid solution, then the encoder outputs a state, where the qubits in the latter subspace B are all equal to $|0\rangle$. This would ensure that the state in subsystem B is always the same when the input is a valid solution. Therefore, subsystem B can be discarded after encoding, while subsystem A, which is called the *latent space*, is encoded with a state that can solely describe a unique valid solution. Therefore, the encoder mechanism can be summarized as,

$$\mathcal{E}\,|\psi\rangle = \begin{cases} |\phi\rangle_A \otimes |0\rangle_B^{\otimes(N-k)} & \text{if } |\psi\rangle \in F \\ |\phi'\rangle_A \otimes |\phi''\rangle_B^{\otimes(N-k)}, & \text{if } |\psi\rangle \notin F, \text{where, } |\phi''\rangle \neq |0\rangle_B^{\otimes(N-k)}. \end{cases}$$

Furthermore, the *decoder* ansatz $\mathcal{D}$, which is nothing but the inverse of the encoder unitary, can be used to reproduce a valid solution in the N-dimensional original subspace. Therefore, if the qubits in subsystem B, which is also called *trash system*, are all mapped to $|0\rangle_B^{\otimes(N-k)}$, the decoder is ensured to output a valid solution. Hence, even though the original input to the encoder is not a valid solution at all, if all the qubits in subsystem B are discarded, or set to $|0\rangle$, the decoder then maps that intermediate state into a valid solution. The encoding and decoding mechanism is illustrated in Figure 3.4.

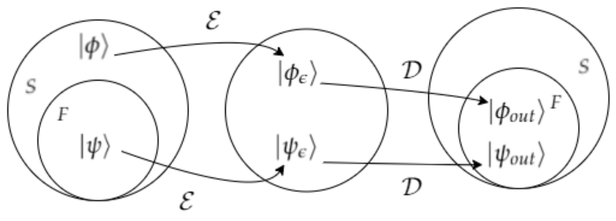


Figure 3.4: A transitional scheme, illustrating the encoding mechanism. both states, $|\psi\rangle \in F$ and $|\phi\rangle \in S$ are mapped to the feasible subspace after encoding/decoding. Note that, here decoding subroutine $\mathcal{D}$, refers to the swapping of subsystems B and B', followed by the decoder.

However, when a system is discarded (traced out) and set with a completely new state, the gradient tracking for the parameters up to that point will be lost. For this reason, the discarding approach is not suitable for variational algorithms. Nevertheless, one method to overcome that, which is also demonstrated in [68], is to have another subsystem B' that has $N-k$ ancillary qubits, that are all initialized with a reference state $|0\rangle_B^{\otimes(N-k)}$. Instead of tracing out the environment B, a simple swap operation (swap gate), between subsystem B and the reference system B' makes sure that all the qubits in subsystem B are equal to $|0\rangle$. Since the ansatz is still reversible, the gradient flow is not interrupted. Therefore, the ansatz requires extra $l \times (N-k)$ ancillary qubits, where l describes the number of times the encoding is employed. The overall QAOA ansatz with encoding and decoding mechanism is illustrated in Figure 3.5. In this diagram, the qubits in subsystems A and B are initialized as an equal superposition state, which includes every member of the solution space, including the non-valid solutions. Following the initialization, the unitary based on the cost Hamiltonian is applied, which has the purpose of phase separation, according to the weights of the edges. Since the states after the cost unitary, also include non-valid solutions, the encoder is applied in order to project each solution onto the feasible subspace. After the encoder, subsystem A comprises the latent space, where each state corresponds to a unique solution. Subsystem B ideally has the state $|0\rangle^{N-k}$, whenever the input to the encoder stands for a valid solution. To project the state-vector back onto the feasible subspace, subsystem B′ is discarded, via swapping it with a reference system that is initialized with $|0\rangle$ states. Hence, after the following decoding operation, each state on the final system, that has a probability value larger than 0 refers to a valid solution. If there is more than one iteration in the QAOA, then extra ancillary qubits must be provided in the reference system.

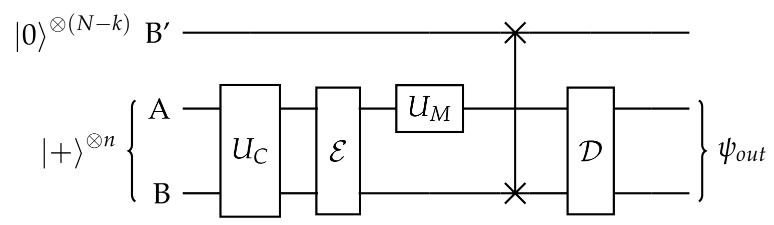


Figure 3.5: An illustration of one QAOA iteration, with an encoder. The final state $|\psi_{out}\rangle$ is guaranteed to be a part of the feasible subspace, which stands for a valid solution.

This approach eliminates the need to construct mixer Hamiltonians, whose ground space spans the feasible subspace. Nevertheless, the mixing unitaries can still be utilized in order to create a hopping mechanism across the feasible subspace. The hopping mechanism ensures that the entirety of the feasible subspace is being explored. The mixers do not have to apply to all of the N-qubits. Instead, the mixer mechanism can be applied to the latent space, which will restrict the exploring phase into the feasible subspace. Therefore, a trivial mixer can be created as,

$$H_M = \sigma_x^1 \sigma_x^2 \ldots \sigma_x^k, \tag{3.9}$$

which is trivial to prepare, both for the simulation environment, and for a quantum hardware.

The role of the mixer, can also be displayed as a transition between two computational basis states x_0 and x_1, such that $|\langle x_0| U_M(\beta) |x_1\rangle| > 0$.

Finally, the algorithm includes l approximation iterations. Each iteration, starts with a parameterized unitary, based on the cost Hamiltonian. After that, the encoder ansatz is applied, to map the state vector into the latent space. With the intention of ensuring that the latent space only gets decoded into the valid paths, a discarding mechanism is applied, by swapping the remaining subspace with a pre-initialized reference state, as explained above. Then, in order to create a mixing mechanism, the mixer unitaries are applied

$$U_M(\gamma_l) = e^{-i\sigma_0^X \gamma_l} \cdot \ldots \cdot e^{-i\sigma_k^X \gamma_l} \tag{3.10}$$

that can be realized with k number of X-rotation gates that apply to the qubits in the latent space. After the decoding, the latent space gets mapped to the original solution space, where each value is a member of the feasible subspace, as illustrated in Figure 3.4.

The mixers, have another significant role in the general QAOA ansatz. Since the mixer Hamiltonian, does not commute with the cost Hamiltonian, they do not share the same eigenspace. Therefore, by applying a mixer, the state is mapped to a different space, which is not an eigenspace of the cost Hamiltonian anymore. If the mixing mechanism is not present in the ansatz, the cost unitary itself does not alter the probabilities of the initial input state vector, since the H_C is a diagonal operator and its eigenvectors are the computational basis states.

Another way of looking at this phenomenon is to think of the approach as a time evolution process, governed by H_C. Since for a system that is evolving under the closed system dynamics, the system's energy will be conserved, by doing a time evolution which is governed by the same Hamiltonian H_C, the energy of the system will not change over time.

Thus, by only having the cost unitaries based on H_C, the optimization would not converge, since changing the parameters of the cost unitary alone would not create any difference in the expectation value of the final state.

By having a mixing mechanism, the state can jump out of the eigenspace of the cost Hamiltonian, therefore, a change in the parameters can result in a change in the expectation value, which enables the optimization routine to converge. Hence, an optimized prediction can be obtained.

Since the encoder also does not share the same eigenspace with the cost Hamiltonian, it makes sure, that not only the constraints are satisfied in the final prediction, but also that the state is mapped to a different eigenspace. Therefore, the encoding mechanism succeeds in replacing the mixer mechanism in 2 different aspects, namely the constraint satisfaction checking, and jumping out from the cost Hamiltonian eigenspace. However, the trivial mixer unitaries are still handy for creating the hopping mechanism over the feasible subspace.

Eventually, the encoding scheme, interleaved with trivial mixing unitaries has 3-main capabilities:

1. Restricting the search space into a much smaller latent space.
2. Ensuring that the final output, sampled from a measurement, produces a bit-string that satisfies the problem-specific constraints.
3. Creating a mixing mechanism in the latent space, so that the entirety of the feasible subspace is being explored.

Furthermore, another essential part of the QAOA ansatz is the initial state preparation. In QAOA, the initial state is generally prepared as the eigenvector of the solution space that corresponds to the highest energy eigenvalue. That state coincides with the eigenvector of the mixer Hamiltonian. Preparing the initial state as the highest energy eigenstate of the mixer Hamiltonian is crucial since in that case, the evolution occurs naturally on the feasible space [38]. In the most trivial case, where the solution space is not restricted, the mixer Hamiltonian is defined as

$$H_M = \sum_{i=1}^{N} \sigma_i^x,$$

which is also known as the transverse-field mixer. The eigendecomposition of the Pauli-X gate is given as,

$$\sigma_x = +1 \cdot |+\rangle \langle+| - 1 \cdot |-\rangle \langle-| \,.$$

Therefore, in such a case where the mixer Hamiltonian consists of σ_x operators, the initial state is prepared as $|\psi'\rangle$, where

$$|\psi'\rangle = |+\rangle^{\otimes n} \,.$$

However, initial state preparation becomes more non-trivial by using the novel approach of using an encoder architecture to map the state into the feasible subspace. Since finding the highest energy eigenstate of the Hamiltonian governing the encoding unitaries is burdensome, a different approach must be followed to prepare the initial state. Similarly, the desired initial state should be an equal superposition of each state in the feasible subspace. Instead of finding the energy eigenspace of the mixer Hamiltonian, preparing an equal superposition in the computational basis states just for the *latent space*, encoded in subsystem A, would encapsulate each possible solution. After preparing an equal superposition in the latent space, an additional decoder must be applied, to project the states back onto the solution space from the latent space. A sub-circuit for preparing the initial state is illustrated in Figure 3.6.

Finally, after the ansatz is complete, a measurement along the computational basis state is done, with the operator $\hat{H}_C$, since the desired solution is the maximum energy eigenstate, that also resides in the feasible subspace F. Therefore, the expectation value of $\hat{H}_C$ can be calculated, or estimated via sampling in a simulator, to obtain the final output. One QAOA iteration is terminated, via running a classical optimization routine, to optimize the expectation value. The algorithm is terminated, when the optimization converges.

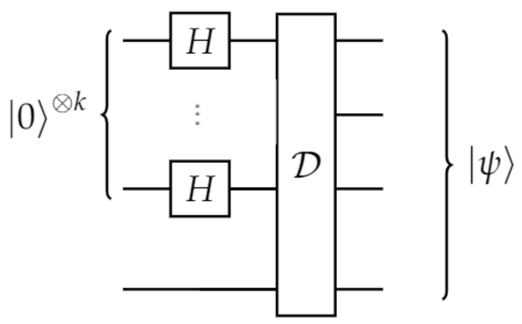


Figure 3.6: The sub-circuit for preparing an initial state as the highest energy eigenstate of the mixer Hamiltonian

Therefore, the steps of the novel QAOA ansatz for solving constrained optimization problems can be summarized as follows:

1. Initialize subsystem AB as $|+\rangle^N$ and the reference subsystem as $|0\rangle_B^{\otimes(N-k)}$,
2. Initialize $2 \times l$ parameters for β and γ,
3. Prepare the cost Hamiltonian depending on the problem structure,
4. Prepare the mixer Hamiltonian as the transverse-field Hamiltonian $\sum_{i=1}^{N} \sigma_i^x$,
5. Apply the unitary transform $U_C^l(\beta)$ based on the cost Hamiltonian,
6. Apply the encoder to map the state onto the latent space
7. Apply the mixer unitary $U_M^l(\gamma)$
8. Swap the trash system B with the reference system B'
9. Decode the system to map the state onto the feasible subspace, if the current iteration is not the final QAOA iteration
10. Repeat the steps between 5-9 l times, where each cycle represents a QAOA iteration, l being the number of QAOA iterations. In the final iteration, skip the decoding part, so that the output state is in the latent space.
11. Sample S number of final states from the circuit or calculate the expectation value $\left\langle \psi_{\vec{\beta},\vec{\gamma}} \middle| H_C \middle| \psi_{\vec{\beta},\vec{\gamma}} \right\rangle$,

12. If sampling is used, then provide a classical loss function $\mathcal{L}$ based on the log-likelihood function, to overcome numerical instabilities, which has the standard form $\mathcal{L}(\beta, \gamma) = \log_2(1 - U_M^p(\beta)U_C^p(\gamma) \dots U_M^1(\beta)U_C^1(\gamma))$ [69],
13. Optimize over the parameters β and γ with a classical optimizer,
14. Iterate until convergence.

3.2.1 Encoder Structure

First of all, throughout this chapter, the quantum state vector that the system is in is sometimes referred to as a *bit-string*. Each bit-string with the length of n corresponds to a computational basis state. Even though, the system can be in a superposition of different computational basis states, the term bit-string, will refer to only one of the basis that appears in the superposition. Therefore, if a system is in a state

$$|\psi\rangle = \frac{|s_1\rangle + |s_2\rangle + \cdots + |s_N\rangle}{\sqrt{N}},$$

each operand that acts on $|\psi\rangle$, can be thought of as an operator that acts on N different bit-strings. Therefore, the two distinct terms 'quantum state' and 'bit-string' are used interchangeably throughout this section, within the context explained above.

For a complete graph, there are $n!$ different possible paths that constitute a valid solution. However, in the problem setting, they are represented with n^2 qubits, even though $2^{2^n} - n!$ of those states refer to a non-valid solution. Hence, instead of representing a valid solution with an n^2-bit string, there exists a more efficient representation, that inhabits a latent space that utilizes k qubits, where $k = \log_2(n!)$. The encoding and decoding scheme for an example with 3 vertices is depicted in Figure 3.7. In this transitional diagram, two distinct cases are presented, that indicate a valid and a non-valid state respectively. The first step is applying the encoder, to find the latent space representation. Following the encoder, the remaining qubits in the trash subsystem B are reset into 0, which can be realized on a quantum device with a swap operator between another reference subsystem B'. As it can be seen from the graph, two distinct solutions, where one of them exemplifies a non-valid solution, get mapped into the same bit-string, that represents a valid solution

The encoder ansatz can be created in various ways. A pivotal factor in how such a mapping into a low-dimensional space is possible is the fact that the feasible subspace is much smaller than the original solution space. While, for each sub-string i, there are

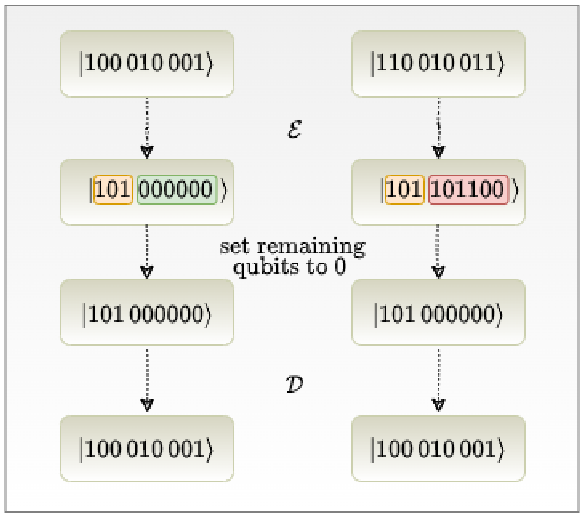


Figure 3.7: Two transitional diagrams for the encoder/decoder subroutines. The transition on the left displays a case where the input stands for a valid solution, whereas the figure on the right displays a non-valid solution. Orange blocks represent the latent subsystem A, where the green and red boxes represent the trash subsystem B.

2^n different bit-strings possible, only n of them refer to a valid solution. Hence, each sub-string can be encoded into $\lceil \log_2(n) \rceil$ qubits. For instance, for a 5-qubit setting, the encoding can be done as follows:

$$10000 \xrightarrow{\mathcal{E}} 00000$$
$$01000 \xrightarrow{\mathcal{E}} 00100$$
$$00100 \xrightarrow{\mathcal{E}} 01000$$
$$00010 \xrightarrow{\mathcal{E}} 01100$$
$$00001 \xrightarrow{\mathcal{E}} 10000$$

Note that the final encoding uses only $3 = \lceil \log_2(5) \rceil$ qubits, whereas the original sub-string uses 5. Since there are n total number of sub-strings, total number of qubits needed will be equal to $n\lceil \log_2(n) \rceil$

Another encoding scheme can be applied as follows: First of all, the first sub-string that stores information about the initial vertex is encoded into $n-1$ bits. The encoding is done, via applying a multi-controlled X gate (a generalization of the Toffoli gate) to the nth bit, with inverted controls (the target qubit is flipped, whenever the control bits are 0). If the input is a valid solution, there are two possibilities; either the first vertex is being visited at the last step or another step. For the former case, the nth qubit would originally have a value of 1, whereas all the other qubits are 0. For the latter, the nth qubit is 0. It is trivial to notice that the nth qubit ends up in a 0 state in both cases after applying a multi-controlled X gate. Eventually, the final state still preserves an encoding that carries information about the first vertex. If the ith qubit in the sub-string has the value of 1, that means that that vertex is being visited at the ith step. If all of the $n-1$ qubits have the value of 0, that means the vertex is being visited at the last step (step n). Similarly, this method can be applied to all the sub-strings, to have extra qubits to store the encoding.

This routine is applied, in order to reset some of the resources and use them as storing system for the encoding. Moreover, even a shallower network can be utilized for the same purpose, with the use of additional ancillary qubits to store the encoding information, without needing to reset some portion of the resources.

Furthermore, for the remaining $n-1$ sub-strings, a comparison mechanism is applied, to store the information on whether the ith vertex is being visited before or after the jth vertex, where $i \neq j$. The comparisons between two sub-strings can be realized as follows: A vertex u, having appeared in the route earlier than the vertex v, means that, the bit-string u yields a larger binary value than the bit-string v. For instance, suppose a sub-string u is $10\ldots0$, whereas the sub-string v is $0010\ldots0$. In this case, the vertex u is being visited before the vertex v, which indicates the condition $u > v$. In such a case, the encoding bit should store the value 1, whereas in the opposite case where $u < v$, it should store 0. Such a comparison can be realized by a set of Toffoli gates. Each Toffoli gate consists of two control bits and one target bit. If both of the control bits are in the state 1, where the one control qubit is the ith bit in the sub-string u, whereas the other is the jth bit of the sub-string v, such that $i > v$. The target qubit is always the same qubit, which will store the result of the comparison. If the whole bit-string s, stands for a valid solution, where u is being visited before v, then exactly one of the Toffoli gates will have both control qubits set to 1, which in turn sets the target qubit to 1. However, if u is being visited after v, then none of the Toffoli gates will have both of the control qubits set to 0, therefore the target qubit stays at 0, which signifies that $u < v$. The comparing mechanism along with an exemplary circuit for comparing two sub-strings is presented in Figure 3.8.

Even though there are $\frac{(n-1)(n-2)}{2}$ different sub-strings to compare, there is a better comparison mechanism, that minimizes the number of qubits needed for storing the information. The comparison mechanism is quite comparable with the comparisons

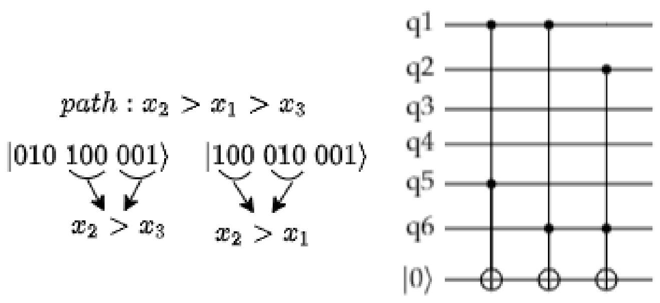


Figure 3.8: The encoding algorithm and the ansatz for a 3-vertex setting. The algorithm calculates the query $u > v$, and its result is stored in the final ancillary qubit, which has been reset before.

done in merge sort, which is a recursive sorting algorithm, that works with a *divide and conquer* strategy. Since, there are $n-1$ remaining vertices, there are also $(n-1)!$ different possible valid solutions. With the mentioned divide-and-conquer approach for comparing different combinations, the required number of qubits to store the comparison information becomes $\log_2(n-1)!$. Since,

$$\log_2(n-1)! = \log_2(n-1) + \cdots + \log_2(1)$$

$$n\log(n) = \log_2(n) + \log_2(n) + \cdots + \log_2(n)$$

$$\log_2(n-1)! \leq n\log(n),$$

asymptotically, those two complexities are equal, such that,

$$\mathcal{O}(\log_2(n-1)!) = \mathcal{O}(nlogn)$$

. A complexity analysis between the two functions is also presented in Figure 3.9. Therefore, the second encoding approach is even more efficient than the first proposed encoding mechanism.

The final step is to, prepare another ansatz to reset each qubit in subsystem B to 0, if the input was a valid solution. The ansatz is created with the same strategy as the encoder, by applying the Toffoli gate with the control qubits indicating a portion of a valid path.

One potential problem can arise, when

$$k = \log_2 n!, \quad \text{where}, k \notin \mathbb{Z},$$

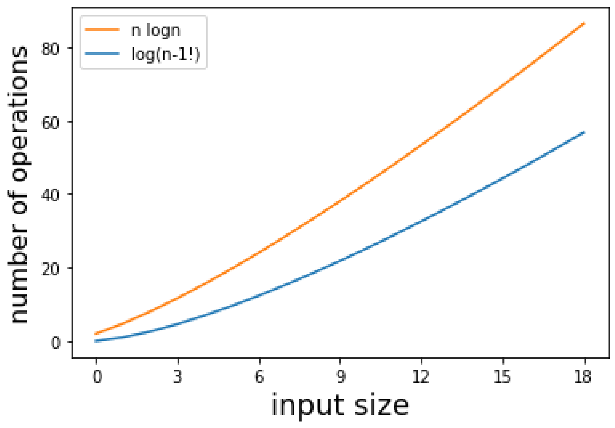


Figure 3.9: Comparing the complexities between $nlogn$ and $\log_2(n-1)!$

which means the number of valid paths is not exactly a power of 2. In such cases, some states in the latent space will refer to a non-valid solution after the encoding. One potential solution to that problem is to provide a penalty mechanism in the classical loss function for the cases where the bit-string does not correspond to an encoding of a valid solution. Therefore, a relaxation with $2^k - n!$ dummy solutions can be added with a very large route cost, such that these solutions are not favoured.

3.3 Optimization

There are several settings for benchmarking the training routine, namely, the size of the problem set, connections in the graph, the optimizer selection, loss function definition, and the hyper-parameter selection.

Firstly, there are several optimization algorithms is being used for the training routine for benchmarking reasons.

One of the optimization methods is the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm, which is a gradient approximation method based on the finite differences method. In this method, in contrast to the analytic-gradient-based algorithms, only the function evaluations are used in order to approximate the gradients with respect to the parameters β, γ. Furthermore, it also differs from the finite differences method, since it requires only two function evaluations in total for one iteration. In finite differences, each parameter is being shifted independently from the others, therefore it requires 2^p function iterations, where p is the number of free parameters.

In SPSA, the measurements given an operator $\hat{O}$, such that

$$\langle \hat{B} \rangle (\boldsymbol{\theta}) = \langle 0 | \, U(\boldsymbol{\theta})^{\dagger} \hat{B} U(\boldsymbol{\theta}) \, | 0 \rangle \, , \tag{3.11}$$

can be utilized to approximate the parameters, such that,

$$\partial_{\theta} \langle \hat{B} \rangle (\boldsymbol{\theta}) = \frac{\langle \hat{B} \rangle (\theta + \boldsymbol{\Delta_{\theta}}) - \langle \hat{B} \rangle (\theta - \boldsymbol{\Delta_{\theta}})}{2 \boldsymbol{\Delta_{\theta}}}.$$

One important aspect of this algorithm is, that a perturbation vector Δ_{θ} is being created stochastically, such that the derivatives of each variable are approximated within different internals. Generally, Δ_{θ} is sampled through a distribution (e.g. Bernoulli Distribution, Gaussian Distribution), as opposed to the finite differences method.

Another optimization methods used are derived from the Stochastic Gradient Descent (SGD) [70], namely, the Adam Optimizer [71] and the RMSProp [72]. SGD is a gradient-based optimization method that utilizes the back-propagation technique for updating the parameters. However, as the circuit grows deeper, back-propagation-based methods become less favourable due to their long execution times.

The output of the circuit has 2 distinct possibilities: First, the output can be the expectation value of the cost Hamiltonian H_C. In the training routine, the optimizer tries to minimize the expectation value of H_C. Therefore, since the value that is expected to be minimized is the measurement value itself, there is no need to define an additional loss function. In order to assess the success of the training routine, there is another output type defined, as a sampling of the state vector for predicting the most optimal route. The sampling is done 1024 times, whereas the choice of the optimal route is being decided upon a *majority vote* method. The state which has been sampled the most is selected as the model prediction.

Hyper-parameter selection is another crucial factor that significantly affects training success. Therefore, several different parameters are being evaluated for training accuracies.

4 Experimental Results

The experiments are run on 3 and 4-vertex graph settings, with 2-different number of layers. However, the optimization is only done using the 3-vertex setting due to the limitations of simulating the QAOA ansatz. The loss function defined is a dummy function, which returns the expectation value of the H_c. As discussed in Section 3.1.1, The H_C is defined as,

$$C = (2 - \frac{n}{2}) \sum_{\langle u,v \rangle \in E} d_{u,v} + \sum_{\langle u,v \rangle \in E'} \sum_{j=0}^{n-1} d_{u,v} (Z_{u \times n+j} \quad Z_{v \times n+j+1} + Z_{u \times n+j+1} \quad Z_{v \times n+j}). \tag{4.1}$$

Therefore, the goal of the optimizer is to find the set of parameters, that minimizes the expectation energy of H_C. Note that the H_C can also be stored as a sparse matrix since the Hamiltonian grows exponentially as the system size increases. Furthermore, since it is a Hermitian matrix, it is also possible to generate the Hamiltonian as a linear combination of different Pauli strings.

The results seek to find two main insights. Firstly, the learning success and the efficiency of the novel approach presented in this Thesis are investigated, with different optimizers and problem settings. Secondly, a benchmark is proposed, comparing the novel approach with the state-of-the-art methods in solving constrained optimization problems. Therefore, along with the QAOA ansatz with encoders, a different method proposed in [2] is also implemented for benchmarking. In that method, the hard constraints of the problem are embodied inside the mixer Hamiltonian, with a similar purpose of restricting the exploration into a feasible subspace. For TSP, the mixer Hamiltonian is defined as,

$$\sum_{\langle u,v \rangle \in E} \prod_{j=1}^{n} (\sigma_{uj}^{x} \sigma_{vj}^{x} + \sigma_{uj}^{y} \sigma_{vj}^{y}). \tag{4.2}$$

For benchmarking, the main factors are once again the learning curve convergence and the model specific details that affect the performance and efficiency. Therefore, while the former is being used to compare the training accuracy, the latter provides metrics in terms of the applicability of the models, that measure the complexities and the resource utilization of the models. The novel QAOA ansatz proposed in this work, which uses encoders, is abbreviated in this section as *"QAOA-E"*, whereas the method proposed in [2] is called as *"QAOA-M"*. The following number after the abbreviations indicates the number of QAOA iteration layers.

As it can be seen in the Equation 4.2, in QAOA-M, the mixers are defined as different combinations of the XY-model Hamiltonian, where the overall Hamiltonian H_M can be expressed in terms of a sum of $(2^n) \times m$ terms, with $m = |E|$. The cost Hamiltonian is the same for both models, which means the global minima are the same for each model.

The experiments for the training accuracy, are realized on a small-scale example of the TSP problem, that only has 3 different cities, and 6 possible valid routes. The graph that is used in the experiments, is illustrated in Figure 4.1. In this toy example, two of the edges share the same weight, while the other edge, which is connecting the nodes 1 and 2, has a significantly higher weight. Hence, in that graph setting, the states that refer to the most optimal routes are encoded as: 010 100 001 and 010 001 100, that refer to the routes $1 \rightarrow 0 \rightarrow 2$ and $2 \rightarrow 0 \rightarrow 1$ respectively, which are the only possible routes that do not include the edge that connects the nodes 1 and 2.

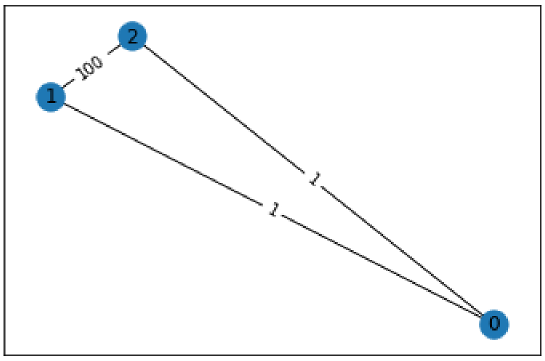
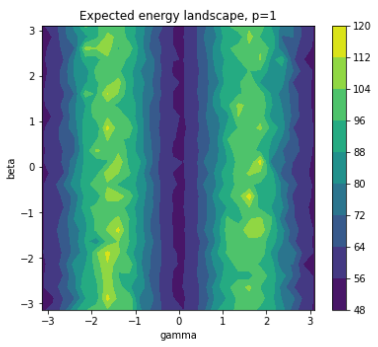


Figure 4.1: Weighted Graph for the TSP with 3 cities.

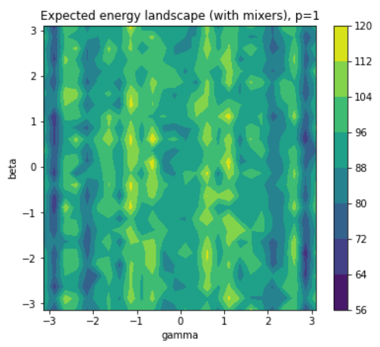
The following sections are divided into two, that both assess the QAOA-E model, and its benchmark results with QAOA-M, in terms of their learning success and performance, in the given order.

4.1 Training

In order to analyze the behavior of the optimizers on the objective function, an energy landscape plot can be used to represent the expectation value of H_C with various parameter configurations. For the mentioned 3-vertex graph, and the circuit setting with 2 parameters, which includes a single QAOA iteration cycle, the energy landscapes for both models are presented in Figure 4.2.



(a) QAOA-E1



(b) QAOA-M1

Figure 4.2: Energy landscape across different configurations of β and γ. The values on the bar indicates the energy, or similarly the expectation value of the cost Hamiltonian. The figure on the left displays the energy landscape of the QAOA ansatz with encoders, while the figure on the right illustrates the same metric for the QAOA ansatz with problem specific mixer Hamiltonians.

As it can be seen in the figure, for the QAOA-E, the preeminent parameter is the γ, the parameter of the cost unitary, where β does not seem to play a crucial role in the optimal configuration setting. However, for the QAOA-M, even though this behavior is still present on a small scale, the mixer parameters also play a significant role in finding the global optimum.

Furthermore, one conspicuous remark based on the energy landscapes, is that using the QAOA ansatz with an encoder, significantly relaxes the objective function, as it creates a smoother objective function, with a greater number of parameter configurations that yields an optimized solution.

To find the learning success, two distinct metrics are being measured for the training and validation accuracy. While the training accuracy is examined through the evolution of the objective function evaluations, the validation accuracy is calculated from the model predictions.

Consequently, both models have been trained with three different optimizers, namely the SPSA [60], RMSProp [72] and Adam [71]. These optimizers have been specifically selected since they incorporate two distinct types of optimization techniques, namely the analytic and numerical methods of gradient calculation.

SPSA is implemented using noisyopt library [73], with $a = 0.2, b = 0.15$ as the step size hyper-parameters. For both Adam, and RMSProp the learning rate is selected as $3 \cdot 10^{-2}$.

The learning curves for the QAOA-E1 model with different optimizers is presented in Figure 4.3, where the y-axis signifies the expectation value of the cost Hamiltonian.

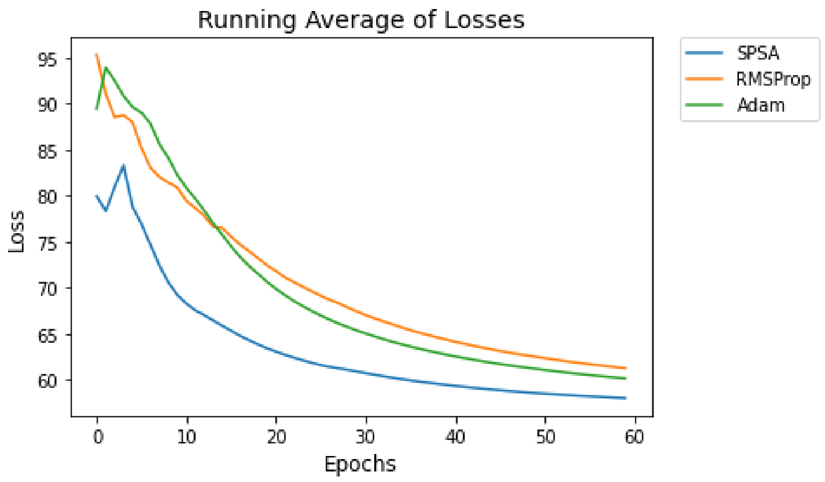


Figure 4.3: The running average plot of the evolution of loss using 3 different optimizers for QAOA-E1.

Furthermore, the predictions of the models are acquired as follows: Instead of measuring the expected energy of the system given the Hamiltonian H_C, the model returns a sample from the latent space during the validation and the test modes. Therefore, in the last iteration, discarding and decoding phases are skipped, and the measurements are done on the encoded states. In order to create a more robust predictor that is less prone to noise, the sampling is done abundantly (1024 times in the 3 vertex setting), where the final sample is selected using the *majority vote* approach. The majority vote approach, basically means that the state that is sampled the most is being selected as the model prediction. The accuracy plot of QAOA-E1 with the same problem setting is presented in Figure 4.4.

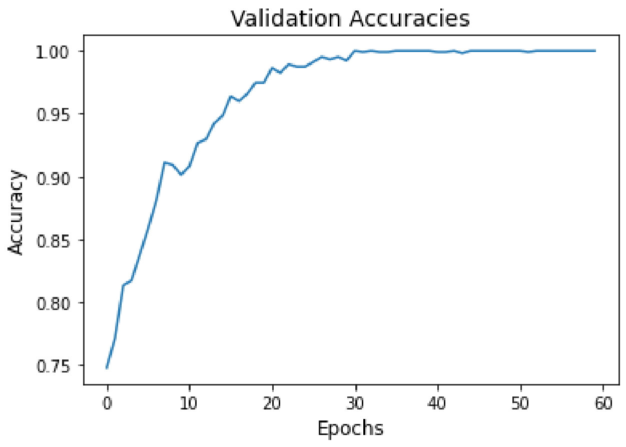


Figure 4.4: The evolution of the validation accuracy of QAOA-E1, using the RMSProp optimizer.

Finally, the two models have been trained SPSA, and the same hyper-parameters for a different graph, which also has the same optimal solution, but with slightly different edge weights (respectively 6, 10 and 15). Therefore, the ground state energy of the Hamiltonian has a different energy spectrum. The comparison of convergences of two models has been plotted in Figure 4.5

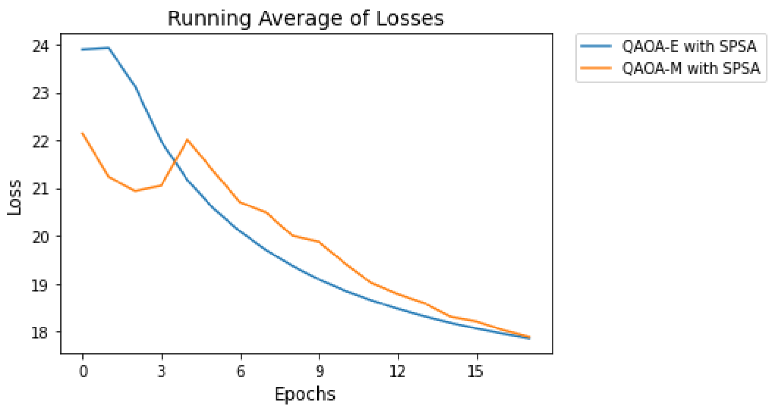


Figure 4.5: Comparison of learning curves of QAOA-E and QAOA-M, using SPSA

4.2 Model Parameters

There are several circuit parameters that can be used to determine the algorithm efficiency. One significant parameter that drastically influences the time complexity of the circuit evaluation is the circuit depth. As the depth of the circuit grows larger, the classical optimizers that rely on back-propagation, perform worse, both in terms of time efficiency, and learning accuracy, as the vanishing gradient problem [74] poses challenges in finding the optimal parameter configuration. Two other important criteria are the number of qubits, and the number of operations utilized in the circuit which increases the vulnerability to the noise of the circuit.

		Comparison between two models			
Model	Number of Vertices	Circuit Depth	Number of Operations	Number of Qubits	Epoch Duration for RMSProp
QAOA-El	3	18	24	$l \cdot 6 + 9$	$0.12s$
	4	154	174	$l \cdot 10 + 16$	$0.20s$
QAOA-Ml	3	147	552	9	$0.71s$
	4	485	2976	16	$23.5s$

Table 4.1: Comparing QAOA-E and QAOA-M in terms of their efficiencies using several metrics.

The depth of the model QAOA-M1 approximately equals to $(2^n \cdot n \cdot m)(4n+1)$, where m denotes the number of edges in the graph. Within the same problem settings, the depth of the QAOA-E1 ansatz equals to $2^{n-1} \cdot (n-1) \cdot (n-1) + \left(\frac{n \cdot (n-1)}{2} + 1\right) \cdot (n-1)$, which does not utilize the edge information and is the same for every n-vertex graph setting.

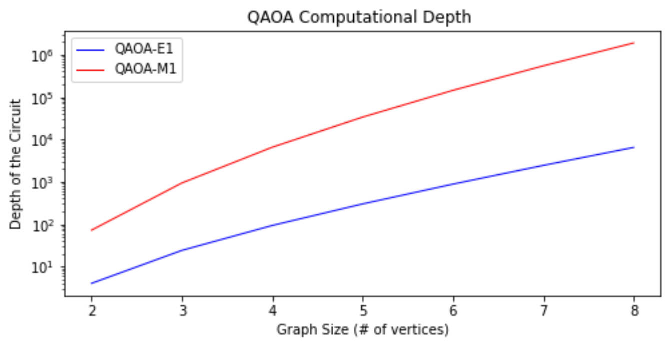


Figure 4.6: Asymptotic complexity of the models with respect to their circuit depths.

Comparison results between QAOA-E and QAOA-M, in terms of the model parameters, are given in Table 4.1. Furthermore, the semilogy plot in Figure 4.6 visualizes the asymptotic computational complexity of both of the methods in terms of their circuit depths.

5 Conclusion

To sum up, this thesis aimed to provide an alternative solution for solving constrained optimization problems. It can be concluded that the QAOA-E is proposing a general-purpose method, which is also shown to be working efficiently on small-scale examples. Compared to the other state-of-the-art methods, several resolutions can be inferred from the experimental results, on when it is more plausible to use QAOA-E over different methods.

Firstly, as it can be seen on the energy landmark plot, in the QAOA-E, the parameter of the mixer unitaries evidently are not decisive factors in the expectation value of the cost Hamiltonian. Instead, only the parameters of the cost unitaries are critical in finding the optimal configuration. Because the projection of the states onto the feasible subspace is done by the non-variational encoder sub-circuit, the mixers only have a limited role, that is creating an exploration mechanism over the states that are already in the feasible subspace. For the same reason, in QAOA-E, the convergence in training happens much faster, since the number of parameters to be optimized is halved, and the objective function has a smoother form. In QAOA-M [2], the same projection occurs by using the problem-specific mixer Hamiltonian models, whose eigenstates refer to the feasible states. Therefore in this family of solutions, the correct configuration of the mixers affects the final expectation value more. However, since the experiments are done on a very small-scale example, the difference in the training speed is not eye-catching.

Furthermore, even though the mixer unitaries proposed in QAOA-M have trivial forms that are easy to implement, QAOA-E is also comprised of a set of gates that can be implemented in the same way, according to the algorithm explained in Section 3.2 that only includes CNOT's, Pauli gates, and the Hadamard gates. Hence, the proposed ansatz is not problem-specific and can be scaled up as the problem size increases.

A considerable difference between the two methods is, that the QAOA-E is significantly shallower compared to the QAOA-M. Most of the classical optimization algorithms that utilize analytic gradients rely heavily on back-propagation. Therefore, having a variational algorithm that has a larger depth, can increase numerical instabilities and lead to a worse training accuracy. Moreover, having a deeper variational ansatz can lead to a *vanishing gradient problem*, which reduces the importance of the parameters that act during earlier steps of the model, which can also cause barren plateaus on the learning problem [74, 75]. Therefore, for the classical optimizers that rely on back-propagation, QAOA-E presents a more feasible variational ansatz.

Moreover, having a shallower circuit also implies being less vulnerable to noise, which will be particularly crucial when the ansatz is implemented on real quantum hardware. It is also worthy to mention that QAOA-M only utilizes 1 and 2-local operations, while QAOA-E comprises unitaries with higher localities. Nevertheless, there are efficient decompositions of such multi-controlled-X gates [3, 76], into polynomial number of 1 and 2-local operations.

However, one important factor in why QAOA-E is able to create an ansatz for projecting the statevector successfully onto the feasible subspace while having a significantly shallower circuit, is that it utilizes ancillary qubits while forming the latent space representation. Furthermore, the number of ancillary qubits scales polynomially, as the problem size, or the number of QAOA iteration cycles increases. Compared to the QAOA-E, the QAOA-M does not use any ancillary qubits, which can be particularly favoured when there are limitations on the execution device, either a real quantum hardware or a simulation environment. Furthermore, the systems that include less number of qubits are also more resilient to noise.

It is also worth mentioning that, the computational limits are reached since the experimental results are obtained using a quantum circuit simulator environment. While utilizing a quantum circuit simulator, the states, and the operators, are basically represented as complex vectors and matrices. For instance, for the 4-graph setting, the cost Hamiltonian would be represented as a $2^{16} \times 2^{16}$ complex-valued matrix, which requires a memory allocation close to 70 GBs.

In order to remedy this shortcoming, a workaround is also implemented that is being used for both of the models. The cost Hamiltonian is normally created by calculating the Kronecker product of different Pauli string terms and then adding them together. However, since the cost Hamiltonian term only includes Pauli-Z operators, the final Hamiltonian is always going to be a diagonal operator, where the elements are either 1 or -1, multiplied with a coefficient corresponding to the edge weights. Therefore, the final operator will eventually be a sparse matrix, except for its diagonal terms, which are all non-zero terms. Hence, storing the operator as a sparse matrix increases the efficiency of the overall algorithm vastly. Using the SciPy's sparse matrix representation [77] and the Pennylane's [56] integration of the sparse matrices with the measurement operators, makes it possible to run the circuit.

6 Discussion

In a nutshell, the QAOA-E aims to achieve a problem-independent ansatz for the constrained optimization problems, that can also be efficiently scaled up for larger tasks. Having implemented on small-scale examples, the model shows convincing results both in terms of learning success along with its efficiency in utilizing resources compared to its counterparts. The model is proven to be converging rapidly for an example model of a constrained optimization problem, namely the Travelling Salesman Problem.

Although the model is proven to be converging in a few iterations for smaller scales, there are still a few problems that the model faces that need to be addressed. The first problem arises, when the number of edges in the graph is not equal to an exact power of 2, such that

$$k = \log_2 |E|, \text{where, } k \notin \mathbb{Z}.$$

In such a setting, there are g number of computational basis states in the latent space, that represent a non-valid route, where $g = 2^{\lceil \log_2 |E| \rceil} - |E|$. For instance, for a complete graph with 7 vertices, 3152 states over 8192 possible states in the latent space, represent a non-valid solution. Even though there is a potential fix for that problem, by penalizing the non-valid routes that are successfully being encoded into the latent space. An implementation of the penalization could be done by relaxing the problem structure, by adding dummy edges with a particularly large weight, between the vertices that do not share a connection. Nevertheless, with this approach, the model fails to provide any generalized solution for distinguishing those particular states for any given graph. Another solution might be, neglecting the output during the training in the iterations where the prediction is a non-valid route. However, this heuristic might also cause over-fitting in the model, since it impedes full exploration of possible parameter configurations.

Furthermore, the ansatz only assumes graphs that are connected, which means the model does not utilize edge information on the graph and presumes that there is a connection between each vertex in the graph. This indicates that, for a graph that is not completely connected, the number of non-valid states in the latent space would be even more, since some of the routes that satisfy the constraints might include an edge that does not appear on the original graph.

For the mentioned problems in the encoding mechanism, the aim of the future work will focus on a better encoder ansatz, which only includes valid routes in its feasible subspace.

Moreover, even though the ansatz is eventually shallower compared to its counterparts, the depth of the circuit still grows exponentially as the problem scales up, which makes the model challenging to be implemented on a NISQ-era device.

Due to the limits on the computation on a simulation device, the method is unable to prove any supremacy over the classical methods on larger scales, similar to the other state-of-the-art approaches for solving constrained optimization problems using quantum algorithms. Even though the sparse matrix representation of the cost Hamiltonian makes the measurement possible in terms of tractability, the circuit simulation is still a cumbersome task as the number of qubits increases. One potential way of tackling this issue could be utilizing a different simulation technique. For instance, a tensor network based simulator could be very beneficial in terms of computability, since the operators in the ansatz are all single, or 2-qubit gates. However, with the state-of-the-art quantum software frameworks, optimizing over a tensor network for a complicated task remains to be a challenging issue.

All in all, even for the small toy examples, the variational method interleaved with an encoder is proven to converge. Furthermore, the toy model ansatz is an implementation for the recipe of a general n-vertex graph, proposing that the model can be scaled up for larger and more complex graphs, which makes it a viable method for solving constrained optimization problems.

List of Figures

List of Tables

Bibliography

[1] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.

[2] S. Hadfield, Z. Wang, E. G. Rieffel, B. O’Gorman, D. Venturelli, and R. Biswas, “Quantum approximate optimization with hard and soft constraints,” in *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, 2017, pp. 15–21.

[3] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, 2002.

[4] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[5] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, “Quantum computation by adiabatic evolution,” *arXiv preprint quant-ph/0001106*, 2000.

[6] P. I. Bunyk, E. M. Hoskinson, M. W. Johnson, E. Tolkacheva, F. Altomare, A. J. Berkley, R. Harris, J. P. Hilton, T. Lanting, A. J. Przybysz, *et al.*, “Architectural considerations in the design of a superconducting quantum annealing processor,” *IEEE Transactions on Applied Superconductivity*, vol. 24, no. 4, pp. 1–10, 2014.

[7] H. Neven, “„when can quantum annealing win?“” *Google Research Blog*, vol. 8, 2015.

[8] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven, “What is the computational value of finite-range tunneling?” *Physical Review X*, vol. 6, no. 3, p. 031 015, 2016.

[9] C. Vu, “Ibm makes quantum computing available on ibm cloud to accelerate innovation,” *IBM News Room*, 2016.

[10] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[11] P. A. M. Dirac, “A new notation for quantum mechanics,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, Cambridge University Press, vol. 35, 1939, pp. 416–418.

[12] A. Y. Kitaev, “Quantum computations: Algorithms and error correction,” *Russian Mathematical Surveys*, vol. 52, no. 6, p. 1191, 1997.

[13] R. P. Feynman *et al.*, “Simulating physics with computers,” *Int. j. Theor. phys*, vol. 21, no. 6/7, 1982.

[14] D. W. Berry, A. M. Childs, R. Cleve, R. Kothari, and R. D. Somma, “Simulating hamiltonian dynamics with a truncated taylor series,” *Physical review letters*, vol. 114, no. 9, p. 090 502, 2015.

[15] A. M. Childs and N. Wiebe, “Hamiltonian simulation using linear combinations of unitary operations,” *arXiv preprint arXiv:1202.5822*, 2012.

[16] M. Suzuki, “General theory of fractal path integrals with applications to many-body theories and statistical physics,” *Journal of Mathematical Physics*, vol. 32, no. 2, pp. 400–407, 1991.

[17] D. Du and P. M. Pardalos, *Handbook of combinatorial optimization*. Springer Science & Business Media, 1998, vol. 4.

[18] S. Yakovlev, “Convex extensions in combinatorial optimization and their applications,” in *Optimization Methods and Applications*, Springer, 2017, pp. 567–584.

[19] S. Onn and U. G. Rothblum, “Convex combinatorial optimization,” *Discrete & Computational Geometry*, vol. 32, no. 4, pp. 549–566, 2004.

[20] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, “The unconstrained binary quadratic programming problem: A survey,” *Journal of combinatorial optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[21] A. Lucas, “Ising formulations of many np problems,” *Frontiers in physics*, p. 5, 2014.

[22] F. Glover, G. Kochenberger, and Y. Du, “A tutorial on formulating and using qubo models,” *arXiv preprint arXiv:1811.11538*, 2018.

[23] S. Jansen, M.-B. Ruskai, and R. Seiler, “Bounds for the adiabatic approximation with applications to quantum computation,” *Journal of Mathematical Physics*, vol. 48, no. 10, p. 102 111, 2007.

[24] B. W. Reichardt, “The quantum adiabatic optimization algorithm and local minima,” in *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC ’04, Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 502–510, ISBN: 1581138520. DOI: `10.1145/1007352.1007428`. [Online]. Available: `https://doi.org/10.1145/1007352.1007428`.

[25] N. G. Dickson, M. Johnson, M. Amin, R. Harris, F. Altomare, A. Berkley, P. Bunyk, J. Cai, E. Chapple, P. Chavez, *et al.*, "Thermally assisted quantum annealing of a 16-qubit problem," *Nature communications*, vol. 4, no. 1, pp. 1–6, 2013.

[26] *D-wave sys, i. (2016a). d-wave developer guide.*

[27] J. Choi and J. Kim, "A tutorial on quantum approximate optimization algorithm (qaoa): Fundamentals and applications," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, 2019, pp. 138–142.

[28] S. Hadfield, "On the representation of boolean and real functions as hamiltonians for quantum computing," *ACM Transactions on Quantum Computing*, vol. 2, no. 4, pp. 1–21, 2021.

[29] R. Shaydulin and S. M. Wild, "Exploiting symmetry reduces the cost of training qaoa," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–9, 2021.

[30] C. Lemarchal, "Lagrangian relaxation," in *Computational combinatorial optimization*, Springer, 2001, pp. 112–156.

[31] D. P. Bertsekas, "Projected newton methods for optimization problems with simple constraints," *SIAM Journal on control and Optimization*, vol. 20, no. 2, pp. 221–246, 1982.

[32] E. G. Birgin and J. M. Martinez, *Practical augmented Lagrangian methods for constrained optimization*. SIAM, 2014.

[33] M. Schluter and M. Gerdts, "The oracle penalty method," *Journal of Global Optimization*, vol. 47, no. 2, pp. 293–325, 2010.

[34] D. W. Coit, A. E. Smith, and D. M. Tate, "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 173–182, 1996.

[35] E. Rieffel, D. Venturelli, M. Do, I. Hen, and J. Frank, "Parametrized families of hard planning problems from phase transitions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, 2014.

[36] E. G. Rieffel, D. Venturelli, B. O Gorman, M. B. Do, E. M. Prystay, and V. N. Smelyanskiy, "A case study in programming a quantum annealer for hard operational planning problems," *Quantum Information Processing*, vol. 14, no. 1, pp. 1–36, 2015.

[37] I. Hen and M. S. Sarandy, "Driver hamiltonians for constrained optimization in quantum annealing," *Physical Review A*, vol. 93, no. 6, p. 062 312, 2016.

[38] I. Hen and F. M. Spedalieri, “Quantum annealing for constrained optimization,” *Physical Review Applied*, vol. 5, no. 3, p. 034 007, 2016.

[39] V. Choi, “Minor-embedding in adiabatic quantum computation: I. the parameter setting problem,” *Quantum Information Processing*, vol. 7, no. 5, pp. 193–209, 2008.

[40] S. Hadfield, Z. Wang, B. O’gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, “From the quantum approximate optimization algorithm to a quantum alternating operator ansatz,” *Algorithms*, vol. 12, no. 2, p. 34, 2019.

[41] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, “Efficient quantum algorithm for solving travelling salesman problem: An ibm quantum experience,” *arXiv preprint arXiv:1805.10928*, 2018.

[42] M. Schuld and N. Killoran, “Quantum machine learning in feature hilbert spaces,” *Physical review letters*, vol. 122, no. 4, p. 040 504, 2019.

[43] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, *et al.*, “Variational quantum algorithms,” *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[44] N. Nguyen and K.-C. Chen, “Quantum embedding search for quantum machine learning,” *IEEE Access*, 2022.

[45] M. Schuld, “Supervised quantum machine learning models are kernel methods,” *arXiv preprint arXiv:2101.11020*, 2021.

[46] C.-K. Li, R. Roberts, and X. Yin, “Decomposition of unitary matrices and quantum gates,” *International Journal of Quantum Information*, vol. 11, no. 01, p. 1 350 015, 2013.

[47] A. Daskin and S. Kais, “Decomposition of unitary matrices for finding quantum circuits: Application to molecular hamiltonians,” *The Journal of chemical physics*, vol. 134, no. 14, p. 144 112, 2011.

[48] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash, “Learning to optimize variational quantum circuits to solve combinatorial problems,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 2367–2375.

[49] S.-G. Hwang, “Cauchy’s interlace theorem for eigenvalues of hermitian matrices,” *The American Mathematical Monthly*, vol. 111, no. 2, pp. 157–159, 2004.

[50] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature communications*, vol. 5, no. 1, pp. 1–7, 2014.

[51] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” *New Journal of Physics*, vol. 18, no. 2, p. 023 023, 2016.

[52] O. Kramer, D. E. Ciaurri, and S. Koziel, “Derivative-free optimization,” in *Computational optimization, methods and algorithms*, Springer, 2011, pp. 61–83.

[53] J. Larson, M. Menickelly, and S. M. Wild, “Derivative-free optimization methods,” *Acta Numerica*, vol. 28, pp. 287–404, 2019.

[54] J. A. Nelder and R. Mead, “A simplex method for function minimization,” *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[55] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash, “Reinforcement-learning-based variational quantum circuits optimization for combinatorial problems,” *arXiv preprint arXiv:1911.04574*, 2019.

[56] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, *et al.*, “Pennylane: Automatic differentiation of hybrid quantum-classical computations,” *arXiv preprint arXiv:1811.04968*, 2018.

[57] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, “Evaluating analytic gradients on quantum hardware,” *Physical Review A*, vol. 99, no. 3, p. 032 331, 2019.

[58] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, “Quantum circuit learning,” *Physical Review A*, vol. 98, no. 3, p. 032 309, 2018.

[59] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, “Circuit-centric quantum classifiers,” *Physical Review A*, vol. 101, no. 3, p. 032 308, 2020.

[60] J. C. Spall, “An overview of the simultaneous perturbation method for efficient optimization,” *Johns Hopkins apl technical digest*, vol. 19, no. 4, pp. 482–492, 1998.

[61] J. Spall, “Implementation of the simultaneous perturbation algorithm for stochastic optimization,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 817–823, 1998. DOI: `10.1109/7.705889`.

[62] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: A survey,” *Journal of Marchine Learning Research*, vol. 18, pp. 1–43, 2018.

[63] N. Ketkar, “Stochastic gradient descent,” in *Deep learning with Python*, Springer, 2017, pp. 113–132.

[64] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[65] W. Espelage, F. Gurski, and E. Wanke, "How to solve np-hard graph problems on clique-width bounded graphs in polynomial time," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 2001, pp. 117–128.

[66] C. Berge, "Graphs and hypergraphs," 1973.

[67] G. G. Langdon Jr, *Computer Design*. Computeach Press, Incorporated, 1982.

[68] J. Romero, J. P. Olson, and A. Aspuru-Guzik, "Quantum autoencoders for efficient compression of quantum data," *Quantum Science and Technology*, vol. 2, no. 4, p. 045 001, 2017.

[69] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 4. Springer, 2006, vol. 4.

[70] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[72] G. Hinton. "Coursera neural networks for machine learning lecture 6." (2018), [Online]. Available: `https://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf` (visited on 10/05/2022).

[73] A. Mayer, "Noisyopt: A python library for optimizing noisy functions.," *J. Open Source Softw.*, vol. 2, no. 13, p. 258, 2017.

[74] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[75] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature communications*, vol. 9, no. 1, pp. 1–6, 2018.

[76] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[77] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: `10.1038/s41592-019-0686-2`.