TDK-dolgozat

Dávid Sipos

University of Szeged

Institute of Informatics

Department of Computational Optimization

Quantum Optimization Suite for Airline Crew Pairing

Written by:

Dávid Sipos
Computer Science
MSc. II. year

Supervisors:

András Czégel
Optimization Expert
Lufthansa Systems

Boglárka G.-Tóth
Senior Research Fellow
Department of Computational Optimization

Absztrakt

A Reptéri Beosztás Probléma a járatok tervezéséhez kapcsolodó problémák egyik legnehezebbike, hisz a járatok számának növekedésével kombinatorikus robbanás léphet fel. Ugyanakkor a probléma hatékony megoldása, vagy akár csak jó közelítések gyors megtalálása hatalmas költségektől mentesítheti a légitársaságokat, hiszen a legtöbb költség a járatok üzemeltetéséhez szükséges munkaerő fenntartásával jár.

Napjainkban egyre nő az érdeklődés a kvantumszámítógépek által nyújtott potenciális számítási előny megvalósítása iránt, és ezek a törekvések már a légitársaságok optimalizálási problémáit is érintik [1]. A kvantum optimalizálás területén a Variációs Kvantum Algoritmusok (VQA) kapták a legtöbb figyelmet a kutatóktól, melyekben a számítás egy részét kvantumszámítógép végzi, amit egy klasszikus számítógép felügyel, vezényel a kvantum algoritmus különféle paramétereinek módosításával.

A Kvantum Közelítő Optimalizáló Algoritmus (QAOA) bevezetése óta annak sok változata, módosítása született elméleti és kísérleti eredményekkel együtt. A különböző változatok célja, hogy sikerüljön belátni a kvantum számítógépek előnyét a klasszikus számítógépekkel szemben, akár már zajos, kis-méretű eszközökön is. Az algoritmus változatai több ízben javasolnak módosításokat, többek közt a kvantum áramkör szerkezetében, a használt globális optimalizáló algoritmusban és annak konfigurációjában.

A dolgozatban bemutatunk egy Kvantum Optimalizációs Szoftvercsomagot, melynek célja a különböző algoritmusok ellenőrzése, kiértékelése és kísérletek végrehajtása mind ipari szakemberek és kutatók által, többek között a Reptéri Beosztás Probléma különböző példányain. Úgy véljük, hogy ez a probléma kitűnő mérceként szolgálhat, mind a mostani kisebb és zajosabb kvantum számítógépeken, mind a jövő hibajavított gépein, és ipari fontossága miatt hatékony bizonyítékként szolgálhat a kvantumszámítógépek hasznára.

Könnyen bővíthetőségének köszönhetően a csomag tartalmazza a Kvantum Közelítő Optimalizáló Algoritmus (QAOA) különböző korszerű változatait, mint például az XQAOA és a QAOA+ implementációit. Ezek futtatásához alapvető integrációt is tartalmaz a szoftver népszerű szolgáltatókkal, mint az Amazon és IBM, lehetővé téve akár valódi kvantum számítógepek használatát. A kvantum szubrutinok mellett, az annak paramétereit állító globális optimalizáló algoritmusok közül is több népszerű módszer is elérhető a programcsomagban.

Abstract

The Airline Crew Pairing problem is one of the hardest subproblems associated with airline planning since the number of flights can lead to a combinatorial explosion. Nevertheless, efficiently solving the problem, or even quickly obtaining an approximate solution can alleviate significant costs burdening airlines as most costs incurred come from labor costs.

Nowadays there is a heightened interest in realizing the potential advantage promised by quantum computing. These efforts have even reached some optimization problems that plague airlines [1]. In the field of quantum optimization, Variational Quantum Algorithms have garnered the attention of many researchers, where a quantum subroutine is aided by a classical optimization algorithm.

Since the proposition of the Quantum Approximate Optimization Algorithm, many variants of it have been introduced along with countless theoretical and experimental results regarding the possibility of quantum advantage using Noisy Intermediate-Scale Quantum hardware and beyond. These variants can alter many aspects of the algorithm from the classical optimization protocol used and its initialization, through the structure of the quantum circuit itself.

We introduce a Quantum Optimization Suite to aid with testing, evaluating, and experimenting with different Variational Quantum Algorithms in both industrial and research settings to solve, among many others, the Airline Crew Pairing problem. We believe that this problem can serve as a good benchmark both for the quantum computers of today and those of the future and due to its industrial importance can serve as proof of the utility of quantum computers.

We include state-of-the-art variants of the QAOA such as xQAOA and QAOA+ along with multiple global optimization algorithms. Necessary integrations to quantum computing providers such as IBM, and Amazon are also part of the suite, allowing testing on real quantum hardware.

Contents

Chapter 1

Introduction

Our goal is to create a framework that streamlines the process of implementation, testing and evaluation of different quantum optimization algorithms for solving the Airline Crew Pairing problem.

Optimizations of the labor force employed by airlines can have significant financial ramifications, making Airline Crew Scheduling a highly relevant problem in industrial settings. The Airline Crew Pairing problem is one of the hardest problems in airline planning as even a modest number of flights can result in exponentially many pairings. Due to this, we believe that the Airline Crew Pairing problem can serve as a good benchmark for experimenting with different state-of-the-art quantum optimization techniques. With smaller hand-crafted problem instances the noisy intermediate-scale quantum devices of today can be examined. As both algorithms and hardware develop, larger instances of the Airline Crew Pairing problem will continue to be a difficult task marking the progress made in quantum optimization. When the solvable problem instances reach industrial scales, quantum optimization will undoubtedly save large sums of money and increase customer satisfaction in the airline industry and hopefully many others.

With the introduction of the optimization suite, we aim to facilitate this journey from exploring circuit constructions and optimization strategies to solving problems plaguing many industries. As such the suite defines an optimization pipeline with interchangeable steps to quickly prepare and evaluate different quantum subroutines. While we chose the Airline Crew Pairing problem as our example, the suite is easy to extend to solve new problems. The development of new and exciting quantum hardware solutions seems to be rapidly accelerating, so one of our goals naturally was to allow easy integration with new quantum providers. In time, we aim to integrate more platforms, problems, classical optimization algorithms, and circuit designs into the suite with the hope of creating an easy-to-use and up-to-date system that can help researchers and industry professionals alike.

First we introduce the industrial problem, then give a brief overview of quantum optimization.

1.1. Airline Crew Scheduling

In the airline industry, a significant source of the costs of operations are labor costs [2, 3] associated with operating the airliners. As such, significant resources are invested into optimizing the costs of airline crews while complying with different labor regulations. Due to the complex nature of crew planning, the problem is often considered as a sequence of subproblems, each with its own set of challenges and possibilities for optimization. Airline Crew Scheduling is one such subproblem where the goal is to find crew schedules that cover all scheduled flights while adhering to regulations, minimizing costs, and taking into account crew preferences. Since Airline Crew Scheduling is itself a large and complex problem, it is often split into two subproblems, Airline Crew Pairing and Airline Crew Assignment (or Rostering).

1.1.1. Airline Crew Pairing

In the Airline Crew Pairing (ACP) subproblem, a sequence of flights is called a *pairing* if the home base from which the first flight departs is the same as the one at which the last flight arrives, and it meets regulatory requirements and constraints placed by airlines. Here crews are only considered in an abstract sense, as this subproblem aims to find pairings such that every flight is a part of exactly one pairing, with the total cost of the pairings being minimal. As such, considerations for individual crew members are not present in this subproblem.

1.1.2. Airline Crew Assignment

Starting from the pairings resulting from the ACP problem, Airline Crew Assignment is concerned with assigning them to crew members, taking into account crew preferences (such as a crew member disliking pairings that start early in the day) and rules imposed by the airline (such as the need for a Japanese speaker on a flight) and authorities (such as compulsory rest times) to maximize robustness and personnel satisfaction.

We focus on the Airline Crew Pairing problem with hopes for cost reductions, exploring how we can evaluate advances in quantum optimization applied to this problem domain.

1.2. Quantum optimization

Quantum computers, originally envisioned as devices that harness the quantum-mechanical nature of our reality to allow the simulation of quantum mechanical systems too complex for a classical computer to handle, have seen widespread speculation of their potential applications to classically hard computational problems. After the definition of the universal quantum computer by Deutsch [4] and the early promises of algorithms such as those of Grover [5] and Shor [6], many large companies have invested significant resources in the development of quantum computing systems. In today's Noisy Intermediate-Scale Quantum (NISQ) era [7] with quantum computers only possessing qubits in the range of hundreds to low-thousands and fully fault-tolerant systems far on the horizon, many have turned to hybrid quantum-classical algorithms [8] where part of the computation is offloaded to classical computers.

The idea of such hybrid algorithms also known as Variational Quantum Algorithms (VQA) was introduced with the Variational Quantum Eigensolver (VQE) [9] and later the Quantum Approximate Optimization Algorithm (QAOA) [10] and its many variants [11]. Such hybrid algorithms have shown promising results among many fields in machine learning [12, 13], quantum-chemistry and physics [14, 15, 16, 17].

VQAs work by classically optimizing a cost function where the value of the cost function is approximated using a quantum subroutine. The quantum subroutine is a parametric quantum circuit that prepares some quantum state from an initial state in accordance with a set of parameters provided by the classical optimizer. Measuring the resulting quantum state is akin to sampling a probability distribution induced by the state. After multiple measurements, we can obtain an estimate for the distribution which is then mapped to the approximate value of the cost function described by the circuit at the given parameters. In a sense, we can look at this as a global optimization problem where, given a black-box function, we are trying to find an optimal set of parameters that minimizes the function.

To solve a problem using VQAs we need to define a cost function that we can map to a quantum circuit. The most important feature of the cost function is that an optimal solution to the problem must map to the quantum state, prepared by the circuit, with the lowest associated cost value. For any single problem, many different circuit designs can exist, which are not necessarily problem-specific. These circuit designs are referred to as *ansätze* (singular ansatz), "educated guesses" about the best way to map the problem to a quantum circuit and in a way are akin to the architectural choices in building a neural network. Different ansätze can provide different benefits such as reducing the Hilbert-space explored by the optimizer or altering it such that infeasible solutions cannot appear as measurement outcomes. Other ansätze may try to reduce the depth of the quantum circuit or adapt to the hardware of the quantum computer running the subroutine.

Chapter 2

Problem Setup

2.1. Definitions

Here we define a couple of terms used when discussing the ACP problem. These definitions are simplified analogs to those enshrined in law, making it easier to discuss the problem in a more theoretical context.

- *Flight leg* A flight from a departure airport to an arrival airport.
- *Duty* A valid sequence of a single day's worth of flight legs. A sequence of legs is valid if it conforms to the considered regulations and additional constraints some of which are discussed later in Section 2.4.
- *Pairing* A valid sequence of duties centered around a home base. As duties themselves are sequences of legs, a pairing can be thought of as a sequence of flight legs that depart from and ultimately arrive at the same airport. A sequence of duties is a valid pairing if the duties themselves are valid and the sequence conforms to the considered regulations and additional constraints some of which are also discussed in Section 2.4.

2.2. Methodology

We introduce an optimization suite designed to evaluate VQAs and demonstrate its use through a couple of experiments on the Minimum Cost Exact Cover and ACP problems. To facilitate this we implement necessary functionalities to use data sampled from [18] and a hand-crafted dataset using the same format. In the case of the ACP problem, to evaluate the different quantum algorithms we first generate the set of valid pairings classically then using the Variational Quantum Algorithm of interest we search for a subset of the valid pairings that each contains every flight leg exactly once while minimizing the cost. This is easily integrated with the suite through plugins, easily swappable pieces of the optimization pipeline, which allow for a thorough modularization of the process.

2.3. Pairing generation

To generate the pairings first we take the flight legs of each day and generate the valid duties from them. In the sequence, consecutive flight legs must come after one another in time, that is the departure time of the latter leg must be after the arrival time of the former. The arrival airport of the former must also match the departure airport of the latter. Additional validation rules may be applied.

With the multiple duties for each day generated, these are in turn joined up into pairings. Pairings must start and end at the same airport, the home base of the crew. In our experiments pairings must not contain other pairings, this requirement is naturally satisfied in the case of a singular home base. Here, just like with duties, additional validation rules are applied.

2.4. Rules

In our experiments we use a ruleset inspired by those that can be found in [18] and may appear in real-world instances of ACP. The rules themselves are parametric to make it easier to match real-life scenarios, needing only to adjust the parameters.

- The rule `max duties` ensures that each pairing can only contain a certain number of duties at most, by default 5.
- The rule `max flights` defines the number of flights a duty can contain at most, by default 4.
- The rule `min rest` guarantees that between every duty a certain number of hours (9.5 by default) must pass, allowing crews to rest.
- The `min connect` rule defines a minimum connection time between flight legs in a duty measured in minutes, 30 by default.
- The `max pairing duration` rule ensures that no pairing can last longer than the given number of days, 4 by default.
- The `max work time` rule sets a cap on the hours worked by crews in each duty, 8 by default.
- Finally, the `max duration duty time` caps the duty time, that is the time between the first departure and last arrival of a duty, to the given number of hours, by default 12.

2.5. Cost model

We employ a simplified cost model. We assign a cost of $1000$$ to every day of a pairing and an additional $300$$ for every night when crew accommodations need to be arranged, which is when duties end at a non-home base.

Chapter 3

Introduction to Quantum Computing

3.1. Quantum State

In Dirac-notation (see Appendix A.2) states of a system with two mutually exclusive states can be identified with the vectors $|0\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ and $|1\rangle = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$. A quantum bit (or qubit) is a superposition of the two states, each with a complex amplitude: $\alpha_0 |0\rangle + \alpha_1 |1\rangle$. The states $|0\rangle$ and $|1\rangle$ form an orthonormal basis and induce a 2-dimensional Hilbert-space[1]. An N- and an M-dimensional vector space $\mathcal{H}_1$ and $\mathcal{H}_2$ can be combined into an NM-dimensional vector space using their tensor product (see Appendix A.1.7) $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ spanned by:

$$\{\, |i\rangle \otimes |j\rangle \mid i \in \{0, \ldots, N-1\}, j \in \{0, \ldots, M-1\} \,\}. \tag{3.1}$$

In a system with N mutually exclusive states $|0\rangle\,|1\rangle \ldots |N-1\rangle$, a *pure quantum state* $|\Phi\rangle$ is a superposition of classical states $|i\rangle$ with each having a complex amplitude α_i such that:

$$|\Phi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle \qquad\qquad \sum_{i=0}^{N-1} |\alpha_i|^2 = 1. \tag{3.2}$$

The pure state $|\Phi\rangle$ can also be written as a unit vector in the N-dimensional Hilbert-space induced by the orthonormal-basis formed by the classical states $|0\rangle\,|1\rangle \ldots |N-1\rangle$ using the amplitudes α_i:

$$|\Phi\rangle = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{N-1} \end{pmatrix}. \tag{3.3}$$

In contrast, a *mixed quantum state* can be thought of as a probability distribution over pure quantum states and is more aptly described by the *density matrix*

$$\rho = \sum_{i=1}^{k} p_i |\Phi_i\rangle \langle\Phi_i|\,, \tag{3.4}$$

[1]A vector space with an inner product

where $\Phi_1, \ldots, \Phi_k$ are pure states and $p_1, \ldots, p_k$ are their respective probabilities. A pure state can also be represented using a density matrix, in which case $\rho = |\Phi\rangle\langle\Phi|$.

Sticking to pure states, in quantum mechanics we can change the state $|\Phi\rangle$ to some other state

$$|\Psi\rangle = \beta_0 |0\rangle \beta_1 |1\rangle \ldots \beta_{N-1} |N-1\rangle \tag{3.5}$$

by applying linear operations that preserve the norm of $|\Phi\rangle$. This mathematically is equivalent to multiplying the vector $|\Phi\rangle$ with an $N \times N$ complex unitary matrix U:

$$U \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{N-1} \end{pmatrix}, \tag{3.6}$$

which is equivalent to applying a rotation to $|\Psi\rangle$. Some of the most fundamental of such linear operations are described by the *Pauli matrices*:

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Every 2×2 matrix can be written as a linear combination of the Pauli matrices and any $2^n \times 2^n$ matrix can be written as a linear combination of the 4^n n-fold tensor products of the Pauli matrices.

3.2. Measurement

Measuring the state $|\Phi\rangle$ induces a probability distribution over k possible measurement outcomes. The probability of measuring outcome i can be computed as $\mathrm{Tr}(M_i |\Phi\rangle\langle\Phi|) = \|M_i |\Phi\rangle\|^2$, where $M_1, \ldots, M_k$ are positive-semidefinite matrices that sum to identity and $\mathrm{Tr}(A)$ is the trace of matrix A, that is, the sum of its diagonal elements. The set $\{M_i\}$, characterizing the measurement, is called a *Positive-Operator-Valued Measure* (POVM).

A *Projection-Valued Measure* (PVM) is a special case of a POVM where M_i are pairwise-orthogonal projections, that is $M_i = M_i^2$ and $M_i = M_i^*$, where M_i^* is the conjugate-transpose of M_i. Using projections, the pure state $|\Phi\rangle \in \mathcal{H}$ can be uniquely decomposed into a sum of orthogonal states $|\Phi\rangle = \sum_{i=1}^{k} |\Phi_i\rangle$ where $|\Phi_i\rangle = M_i |\Phi\rangle$ are the projections of the state $|\Phi\rangle$ onto an orthogonal subspace $\mathcal{H}_i \subseteq \mathcal{H}$.

After measurement the state will collapse to the post-measurement state $|\Phi_i\rangle / \| |\Phi_i\rangle \|$.

A special case of PVM is when the projections are formed from the orthonormal basis states of the Hilbert-space, $M_i = |i\rangle\langle i|$ for $i = 0, \ldots, N$. A measurement characterized by such a projector is referred to as *a measurement in the computational basis*. As per Born's rule, measurement in the computational basis yields the outcome i with probability $|\alpha_i|^2$ and collapses the state to $|i\rangle$.

Given a PVM with projections $M_1, \ldots, M_k$ and outcomes $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the matrix $O = \sum_{i=1}^{k} \lambda_i M_i$ is referred to as an *observable*. Observables are self-adjoint linear operators that in the real world correspond to some measurable quantity of a physical system. The possible outcomes for the measurement are given by the eigenvalues $\lambda_1, \ldots, \lambda_k$ of the operator. Mathematically, $\sum_{i=1}^{k} \lambda_i M_i$ is the spectral-decomposition of the operator O. Since the probability of outcome λ_i is $\| M_i \left|\Phi\right\rangle \|^2$, we can calculate the expected value of the measurement to be $\mathrm{Tr}(O \left|\Phi\right\rangle \left\langle\Phi\right|)$.

3.3. Hamiltonian operator

The *Hamiltonian* is the linear operator $\mathcal{H}$ associated with the observable representing the total energy of a physical system. The energy of the system is equal to the expected value $\left\langle\psi\right| \mathcal{H} \left|\psi\right\rangle$. The Hamiltonian describes the evolution of the system through the Schrödinger equation, which is a linear differential equation,

$$i\hbar \frac{d\left|\psi(t)\right\rangle}{dt} = \mathcal{H} \left|\psi(t)\right\rangle .$$

According to the equation, with letting $\hbar = 1$, starting from an initial state $\left|\psi(0)\right\rangle$, the state at time t is

$$\left|\psi(t)\right\rangle = U \left|\psi(0)\right\rangle ,$$

where U is the unitary operator $e^{-i\mathcal{H}t}$.

3.4. Entanglement

An important phenomenon in quantum mechanics used by many fundamental algorithms is that of *entanglement*. This refers to a correlation between different qubits. Mathematically for quantum states $\left|\Phi\right\rangle \in \mathcal{H}_1$ and $\left|\Psi\right\rangle \in \mathcal{H}_2$, the state $\left|\Phi\right\rangle \left|\Psi\right\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ is called a *product state*. A quantum state that cannot be written as a product state but only as a sum of product states is called an *entangled state*.

For example the quantum state

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left|00\right\rangle + \frac{1}{\sqrt{2}} \left|11\right\rangle \tag{3.7}$$

is an entangled state and measuring the first qubit in the state $\left|0\right\rangle$ guarantees that the post-measurement state of the second qubit will also be $\left|0\right\rangle$.

The quantum state $|\Phi^+\rangle$ cannot be written as a product state like $|\Phi\rangle|\Psi\rangle$ since

$$\left|\Phi^+\right\rangle = |\Phi\rangle|\Psi\rangle = \begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix} \otimes \begin{pmatrix}\beta_0\\ \beta_1\end{pmatrix} = \begin{pmatrix}\alpha_0\beta_0\\ \alpha_0\beta_1\\ \alpha_1\beta_0\\ \alpha_1\beta_1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\1\end{pmatrix} \implies \begin{cases}\alpha_0\beta_0 = \frac{1}{\sqrt{2}}\\ \alpha_0\beta_1 = 0\\ \alpha_1\beta_0 = 0\\ \alpha_1\beta_1 = \frac{1}{\sqrt{2}}\end{cases}, \tag{3.8}$$

which cannot hold.

3.5. Interference

Another often exploited phenomenon of quantum mechanics is that of *interference*, where the complex amplitudes of states can amplify and cancel each other out. As an example we can look at the linear operator $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$ applied to the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$:

$$\begin{aligned} &H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}H|0\rangle + \frac{1}{\sqrt{2}}H|1\rangle\\ &= \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\\ &= \frac{1}{2}\left[\begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix}\right] + \frac{1}{2}\left[\begin{pmatrix}1\\0\end{pmatrix} - \begin{pmatrix}0\\1\end{pmatrix}\right]\\ &= \frac{1}{2}\Big(|0\rangle + |1\rangle\Big) + \frac{1}{2}\Big(|0\rangle - |1\rangle\Big)\\ &= |0\rangle, \end{aligned} \tag{3.9}$$

where the amplitudes of the $|1\rangle$ state sum to 0.

3.6. Quantum Circuits

Similarly to classical circuits, in the *circuit model of quantum computation* algorithms are constructed using gates, in this case quantum gates, which are unitary transformations acting on only a few qubits.

As in the case of classical gates, more complex operations can be implemented using a combination of quantum gates. Applying gates in parallel results in a unitary transformation that is the tensor product of the applied gates, while applying gates in sequence produces the unitary transformation described by the matrix product of the applied gates.

At the end of the algorithm, to obtain results we perform measurements, conventionally in the computational basis.

We recall some quantum gates that will be used in later parts of this text:

- The I gate or identity gate, implements the unitary transformation $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$.
- The Pauli-X, Pauli-Y, and Pauli-Z gates (X, Y, Z) implement the unitary transformations $\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}0&-i\\i&0\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}1&0\\0&-1\end{smallmatrix}\right)$ respectively.
- The H gate or Hadamard-gate, implements the unitary transformation $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$.
- The $R_A(\theta)$, $A \in X, Y, Z$, gate is defined as $\exp(-iA\theta)$.
- The $R_{ZZ}(\theta)$ gate, is defined as $\exp\left(-i\frac{\theta}{2} Z \otimes Z\right)$

When describing a quantum circuit, indexing of the different gates means that the gate is applied to the particular qubit(s) of the quantum system, for example, $R_{ZZ}(\theta)_{jj'}$ applies the R_{ZZ} gate to qubits j and j'. The unitary transformations implemented by gates configured as such can be constructed from the tensor- and matrix-products of the unitary transformations described above.

Chapter 4

Mathematical Model

In this section, we discuss how the ACP problem can be formulated as an instance of the Minimum Cost Exact Cover problem and in turn how that formulation can be solved using a Variational Quantum Algorithm.

4.1. Minimum Cost Exact Cover

The *Exact Cover* problem is concerned with finding a set S of subsets of a set A such that the elements of the sets in S contain each element of the original set exactly once.

Formally, given a collection $S = \{s_1, \dots, s_m\}$ of subsets of a set $A = \{a_1 \dots a_n\}$, an *exact cover* of A is a subcollection $S^* \subseteq S$ that satisfies the following:

$$s_i \cap s_j = \emptyset \quad \forall s_i, s_j \in S^*, i \neq j \tag{4.1}$$

$$\bigcup_{s_j \in S^*} s_j = A \tag{4.2}$$

An optimization version of the Exact Cover problem can be formulated as trying to find an exact cover using a minimal number of subsets.
Generalizing this further we get the *Minimum Cost Exact Cover* (MCEC) problem where each subset s_i has an associated cost c_i, and we call an exact cover *minimal* if the sum of the costs of the included subsets, $\sum_{s_j \in S^*} c_j$, is minimal. The optimization problem where our goal is to use a minimal number of subsets is a special case of the MCEC problem where every subset shares the same positive cost.

It is easy to see that with flight legs being the elements of set A, while valid pairings with their associated costs being the subsets in S, the ACP problem can be viewed as MCEC.

4.2. Quadratic Unconstrained Binary Optimization

Given a symmetric matrix $Q \in \mathbb{R}^{n \times n}$, finding a binary vector $x \in \{0, 1\}^n$ such that the term $x^\intercal Q x$ is minimized is often referred to as the *Quadratic Unconstrained Binary Optimization*

(QUBO) problem. it is a combinatorial optimization problem with great importance when solving problems using quantum optimization [19, 20, 21] due to their connection to the Ising model which we will discuss in Section 4.3.

We can rewrite the ACP problem formulated as MCEC into a QUBO problem by first noticing that the original MCEC formulation is equivalent to the following binary linear optimization problem:

$$\min \quad z = \sum_{j=1}^{m} c_j x_j \tag{4.3}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} b_{ij} x_j = 1 \qquad \forall i \in \{1, \dots, n\} \tag{4.4}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, \dots, m\} \tag{4.5}$$

where b_{ij} is 1 if subset $s_j \in S$ contains element $a_i \in A$, otherwise 0. The binary variables x_j ($j = 1, \dots, m$) determine whether a subset s_j is part of the exact cover ($x_j = 1$), or not ($x_j = 0$).

We can turn this into a QUBO problem by placing penalty terms in the objective that fulfill similar roles as the constraints. We will use the penalty term $\left(1 - \sum_{j=1}^{m} b_{ij} x_j\right)^2$ mentioned in Section 4.1 of [22]. It will be minimal (0) if all the constraints are satisfied. Accordingly, the reformulated objective function is

$$\min \quad z = D \sum_{i=1}^{n} \left(1 - \sum_{j=1}^{m} b_{ij} x_j\right)^2 + \sum_{j=1}^{m} c_j x_j, \tag{4.6}$$

where the first term penalizes violating the constraint, and the second term encodes the cost of a particular solution. Note that the penalty term is scaled by a factor $D > n \max_i\{c_i\}$ to make sure that *"it doesn't make sense"* to violate a constraint.

4.3. Constructing the physical system

The *Ising model* [23] introduced by Ernst Ising and Wilhelm Lenz is a mathematical model of ferromagnetism. In the model, discrete variables, with values of ± 1, are used to represent the magnetic moments of atomic spins. A *magnetic moment* is the strength and orientation of an object producing a magnetic field. In the case of elementary particles, such as electrons, *spin* refers to an intrinsic angular momentum of the particle. The spin of elementary particles induces a magnetic moment called a *nuclear magnetic moment*.

In the Ising model, we consider a set L of so-called lattice sites, each with a set of adjacent sites *(essentially a graph)*, forming a d-dimensional lattice. Each site $k \in L$ is associated with a spin variable $\omega_k \in \{-1, +1\}$, which represents the spin of that site. Between any two adjacent sites $i, j \in L$ there is an *interaction* J_{ij}, this is a real number that controls whether adjacent

spins should *align* (have the same value) or *anti-align* (have different values) or in the case of $J_{ij} = 0$, ω_i and ω_j can have arbitrary values meaning that the spins don't interact. Additionally, each site i is affected by an external magnetic field represented by the real-valued h_i, in the model it is desirable for spins to align with the external magnetic field as that lowers the energy of the physical system.
For a given spin configuration $\omega = \{\omega_k\}_{k \in L}$ the *energy* of the configuration is given by the Hamiltonian function:

$$H(\omega) = \sum_{\langle ij \rangle} J_{ij}\omega_i\omega_j + \sum_{j \in L} h_j\omega_j \tag{4.7}$$

Where $\langle ij \rangle$ indicates that i and j are adjacent. Note that conventionally the signs of both sums are flipped, we consider this version as it conforms better with the later description of QAOA.
One important object of study in physics is finding the *ground state* or the lowest energy spin configuration of a system. In the following, we will leverage the fact that quantum computers are universal simulators of quantum mechanical phenomena and that finding the lowest energy spin configuration in the Ising model is an NP-complete problem [24].
If we want to examine how the state of a system in the Ising model evolves over time, using a quantum computer, we need only to rewrite the Hamiltonian above using unitary operators.
To this end, in a system of m qubits we define the operator σ_j^z that applies the *Pauli-Z* operator $\sigma^z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ to the j-th qubit of the system and applies the identity operator $\mathbb{1} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ to the rest:

$$\sigma_j^z = \underbrace{\mathbb{1} \otimes \cdots \otimes \mathbb{1}}_{j-1} \otimes \sigma^z \otimes \underbrace{\mathbb{1} \otimes \cdots \otimes \mathbb{1}}_{m-j}. \tag{4.8}$$

In the case of the Ising model, we can simply replace the spin variable ω_j with the σ_j^z operator since the eigenvalues of the σ^z operator are ± 1 similar to the domain of the spin variables, and the expected value of the quantum system is equal to the energy of the physical system described by the model. The Hamiltonian we get is the following:

$$\mathcal{H}_C = \sum_{\substack{j,j'=1 \\ j<j'}}^{m} J_{jj'}\sigma_j^z\sigma_{j'}^z + \sum_{j=1}^{m} h_j\sigma_j^z. \tag{4.9}$$

The energy of the physical system described by the Ising model that is in the quantum state $|\phi\rangle$ is the expected value

$$\langle\phi| \mathcal{H}_C |\phi\rangle\,. \tag{4.10}$$

Our goal now is to reduce the QUBO formulation of the ACP problem to finding the ground state of a system described by the Hamiltonian in (4.9).

Based on [1], by replacing the variables $x_i \in \{0, 1\}$ with spin variables $\omega_i \in \{-1, 1\}$ in (4.6), the value of our objective function is equivalent to the energy of the spin configuration ω

given by the Hamiltonian function:

$$\mathcal{H}(\omega) = D\sum_{i=1}^{n}\left(1-\sum_{j=1}^{m} b_{ij}\frac{\omega_j+1}{2}\right)^2 + \sum_{j=1}^{m} c_j\frac{\omega_j+1}{2} \tag{4.11}$$

Notice that expanding the square and collecting terms (C) yields the following:

$$\begin{aligned}\mathcal{H}(\omega) = &\frac{D}{4}\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{j'=1}^{m} b_{ij}b_{ij'}\omega_j\omega_{j'}\\ &+\frac{D}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}\omega_j\left(\sum_{j'=1}^{m} b_{ij'}-2\right)+\sum_{j=1}^{m}\frac{\omega_j c_j}{2}\\ &+\frac{D}{4}\sum_{i=1}^{n}\left(\sum_{j=1}^{m} b_{ij}-2\right)^2+\sum_{j=1}^{m}\frac{c_j}{2}.\end{aligned} \tag{4.12}$$

By defining

$$J_{jj'} = \frac{D}{2}\sum_{i=1}^{n} b_{ij}b_{ij'} \qquad \forall j,j' \in \{1..m\}, \tag{4.13}$$

$$h_j = \frac{D}{2}\sum_{i=1}^{n} b_{ij}\left(\sum_{j'=1}^{m} b_{ij'}-2\right)+\frac{c_j}{2} \qquad \forall j \in \{1..m\}, \tag{4.14}$$

$$const = \frac{D}{4}\sum_{i=1}^{n}\left(\sum_{j=1}^{m} b_{ij}-2\right)^2+\sum_{j=1}^{m}\frac{c_j}{2} \tag{4.15}$$

the Hamiltonian function becomes

$$\mathcal{H}(\omega) = 1/2\sum_{j=1}^{m}\sum_{j'=1}^{m} J_{jj'}\omega_j\omega_{j'}+\sum_{j=1}^{m} h_j\omega_j + const. \tag{4.16}$$

Since J is symmetric and for all i, $J_{ii}\omega_i\omega_i = J_{ii}(\pm 1)^2 = J_{ii}$ we can further rewrite it as:

$$\mathcal{H}(\omega) = \sum_{\substack{j,j'=1\\ j<j'}}^{m} J_{jj'}\omega_j\omega_{j'}+\sum_{j=1}^{m} h_j\omega_j + const + \sum_{j=1}^{m} J_{jj}/2 \tag{4.17}$$

From this point, we will leave out the constant terms $const + \sum_{j=1}^{m} J_{jj}/2$ from the formulation as they do not affect which spin configurations have the lowest energy and can be added back later if we wish to know the exact energy of a given configuration. With this, we have arrived at a Hamiltonian[1] similar in form to that in (4.9) repeated here:

$$\mathcal{H}_C = \sum_{\substack{j,j'=1\\ j<j'}}^{m} J_{jj'}\sigma_j^z\sigma_{j'}^z+\sum_{j=1}^{m} h_j\sigma_j^z$$

[1]Note that this operator is exponential in size but sparse, and we can write it using polynomially many terms

Chapter 5

MCEC on Quantum Hardware

Since the Hamiltonian defined above describes a physical system for which a lowest energy state is analogous to a minimal valued solution to the MCEC problem, our goal now is to find a lowest energy state of the system.

5.1. Variational Quantum Algorithms

VQAs are a family of algorithms where a cost function, whose value is evaluated using quantum computation, is minimized using classical optimization strategies (Figure 1). Formally, as per [25], the cost function of a VQA can be written as

$$C(\theta) = f(\{\rho_k\}, \{O_k\}, U(\theta)), \tag{5.1}$$

where f is some function of a set of states $\{\rho_k\}$, a set of observables $\{O_k\}$ and the parametrised unitary transformation $U(\theta)$ with paramters θ. The goal of a VQA, in turn, is to find the parameters θ^*:

$$\theta^* = \operatorname*{argmin}_{\theta} C(\theta). \tag{5.2}$$

This is achieved by running a hybrid optimization procedure where a global optimizer seeks the optimal parameters of the cost function C classically, while the value of the cost function itself is derived using a quantum subroutine.

The quantum subroutine prepares and measures the quantum state $U(\theta)\left|\Phi\right\rangle$ for some initial state $\left|\Phi\right\rangle$, and unitary transformation $U(\theta)$, defined by the ansatz, several times to obtain an approximation of the probability distribution of the measurement outcomes. Based on the measurement results the value of the cost function is determined for the parameters θ, which the global optimizer uses to determine the next set of parameters.

The unitary transformation $U(\theta)$ is realized using a parametric quantum circuit, a quantum circuit that uses gates that can be parametrized, like the $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ and $R_{ZZ}(\theta)$ gates mentioned at the end of Section 3.6.

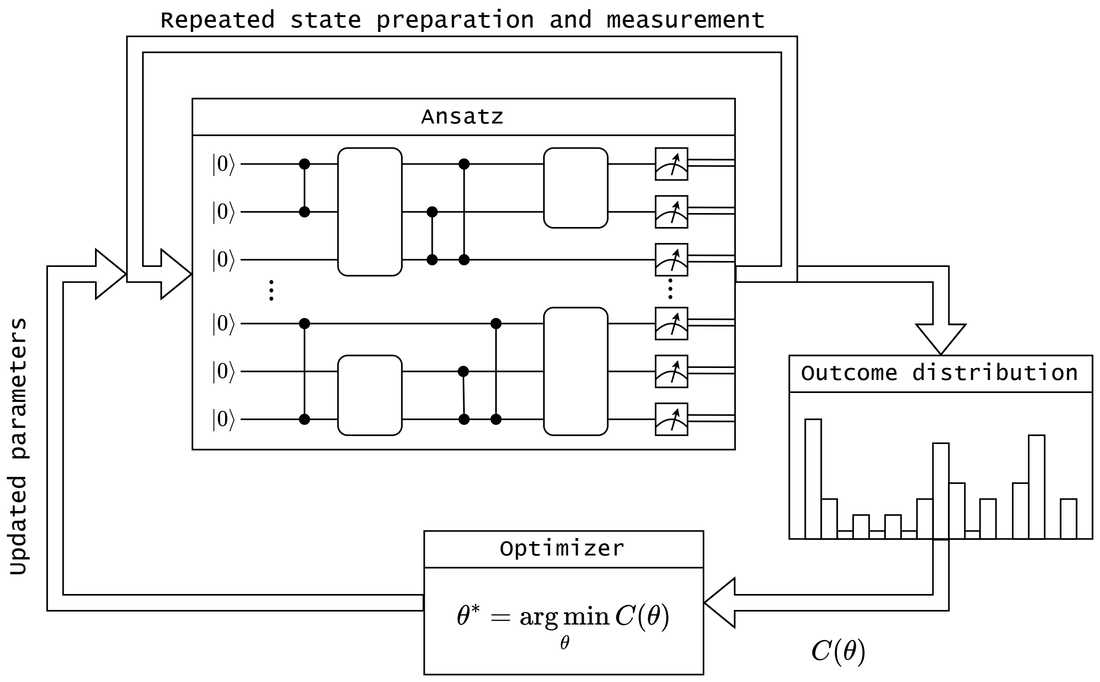
Repeated state preparation and measurement
Ansatz
$|0\rangle$
Updated parameters
Outcome distribution
Optimizer
$\theta^* = \arg\min_\theta C(\theta)$
$C(\theta)$

Figure 1: Schematic diagram of a VQA

5.2. QAOA

The Quantum Approximate Optimization Algorithm introduced in [10] produces approximate solutions to combinatorial problems using a discretized simulation of *Adiabatic Quantum Computation* [26, 27].

5.2.1. Finding a lowest energy state of a system

According to the *Adiabatic theorem* [28, 29], a physical system that is in a given state at a given time will remain in that state provided that the system is modified slowly enough and there is no ambiguity in the ordering of states based on their corresponding eigenvectors.

Thus given a Hamiltonian $\mathcal{H}_C$, it is possible to find its ground state with the help of a Hamiltonian - with the same degrees of freedom - $\mathcal{H}_M$, which does not commute with $\mathcal{H}_C$ and the ground state of which is known and easy to construct. By performing time-evolution on a system that is in the ground state of $\mathcal{H}_M$ for time T, the energy of which at time t is described by the time-dependent Hamiltonian

$$\mathcal{H}(t) = (1 - \frac{t}{T})\mathcal{H}_C + \frac{t}{T}\mathcal{H}_M, \tag{5.3}$$

we can transition to a ground state of $\mathcal{H}_C$, provided that the time it takes to transition between the two Hamiltonians approaches infinity. In the context of QAOA, $\mathcal{H}_C$ is called the cost Hamiltonian and $\mathcal{H}_M$ is called the mixer Hamiltonian.

5.2.2. Hamiltonian evolution

To implement time evolution under the time-dependent Hamiltonian above on a gate-based quantum computer we need to map it to unitary transformations that more straightforwardly map to quantum gates. To this end, we can approximate it with a product of Hamiltonians that each only act on a few qubits and only consist of commuting terms. As explained in [26], this approximation consists of discretizing the evolution time and applying the Trotter formula at each discrete time. The time-dependent Hamiltonian $\mathcal{H}(t)$ can be represented by the unitary time evolution operator $U(t, t_0)$ which describes the evolution of the system from time t_0 to time t. Plugging this into the Schrödinger-equation we get

$$i\frac{d}{dt}U(t,t_0) = \mathcal{H}(t)U(t,t_0). \tag{5.4}$$

This means that the state $|\psi(T)\rangle$ of the system at the end of the adiabatic evolution at time T is

$$|\psi(T)\rangle = U(T,0)\,|\psi(0)\rangle\,. \tag{5.5}$$

The unitary operator $U(T, 0)$ can be written as a product of M operators

$$U(T,0) = U(T,T-\delta)U(T-\delta,T-2\delta)\dots U(\delta,0), \tag{5.6}$$

where $\delta = T/M$. Still following [26], this is approximately equal to

$$e^{-i\delta\mathcal{H}(T-\delta)}e^{-i\delta\mathcal{H}(T-2\delta)}\dots e^{-i\delta\mathcal{H}(0)}. \tag{5.7}$$

With $\mathcal{H}(k\delta) = \beta\mathcal{H}_M + \gamma\mathcal{H}_C$, where $\gamma = 1-(k\delta/T)$ and $\beta = k\delta/T$ are real valued coefficients between 0 and 1. Applying the first-order Trotter-Suzuki decomposition [30, 31] for the number of Trotter-steps K gives

$$e^{-i\delta\mathcal{H}(k\delta)} \simeq (e^{-i\delta\beta\mathcal{H}_M/K}e^{-i\delta\gamma\mathcal{H}_C/K})^K. \tag{5.8}$$

The error term, $\mathcal{O}((k\delta)^2/K)$, vanishes quadratically as δ approaches 0 as M approaches infinity. Applying the Trotter-Suzuki decomposition to each term in (5.7) gives rise to the generalized formula introduced in [10]:

$$U_M(\beta_p)U_C(\gamma_p)...U_M(\beta_1)U_C(\gamma_1). \tag{5.9}$$

where γ_i and β_i are the times for which the system evolves under the two Hamiltonians represented by the unitary transformations $U_C(\gamma) = e^{-i\gamma\mathcal{H}_C}$ and $U_M(\beta) = e^{-i\beta\mathcal{H}_M}$.

5.2.3. Building the circuit

Now we will be showing the construction of the circuit implementing the following unitary transformation approximating adiabatic evolution:

$$U_M(\beta_p)U_C(\gamma_p)...U_M(\beta_1)U_C(\gamma_1). \tag{5.10}$$

Over time many alternative circuit designs have been formulated some of which we will explore in Section 5.3.

5.2.4. Building the cost layer

Recall that in (4.9) we encoded the objective function of the QUBO formulation of the MCEC problem in terms of the Ising model,

$$\mathcal{H}_C = \sum_{\substack{j,j'=1 \\ j<j'}}^{m} J_{jj'} \sigma_j^z \sigma_{j'}^z + \sum_{j=1}^{m} h_j \sigma_j^z.$$

Our goal is to implement the unitary transformation $U_C(\gamma) = \exp(-i\gamma\mathcal{H}_C)$ where γ is a real number representing the time for which the system evolves under the cost Hamiltonian $\mathcal{H}_C$.

$$\begin{aligned} U_C(\gamma) &= \exp\left(-i\gamma\mathcal{H}_C\right) \\ &= \exp\left(-i\gamma\left(\sum_{j'}^{m}\sum_{\substack{j=1 \\ j<j'}}^{m} J_{jj'}\sigma_j^z\sigma_{j'}^z + \sum_{j}^{m} h_j\sigma_j^z\right)\right) \\ &= \exp\left(-i\gamma\sum_{j'}^{m}\sum_{\substack{j=1 \\ j<j'}}^{m} J_{jj'}\sigma_j^z\sigma_{j'}^z\right)\exp\left(-i\gamma\sum_{j}^{m} h_j\sigma_j^z\right) \\ &= \prod_{j'}^{m}\prod_{\substack{j=1 \\ j<j'}}^{m} e^{-i\gamma J_{jj'}\sigma_j^z\sigma_{j'}^z}\prod_{j}^{m} e^{-i\gamma h_j\sigma_j^z} \end{aligned} \tag{5.11}$$

From this, the circuit itself is easy to construct:

$$U_C(\gamma) = \prod_{j'}^{m}\prod_{\substack{j=1 \\ j<j'}}^{m} R_{ZZ}(2\gamma J_{jj'})_{jj'}\prod_{j}^{m} R_Z(2\gamma h_j)_j \tag{5.12}$$

5.2.5. Building the mixer layer

Based on the original QAOA formulation the choice for the mixer Hamiltonian is:

$$\mathcal{H}_M = \sum_{j=1}^{m} \sigma_j^x \tag{5.13}$$

as it does not commute with $\mathcal{H}_C$ and its ground state $|+\rangle^{\otimes m}$, where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ is the equal superposition state, is easy to construct by applying the Hadamard gate to each qubit of the all-zero state, $H^{\otimes m}|0\rangle^{\otimes m} = |+\rangle^{\otimes m}$. We can implement the unitary transformation representing time evolution under this Hamiltonian for time β using the following circuit:

$$U_M(\beta) = \prod_{j}^{m} R_X(2\beta)_j. \tag{5.14}$$

5.2.6. Creating the quantum subroutine

As we have seen, finding an exact cover while minimizing cost is equivalent to minimizing an energy function, which in turn is equivalent to minimizing the expected value of a quantum system characterized by the Ising-Hamiltonian described in (4.9). Time evolution under the Ising-Hamiltonian can be implemented in the circuit model by approximating Adiabatic Quantum Computing via discretization followed by Trotterization. From this, we can obtain the variational circuit implementing the unitary transformation by repeatedly applying the cost and mixer layers in sequence p times, usually referred to as the *depth* of the algorithm:

$$U = U_M(\beta_p)U_C(\gamma_p)...U_M(\beta_1)U_C(\gamma_1). \tag{5.15}$$

Applying this transformation to the equal superposition $\left|+\right\rangle^{\otimes m}$ of states of m qubits

$$U\left|+\right\rangle^{\otimes m}, \tag{5.16}$$

the measured state corresponds directly to sampling a distribution over the solutions to the original problem, and the energy of the system in the state $U\left|+\right\rangle^{\otimes m}$ can be efficiently approximated classically.
By setting the $2p$ parameters appropriately we can increase the probability of measuring a minimal energy state of the system which should yield a binary vector that minimizes the cost of the exact cover while abiding by the constraints implicitly given by the penalty terms in the QUBO formulation.

5.2.7. Finding the optimal parameters

Since finding the $2p$ parameters maximizing the probability of finding a minimal energy state is not trivial, in QAOA a hybrid approach is used where a classical global optimizer is tasked with finding these variational parameters. The energy of the system as the function of the parameters can be approximated by calculating the empirical expectated value of the system using repeated executions of the quantum subroutine described above.

5.3. Alternative ansätze choices

Since the introduction of the QAOA, there have been many explorations into different circuit designs that utilize more parameters or different mixer Hamiltonians.

5.3.1. ma-QAOA

The *Multi-angle QAOA* (ma-QAOA) ansatz based on [32] assigns a variational parameter to each summand in the problem Hamiltonian $\mathcal{H}_C$ and mixing Hamiltonian $\mathcal{H}_M$, generalizing the original formulation where the cost and mixer Hamiltonians had one variational parameter each. Instead of each layer i, $i = 1 \dots p$, having 2 variational parameters, γ_i and β_i, we now have parameters $\hat{\gamma}_i \in \mathbb{R}^{m \times m}$, $\hat{\theta}_i \in \mathbb{R}^m$ and $\hat{\beta}_i \in \mathbb{R}^m$ such that:

$$U = U_M(\hat{\beta}_p)U_C(\hat{\gamma}_p, \hat{\theta}_p)...U_M(\hat{\beta}_1)U_C(\hat{\gamma}_1, \hat{\theta}_1), \tag{5.17}$$

where

$$U_C(\gamma, \theta) = \prod_{j'}^{m} \prod_{\substack{j=1 \\ j<j'}}^{m} R_{ZZ}(2\gamma_{jj'}J_{jj'})_{jj'} \prod_{j}^{m} R_Z(2\theta_j h_j)_j, \text{ and} \tag{5.18}$$

$$U_M(\beta) = \prod_{j}^{m} R_X(2\beta_j)_j. \tag{5.19}$$

5.3.2. QAOA+

The QAOA+ ansatz [33] augments the original QAOA ansatz by adding a problem-independent layer of parametric quantum circuits allowing it to outperform the original QAOA at shallower depths and even outperform the ma-QAOA in some cases. The circuit implements the unitary transformation

$$U = U_+(\hat{\nu}, \hat{\mu})U_M(\hat{\beta}_p)U_C(\hat{\gamma}_p)...U_M(\hat{\beta}_1)U_C(\hat{\gamma}_1), \tag{5.20}$$

where $\hat{\nu}, \hat{\mu} \in \mathbb{R}^m$ and U_+ is the problem independent unitary transformation,

$$U_+(\nu, \mu) = \prod_{j=2}^{m} R_{ZZ}(\nu_j)_{j,j-1} \prod_{j=1}^{m} R_X(\mu_j)_j. \tag{5.21}$$

5.3.3. XQAOA

The *eXpressive QAOA* (XQAOA) ansatz [34] is a generalization of ma-QAOA. It extends the mixer Hamiltonian by adding in Y-rotations, parametrized using a vector $\hat{\alpha}_i \in \mathbb{R}^m$ for each layer $i = 1 \dots p$ to increase the expressiveness of the ansatz by being able to express any computational-basis state with appropriate parametrization using the unitary

$$U = U_M(\hat{\alpha}_p, \hat{\beta}_p)U_C(\hat{\gamma}_p, \hat{\theta}_p)...U_M(\hat{\alpha}_1, \hat{\beta}_1)U_C(\hat{\gamma}_1, \hat{\theta}_1), \tag{5.22}$$

where U_C is similar to (5.17) used in ma-QAOA and

$$U_M(\beta, \alpha) = \prod_{j}^{m} R_Y(2\alpha_j)_j \prod_{j}^{m} R_X(2\beta_j)_j. \tag{5.23}$$

Table 5.1 summarizes the number of variational parameters used in these ansätze.

Table 5.1: Number of parameters for ansätze in Section 5.3 depending on algorithm depth p and number of variables m

Ansatz	Number of parameters
QAOA	$2p$
ma-QAOA	$p(m^2 + 2m)$
QAOA+	$2(p + m)$
XQAOA	$p(m^2 + 3m)$

5.4. Optimization strategies for finding optimal parameters

In this section, we highlight a few methods to optimize the parameters, which are compared in Section 7.1.

5.4.1. COBYLA

The *Constrained Optimization BY Linear Approximation* (COBYLA) method [35] is a popular gradient-free optimization technique for finding optimal QAOA parameters.

5.4.2. SPSA

The *Simultaneous Perturbation Stochastic Approximation* (SPSA) [36, 37] method is a gradient-free optimization algorithm that requires only two evaluations of the cost function C in every iteration. This can help with reducing the costs incurred by executing the quantum subroutines of VQAs. In the SPSA algorithm the kth estimate $\hat{\theta}_k \in \mathbb{R}^p$ of the optimal parameter $\theta^* \in \mathbb{R}^p$ is defined as

$$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k), \tag{5.24}$$

where $\hat{g}_k(x)$ is the approximate gradient of the cost function at point $x \in \mathbb{R}^p$ and the sequence a_k satisfies conditions seen in Section 3 of [36]. The gradient approximations are obtained by perturbing the point $\hat{\theta}_k$ by a vector of mutually independent zero-mean random variables $\delta_k \in \mathbb{R}^p$ to get

$$y_k^{(+)} = C(\hat{\theta}_k + c_k \delta_k) + \epsilon_k^{(+)} \qquad y_k^{(-)} = C(\hat{\theta}_k - c_k \delta_k) + \epsilon_k^{(-)}$$

where c_k are positive scalars and $\epsilon^{(\pm)}$ represent measurement noise. From this the gradient estimate $\hat{g}_k(\hat{\theta}_k)$ is given by

$$\hat{g}_k(\hat{\theta}_k) = \begin{pmatrix} \frac{y_k^{(+)} - y_k^{(-)}}{2c_k \delta_{k1}} \\ \vdots \\ \frac{y_k^{(+)} - y_k^{(-)}}{2c_k \delta_{kp}} \end{pmatrix}. \tag{5.25}$$

5.4.3. Genetic Algorithms

The optimization landscape of QAOA, especially at large depths, is rather complex, characterized by many local minima and barren plateaus, regions with vanishingly small gradients, which makes finding the optimal parameters increasingly difficult. The *Genetic Algorithm* (GA) proposed in [38] aims to alleviate these concerns by leveraging population-based metaheuristics to manage candidate solutions and starting from a random population search for the optimal solution by applying stochastic operators.

In genetic algorithms, inspired by Darwinian evolution, an initial population of candidate solutions evolve stochastically over generations. Candidate solutions (chromosomes) are strings of numbers (genes), which in our case are the ansatz parameters. Every generation, genetic operators are applied to the population to determine the population of the next generation. This involves selecting the best chromosomes from the population using a fitness function, which in our case means that a chromosome that achieves a lower energy state of the quantum system is deemed fitter.

Initial chromosomes are generated randomly and evaluated by querying the quantum computer. Afterward, in every iteration, parent chromosomes are chosen from groups of k based on their fitness. Offspring are generated from parent chromosomes via uniform crossover and gaussian mutation. Additionally, the best chromosome of a generation is moved to the next generation so as not to lose it.

The algorithms showcased in this section are all global optimization methods that have no information about the function optimized as such we cannot claim anything about their convergence. As such it is quite common to make assumptions about the behaviors of such algorithms using a vast array of experimental results. The optimization suite offers the necessary building blocks to conduct such experiments.

Chapter 6

Software

In this section, we describe some key architectural decisions regarding the optimization suite, along with ways to use the software in both research and industrial settings. To facilitate this, the suite includes a *Command Line Interface* (CLI) tool, alongside the definitions needed to start the suite inside a containerized environment, allowing easy and predictable deployment.

6.1. Architectural overview

The main design philosophy behind the optimization suite is extensibility, to this end, we provide a core set of primitives that together define a variational quantum optimization pipeline. This makes the pipeline highly customizable, allowing both researchers and industry professionals to quickly test and iterate on different approaches. For a quick overview of the architecture see Figure 2, which shows how the suite is structured with plugin registries for each part of the pipeline, a namespace with implemented plugins and separately installable platform integrations. We now take a brief look at the different plugin implementations.

6.1.1. Plugins

Plugins provide functionality as distinct pieces of the optimization pipeline. They are Python classes implementing plugin interfaces by inheriting from a plugin base class. Plugins are registered upon importing the module that defines them, which allows them to be loaded automatically if they are in the appropriate namespace. This allows third parties to contribute plugins to the appropriate namespace making them easy to install through a package manager (e.g. *pip* [39]). Additionally, plugins can be registered by simply importing their definitions in Python scripts or notebooks or by specifying a list of Python filenames from which to load plugins. The latter method is used, for example, by the CLI tool to allow it to load custom plugins by adding the paths to their source files in the configuration.

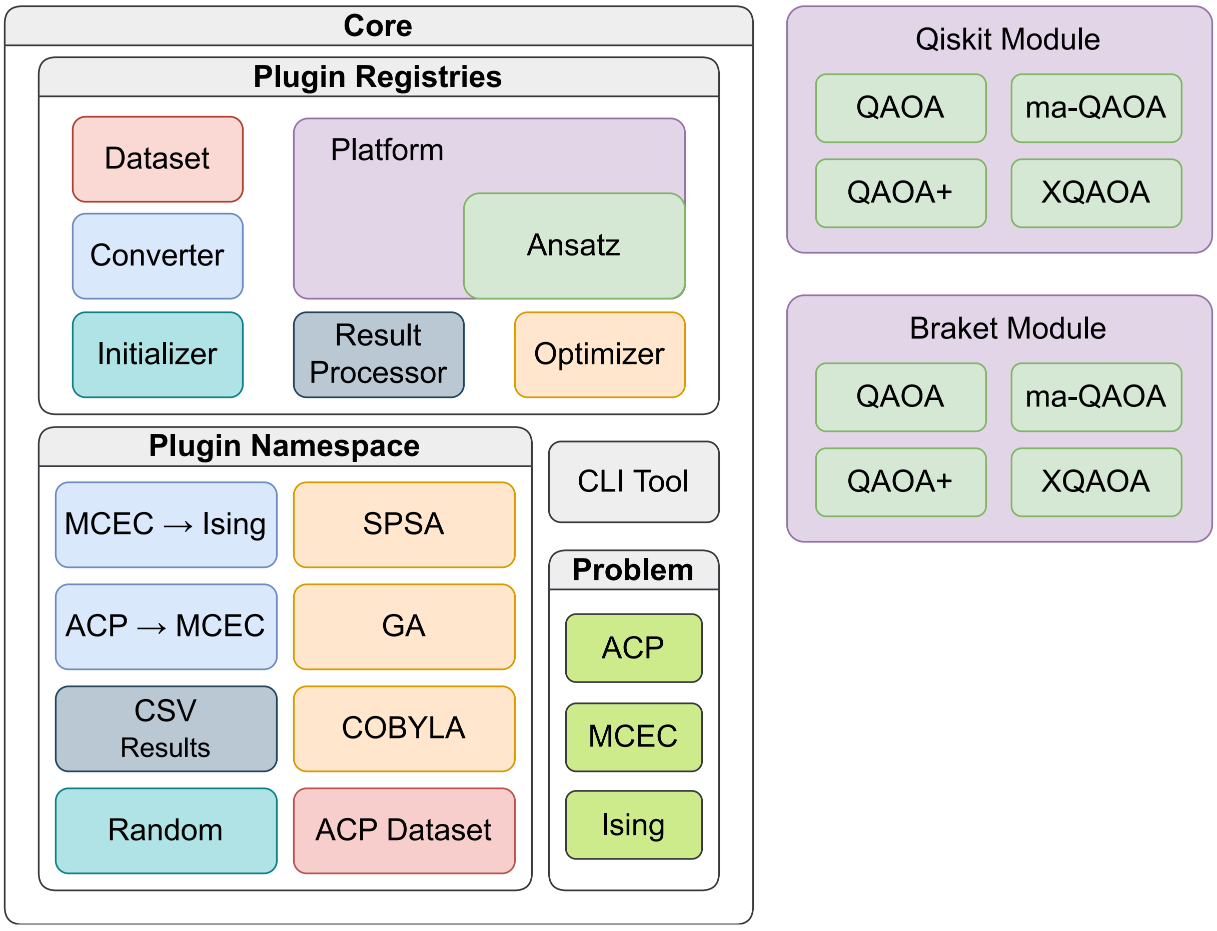

Figure 2: Architectural overview of the suite

6.1.2. Datasets

The optimization pipeline starts with a `DatasetPlugin`. The purpose of such a plugin is to load or generate data and formulate the instance of the problem to be solved. The problem is represented by a `Problem` object that may store arbitrary information about the problem instance alongside the name of the problem and a collection of its *forms*, instances of different problems that are equivalent to the original problem.

6.1.3. Platforms

The optimization suite supports multiple quantum computing platforms via the plugin called `PlatformPlugin`. Each platform plugin must define the methods `run_circuit` and `wrap_optimization`, the former executing the quantum circuit it receives as an argument, while the latter wraps the optimization process, which for example makes it possible to execute the optimization inside a Qiskit Session which is designed to streamline the execution of VQAs.

6.1.4. Ansätze

The quantum circuit executed by the platform plugin is determined by the ansatz of choice and constructed by the corresponding `AnsatzPlugin`. By defining the `construct_circuit` method, an ansatz plugin constructs the quantum circuit for a problem, using the primitives offered by the platform.

6.1.5. Converters

Problems must be in a form appropriate for the construction of the ansatz used. To facilitate this, it is possible to define conversions between problems through a `ConverterPlugin`. Such a plugin implements a one-way conversion from a problem instance to an equivalent instance of a different problem. The conversions implemented by the plugins define a directed graph, from which the shortest path is chosen to convert the problem loaded from the dataset to a problem required by the ansatz.

6.1.6. Initializers

Many approaches suggest that using different initialization strategies in the optimization process can yield improvements in the quality of the solutions. As such, the optimization suite allows for implementing these initialization strategies with the `InitializerPlugin`.

6.1.7. Optimizers

To evaluate different global optimization algorithms, the suite can integrate them using the `OptimizerPlugin`. An optimizer plugin implements the `optimize` method which given a black box function and initial parameters, obtained from an initializer, adjusts the parameters until the minimum is found or some other termination criterion is met.

6.1.8. Result processors

To interpret the results of the optimization process a `ResProcPlugin` can be defined. These plugins receive information about the entirety of the optimization process and are meant for generating different logs and plots.

6.2. Plugin management

As noted in Section 6.1.1, the optimization pipeline is executed by a series of interchangeable plugins. This is achieved through a plugin-based architecture where each stage of the pipeline declares a set of functions for plugins to implement. In this section, we discuss how plugins are managed in the optimization suite by leveraging functionality offered by the Python language.

6.2.1. Python metaclasses

In the data model of the Python programming language [40] data are represented by objects. All objects possess an identity, a type, and a value. The identity of an object is unchangeable after object creation, and its integer representation can be obtained by calling the `id` function. The type of an object defines the supported operations and its possible values, and the `type` function returns the type of an object. Type, generally, is likewise an immutable property of an object. The value of an object can be mutable, like numbers, or immutable, like lists.

The language allows the creation of user-defined types[1] using the `class` keyword or by calling the `type` function with appropriate parameters. Since all data are represented as objects, user-defined types are themselves objects and have a type. The type of a class is often referred to as a *metaclass* and is responsible for defining the object that is bound to the name of the class, which follows the `class` keyword.

Essentially, in the Python programming language metaclasses serve much the same purpose to classes as classes do to objects. It is possible to change the metaclass of a Python class by setting the `metaclass` keyword in the class definition. We exploit this in creating our plugin system, as this allows the definition of behavior that is invoked when a class is defined.

6.2.2. Plugin lifecycle

Every plugin is eventually cataloged by a `Registry`, an object tasked with finding and maintaining references to plugins of a certain type. Each piece of the optimization pipeline has an associated registry that stores discovered plugins in Python dictionaries.

Every registry defines a metaclass, with specialized initialization behavior. In most cases, this behavior simply checks if the class holds the definition of a plugin appropriate for the registry, and if so, registers the plugin with the registry by storing a reference to it, for example in a Python dictionary, where the key is the name of the plugin, while the value is the reference to the plugin object.

The metaclass defined by the registry is used to initialize the definition of a plugin abstract base class, which defines the interface that plugins need to implement in order to participate in the optimization pipeline. As plugins inherit from this abstract base class, the initialization logic of the metaclass is executed up the definition of a plugin class. This automatically registers the plugins in the corresponding registry.

Since plugins are registered upon definition, they are discovered using the import mechanism of the Python language, as class definitions and consequently, the metaclass behaviors execute when a class is first imported. The suite supports two ways of plugin discovery. By default, registries import namespaces associated with the piece of the optimization pipeline the

[1]in older versions of Python class and type were separate concepts [41]

registry is responsible for, this imports plugins that are part of the optimization suite by default and any third-party plugins that are defined in the appropriate namespace. Additionally, registries can import plugins from Python files by passing their file paths to the `discover` method of a registry. This method imports the definitions in the Python files, executing the plugin class definitions, and thereby registering them.

As part of the optimization pipeline the plugins used are instantiated with a user-provided configuration, a Python dictionary, which plugins can use to further configure their behavior. Plugins conform to interfaces that are expected by certain parts of the optimization process. Most of these hooks are provided the current state of the optimization process allowing them to adjust their behavior based on the other parts of the pipeline, for example, an `InitializerPlugin` might request information from an `AnsatzPlugin` about the number of parameters that need to be initialized.

6.3. The optimization pipeline

The optimization pipeline is defined by a `Pipeline` object, that uses the plugins instantiated with the user-provided configurations to orchestrate the optimization process. Every part of the pipeline contains a single plugin except for result processing where multiple result processor plugins can be applied.

By calling the `solve` method of the configured pipeline with the problem to be solved, the optimization process is started. Informed by the `AnsatzPlugin` of choice, the suite looks for the shortest chain of `ConveterterPlugins` to find an appropriate set of conversions to reduce the problem instance to one admitted by the ansatz.

After conversion, the `Problem`, `AnsatzPlugin`, and the `InitializerPlugin` are used to construct a black-box cost function, that executes the ansatz using the integrations provided by the `PlatformPlugin`. The cost function is optimized using the optimization strategy defined by the `OptimizerPlugin`. Platform integrations can define a context in which they intend to run the optimization process, allowing platform-specific circuit optimizations to be applied. The initial values used by the global optimizer to parametrise the ansatz are given by the `InitializerPlugin`.

When the `OptimizerPlugin` finishes execution, an object containing the results are passed to a list of `ResultProcessorPlugins` that interpret the results by generating plots and logfiles.

6.4. Usage

There are multiple ways to use the optimization suite to meet the needs of both industry and research.

The suite can be run inside an easy-to-set-up containerized environment (Docker). Containerized environments offer portable and predictable deployment and have seen widespread utilization in industrial settings. Inside the containerized environment, the suite can be accessed using a CLI tool that reads and parses a configuration file to set up the optimization process. Configuration includes the location of the plugin modules to load, the names of the plugins from which to assemble the optimization pipeline, and also any configurations needed by the plugins themselves. An `input` and a `result` folder are mounted from outside the containerized environment, meant to store the raw data of the datasets used and the results of optimizations respectively. All these make it easy to integrate the suite into industrial processes, but are not necessary when evaluating different approaches to solving a problem.

In such cases, we recommend installing the optimization suite directly inside a Python virtual environment (e.g. *venv*). This method allows for interacting with the CLI tool and also with the package itself using a Python script or notebook, without the added overhead of a containerized environment.

6.5. Example of using the suite

In this section, we show how the suite can be used both through the CLI tool and a Python notebook to execute the optimization pipeline. We solve an instance of the MCEC problem for 2 elements. For this we will use the `GenMCECDataset` plugin that, configured for 2 elements, assigns a prohibitively large cost to the empty and 2-element sets, and a random cost to every other subset. This results in 4 subsets in total, meaning $2^4 = 16$ possible solutions.

We execute the quantum subroutine on the Braket platform with the `BraketIntegration` plugin by mapping the problem to the XQAOA ansatz at depth 5. For the classical optimizer, we use the `ScipyOptimizer` plugin, which wraps the scipy minimize function [42], configured to use the Nelder-Mead algorithm [43, 44] initialized randomly.

We plot the sampling of the probability distribution associated with the lowest energy state obtained during the optimization process using the `BarResProc` result processor plugin.

Since all plugins used are part of the suite, there is no need to list any Python files that contain additional plugin definitions.

As mentioned, we use a generated instance of the MCEC problem with 2 elements, as such we set the `name` field of the dataset plugin to `gen-mcec`, which is the name of the plugin we want to use, additionally, we specify the `num elements` option as 2 to generate a problem instance with 2 elements. Namely, the correxponding part of the configuration file looks like the following.

```
dataset:
  name: gen-mcec
  options:
    num elements: 2
```

The `platform` section of the configuration file is a bit more detailed, we specify that we wish to use the Braket platform but specify no backend as in a locally executed experiment the integration can not be configured to use backends other than the `LocalSimulator` provided in Braket. In the execution section, we explicitly set the execution to be local, and in `argument mapping` provide a key-value pair that will be passed to the platform during execution. In the following example, 1024 measurements are specified, which are used to approximate the probability distribution of outcomes.

```
platform:
  name: braket

  execution:
    local: True
    argument mapping:
      shots: 1024
```

In the `ansatz` section of the file we declare, that we will be using the XQAOA ansatz. In the plugin options `steps` refers to the p value or depth of the ansatz.

```
ansatz:
  name: xqaoa
  options:
    steps: 5
```

The optimization process is initialized using the `RandomInitializer` plugin.

```
initializer:
  name: random
```

As our classical optimizer, we will use the Nelder-Mead algorithm implemented in the scipy package, by using the plugin with the same name.

```
optimizer:
  name: scipy
  options:
    method: Nelder-Mead
```

The list of result processor plugins includes a single item, the `BarResProc` result processor plugin, which generates a bar chart showing the probabilities and costs of the most likely measurement outcomes.

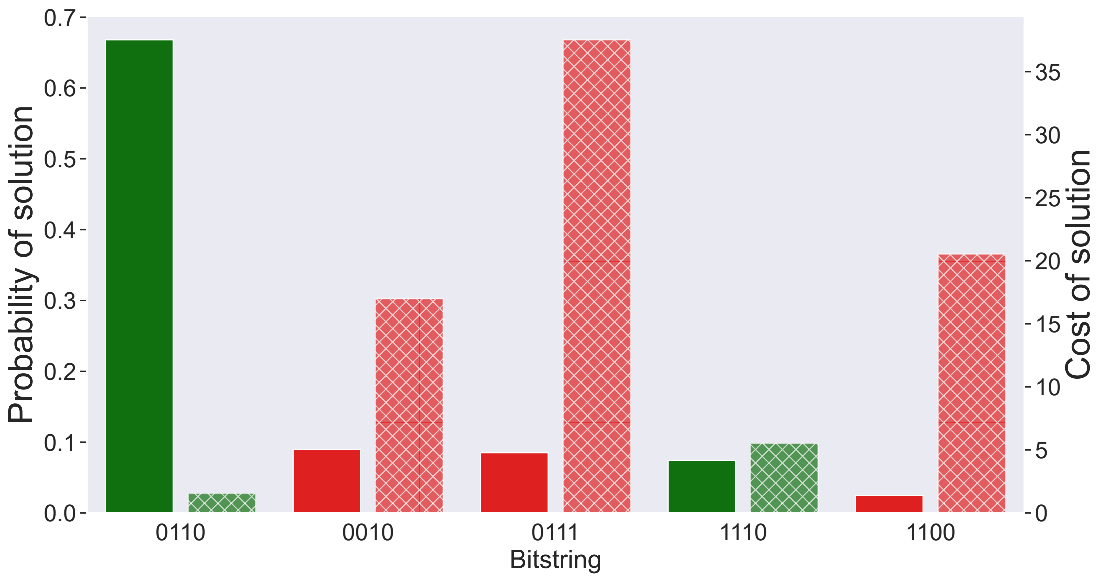

Figure 3: Solid bars indicate the probability of measuring the bitstring, while the cross-hatched bars indicate their associated costs. Red bars indicate an infeasible solution and green bars indicate a feasible solution. In this example the optimum solution is measured with over 0.6 probability.

```
result processor:
  - name: bar
```

The pipeline can similarly be configured inside a Python notebook as seen in Appendix B. After the optimization is finished a figure of the results, like Figure 3, is saved inside the `results` folder.

Chapter 7

Results

In this section, we demonstrate how the optimization suite can be used to obtain some experimental results with a couple of examples.

7.1. Random Instances of MCEC

We implemented a dataset plugin to generate randomized instances of the MCEC problem. The plugin makes use of the interfaces and the definition of the MCEC problem provided by the suite to generate instances of the problem based on the configuration passed to the CLI tool.

In an instance, all subsets of an n element set are available for use in finding an exact cover. Each subset has an associated cost randomly sampled from the interval $[0, 1]$ except the empty and the original set which are given higher costs to prevent them from being part of an optimal solution.

We evaluated the optimization algorithms described in Section 5.4 by solving 30 randomly generated instances for $p = 1, \ldots, 5$ and $n \in \{1, 2\}$, the latter having 16 possible solutions of which 4 are feasible. The main focus of our evaluation was the probability of finding an optimal solution and the number of queries made to the quantum computer during the optimization process.

For each instance, we solved the problem classically and using the QAOA+ ansatz. The quantum subroutine was executed using the Qiskit platform plugin with a locally simulated system. At each query, an approximate distribution over the possible solutions was obtained by sampling the simulated quantum system 1024 times. In the case of multiple solutions with minimal associated costs, their probabilities were added.

To run the experiment we created a simple script that executed the optimization pipeline using the appropriate plugins multiple times and made use of a built-in plugin to obtain information about each optimization run, see Figures 4 and 5 for the results, which shows the distribution of the probability of measuring the optimal solutions and outliers, along with the required number of queries needed for the results with a 95% confidence interval.

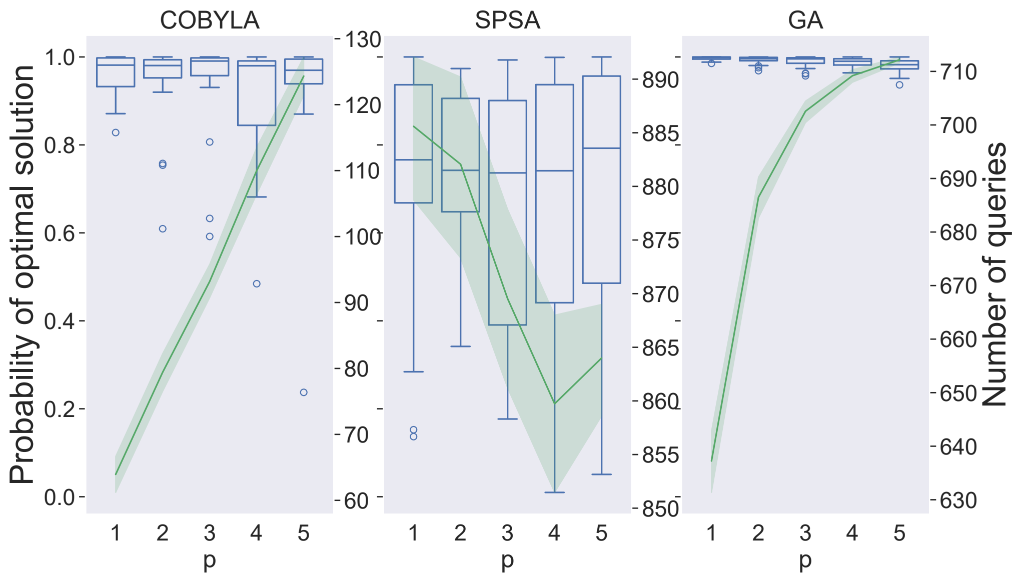

Figure 4: Probabilities of finding an optimal solution (box chart), and number of queries made to the quantum computer (line chart) for different depths p using different classical optimizers in a 2-subset problem

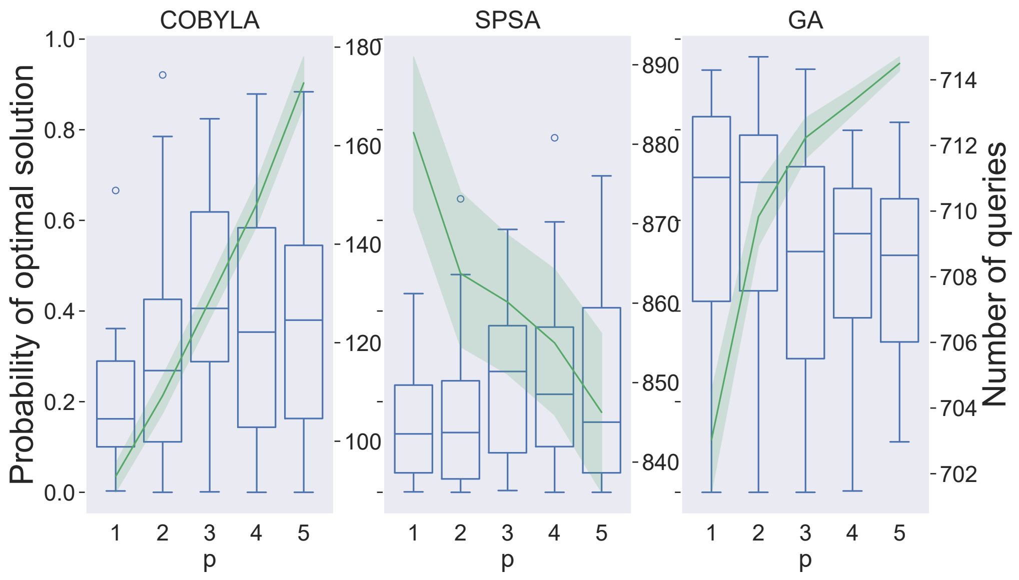

Figure 5: Probabilities of finding an optimal solution (box chart), and number of queries made to the quantum computer (line chart) for different depths p using different classical optimizers in a 4-subset problem

Mostly in accordance with our expectations, for the smaller problem instance, the algorithm performs quite well, while for the larger problem, the results are not so promising with even depth not improving the outcomes. We see an alarming trend in the performance of the SPSA optimizer, which warrants further investigation. We note that these global optimizers support several hyperparameters, their tuning is not part of this demonstration, which focuses on showing the ease of use of the suite.

7.2. Airline Crew Pairing Dataset

In addition to the dataset plugin that generates instances of the MCEC problem, we created a set of plugins that introduce the ACP problem and a sample dataset for the problem. These include a converter plugin that defines the mapping from an instance of the ACP problem to that of the MCEC problem. The dataset itself generates the pairings for the problem from a set of flight legs stored in a file referenced in the configuration. Generation is done as described in Section 2.3 using the set of rules and cost model mentioned in Sections 2.4 and 2.5. The dataset plugin uses the same mechanisms as the suite itself to load the rules and the cost model making it easily extensible.

The dataset we used, with the applied rules, produced 23 pairings, of which 7 can achieve an exact cover with minimal cost.

We ran the optimization process using the CLI tool and configuration files, just as with the MCEC problems we used a local quantum simulator. We ran the pipeline with the QAOA+ ansatz however results warrant further investigation.

Chapter 8

Conclusion

We introduced a quantum optimization suite, a Python package to streamline the use of Variational Quantum Algorithms for experimenting on a variety of problems. We showed how the Airline Crew Pairing problem can be reduced to an instance of Quadratic Unconstrained Binary Optimization and how that formulation can be used to define a cost function of a Variational Quantum Algorithm. We argue that this problem is of great industrial importance and a excellent candidate for evaluating the capabilities of quantum computers of both today and the future.

We explored the Quantum Approximate Optimization Algorithm and some of its popular variants through the optimization suite by integrating them via plugins. Additionally we looked at several optimization algorithms for finding the optimal parameters for the quantum circuit used in VQAs and compared them using the suite.

Through examples we show how the suite can be used to define an optimization pipeline comprised of modular pieces that are easily swappable, leading to fast iteration on the optimization process, allowing researchers to test new ideas quickly. The suite can also be used through a CLI tool, and is straightforwardly deployable via a Docker container making it easy to integrate into industrial processes.

As quantum computers and adoptions of related technologies, such as OpenQASM3 [45], mature so will the optimization suite by introducing new plugins, problems, and workflows to create a unified solution for quantum optimization in industry and research alike.

Appendix A

Mathematical Notation Supplement

A.1. Matrices

A.1.1. Conjugate transpose

The *conjugate transpose* (or adjoint) of a matrix $A \in \mathbb{C}^{n \times m}$ is the matrix $A^* \in \mathbb{C}^{m \times n}$ which is the transpose of A with every element replaced with its complex conjugate.

A.1.2. Unitary Matrices

A square matrix $A \in \mathbb{C}^{n \times n}$ is *unitary* if $A^{-1} = A^*$.

A.1.3. Normal Matrices

A square matrix $A \in \mathbb{C}^{n \times n}$ is *normal* if it commutes with its conjugate transpose, that is: $A^*A = AA^*$. It is easy to see that all unitary matrices are normal, since $A^*A = A^{-1}A = I = AA^{-1} = AA^*$.

A.1.4. Hermitian Matrices

A normal matrix $A \in \mathbb{C}^{n \times n}$ is *hermitian* if $A = A^*$

A.1.5. Matrix diagonalization

A matrix A is *diagonalizeable* if $A = SDS^{-1}$ where D is a diagonal matrix. In this case, the diagonal elements of D are the eigenvalues of A while the columns of S are the corresponding eigenvectors. A matrix is called *unitarily diagonalizable* if it can be diagonalized using a unitary matrix U, meaning: $A = UDU^{-1}$. Matrices are unitarily diagonalizable if and only if they are normal.

A.1.6. Projection Matrices

If for a matrix P all its eigenvalues are either 0 or 1, it is referred to as a *projection matrix*. Projection matrices are hermitian and their associated linear transformation is idempotent, $P = PP$.

A.1.7. Tensor product

If A is an $n \times m$ matrix and B is an $n' \times m'$ matrix, their *tensor product* (also called *Kronecker product*) is the following $nn' \times mm'$ matrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ a_{21}B & \dots & a_{2m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix},$$

where a_{ij} is the element of A in the ith row and jth column.

A.1.8. Matrix exponent

The exponential of an $n \times n$ matrix A, denoted e^A is an $n \times n$ matrix given by the series

$$e^A = \sum_{i=0}^{\infty} \frac{1}{i!} A^i$$

where A^0 is the $n \times n$ identity matrix. For a diagonal matrix D, its exponential is equal to a diagonal matrix with each diagonal element being the exponential function applied to the corresponding element of D, i.e.

$$e^D = \begin{pmatrix} e^{d_{11}} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & e^{d_{nn}} \end{pmatrix}$$

If A is diagonalizable such that $A = SDS^{-1}$ then its exponential is the following:

$$e^A = Se^D S^{-1}.$$

A.2. Dirac notation

Dirac notation is commonly used in physics (and consequently in quantum information science) to write linear algebra. A column vector v is written as $|v\rangle$ (pronounced *"ket v"*) and v^* is written as $\langle v|$ (pronounced *"bra v"*).

With this the inner-product of two vectors v, w can be written as $\langle v|w\rangle$, while their outer-product is $|v\rangle \langle w|$. The tensor product of two vectors v, w is written as $|v\rangle |w\rangle$.

In Dirac notation it is also common to write the vectors $\begin{pmatrix}1 & 0\end{pmatrix}^T$ and $\begin{pmatrix}0 & 1\end{pmatrix}^T$ as $|0\rangle$ and $|1\rangle$ respectively. In the literature, the tensor products of such vectors are often abbreviated as bitstrings and even decimal numbers, for example in a 4 qubit system, $|0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle = |0110\rangle = |6\rangle$.

Appendix B

Experiment in a Python notebook

```python
from vqaopt.core.implementations.dataset.data_gen_mcec import GenMCECDataset
from vqaopt.platform.braket import BraketIntegration
from vqaopt.platform.braket.ansatz import XQAOAAnsatz
from vqaopt.core.implementations.initializer.init_qaoa_random import␣
  ↪RandomInitializer
from vqaopt.core.implementations.optimizer.opt_scipy import ScipyOpt
from vqaopt.core.implementations.resproc.res_bar import BarResProc
from vqaopt.core import Pipeline
```

```python
dataset = GenMCECDataset({"num elements": 2})

platform = BraketIntegration(
    {
        "local": True,
        "run_kwargs": {"shots": 1024},
    }
)
ansatz = XQAOAAnsatz({"steps": 5})
initializer = RandomInitializer()
optimizer = ScipyOpt({"method": "Nelder-Mead"})
bar_processor = BarResProc()
```

```python
problem = dataset.load_problem()
```

```python
_, processed = Pipeline(
    platform,
    ansatz,
    optimizer,
    initializer,
    [bar_processor],
).solve(problem)
```

```python
processed["bar"]
```

Appendix C

Expansion of 4.11

$$D\sum_{i=1}^{n}\left(1-\sum_{j=1}^{m}b_{ij}\frac{\omega_j+1}{2}\right)^2+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$=D\sum_{i=1}^{n}\left[\left(\sum_{j=1}^{m}b_{ij}\frac{\omega_j+1}{2}\right)^2-2\left(\sum_{j=1}^{m}b_{ij}\frac{\omega_j+1}{2}\right)+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$=D\sum_{i=1}^{n}\left[\left(\sum_{j=1}^{m}b_{ij}\frac{\omega_j+1}{2}\right)^2-\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$=D\sum_{i=1}^{n}\left[\frac{1}{4}\left(\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)\sum_{j'=1}^{m}\left(b_{ij'}\omega_{j'}+b_{ij'}\right)\right)-\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)+1\right]$$
$$+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$=D\sum_{i=1}^{n}\left[\frac{1}{4}\left(\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)\left(b_{ij'}\omega_{j'}+b_{ij'}\right)\right)-\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)+1\right]$$
$$+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$=D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}+b_{ij}b_{ij'}\omega_j+b_{ij}b_{ij'}\omega_{j'}+b_{ij}b_{ij'}\right)-\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)+1\right]$$
$$+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\right)+\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}b_{ij}b_{ij'}\right.$$
$$\left.-\sum_{j=1}^{m}\left(b_{ij}\omega_j+b_{ij}\right)+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\right)+\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}b_{ij}b_{ij'}\right.$$
$$\left.-\sum_{j=1}^{m}b_{ij}\omega_j-\sum_{j=1}^{m}b_{ij}+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\right)-\frac{1}{2}\sum_{j=1}^{m}2b_{ij}\omega_j\right.$$
$$\left.+\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}b_{ij}b_{ij'}-\sum_{j=1}^{m}b_{ij}+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\left(b_{ij}\omega_j\left(\sum_{j'=1}^{m}b_{ij'}-2\right)\right)\right.$$
$$\left.+\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}b_{ij}b_{ij'}-\sum_{j=1}^{m}b_{ij}+1\right]+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\left(b_{ij}\omega_j\left(\sum_{j'=1}^{m}b_{ij'}-2\right)\right)+\frac{1}{4}\left(\sum_{j=1}^{m}b_{ij}-2\right)^2\right]$$
$$+\sum_{j=1}^{m}c_j\frac{\omega_j+1}{2}$$

$$= D\sum_{i=1}^{n}\left[\frac{1}{4}\sum_{j=1}^{m}\sum_{j'=1}^{m}\left(b_{ij}b_{ij'}\omega_j\omega_{j'}\right)+\frac{1}{2}\sum_{j=1}^{m}\left(b_{ij}\omega_j\left(\sum_{j'=1}^{m}b_{ij'}-2\right)\right)+\frac{1}{4}\left(\sum_{j=1}^{m}b_{ij}-2\right)^2\right]$$
$$+\sum_{j=1}^{m}\frac{c_j\omega_j}{2}+\sum_{j=1}^{m}\frac{c_j}{2}$$

$$
\begin{aligned}
&= \frac{D}{4}\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{j'=1}^{m} b_{ij}b_{ij'}\omega_j\omega_{j'} \\
&\quad + \frac{D}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}\omega_j\left(\sum_{j'=1}^{m} b_{ij'} - 2\right) + \sum_{j=1}^{m}\frac{\omega_j c_j}{2} \\
&\quad + \frac{D}{4}\sum_{i=1}^{n}\left(\sum_{j=1}^{m} b_{ij} - 2\right)^2 + \sum_{j=1}^{m}\frac{c_j}{2}
\end{aligned}
$$

Bibliography

[1] Pontus Vikstål, Mattias Grönkvist, Marika Svensson, Martin Andersson, Göran Johansson, and Giulia Ferrini. Applying the quantum approximate optimization algorithm to the tail-assignment problem. *Physical Review Applied*, 14(3), September 2020.

[2] Airlines for America. A4A Passenger Airline Cost Index. https://www.airlines.org/dataset/a4a-quarterly-passenger-airline-cost-index-u-s-passenger-airlines/, 2024. Accessed: 2024.02.15.

[3] Airlines for America. State of U.S. Aviation. https://www.airlines.org/dataset/state-of-us-aviation/, 2024. Accessed: 2024.02.15.

[4] D. Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, 400(1818):97–117, July 1985.

[5] Lov K. Grover. A fast quantum mechanical algorithm for database search. 1996.

[6] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, SFCS-94. IEEE Comput. Soc. Press.

[7] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[8] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics*, 94(1), February 2022.

[9] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014.

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[11] Kostas Blekos, Dean Brand, Andrea Ceschini, Chiao-Hui Chou, Rui-Hao Li, Komal Pandya, and Alessandro Summer. A review on quantum approximate optimization algorithm and its variants. 2023.

[12] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019.

[13] Yuta Shingu, Yuya Seki, Shohei Watabe, Suguru Endo, Yuichiro Matsuzaki, Shiro Kawabata, Tetsuro Nikuni, and Hideaki Hakoshima. Boltzmann machine learning with a variational quantum algorithm. *Phys. Rev. A*, 104:032413, Sep 2021.

[14] Scott E. Smart and David A. Mazziotti. Quantum solver of contracted eigenvalue equations for scalable molecular simulations on quantum computing devices. *Physical Review Letters*, 126(7), February 2021.

[15] Qi Gao, Hajime Nakamura, Tanvi P. Gujarati, Gavin O. Jones, Julia E. Rice, Stephen P. Wood, Marco Pistoia, Jeannette M. Garcia, and Naoki Yamamoto. Computational investigations of the lithium superoxide dimer rearrangement on noisy quantum devices. *The Journal of Physical Chemistry A*, 125(9):1827–1836, February 2021.

[16] Tyson Jones, Suguru Endo, Sam McArdle, Xiao Yuan, and Simon C. Benjamin. Variational quantum algorithms for discovering hamiltonian spectra. *Phys. Rev. A*, 99:062304, Jun 2019.

[17] Alain Delgado, Juan Miguel Arrazola, Soran Jahangiri, Zeyue Niu, Josh Izaac, Chase Roberts, and Nathan Killoran. Variational quantum algorithm for molecular geometry optimization. *Phys. Rev. A*, 104:052402, Nov 2021.

[18] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, June 2017.

[19] Prasanna Date, Davis Arthur, and Lauren Pusey-Nazzaro. Qubo formulations for training machine learning models. *Scientific Reports*, 11(1), May 2021.

[20] Joey McCollum and Thomas Krauss. Qubo formulations of the longest path problem. *Theoretical Computer Science*, 863:86–101, 2021.

[21] Kyungtaek Jun. Qubo formulations for a system of linear equations. *Results in Control and Optimization*, 14:100380, 2024.

[22] Andrew Lucas. Ising formulations of many NP problems. *Front Phys*, 2, 2014.

[23] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, February 1925.

[24] Sorin Istrail. Statistical mechanics, three-dimensionality and NP-completeness: I. universality of intracatability for the partition function of the ising model across non-planar surfaces (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing - STOC '00*, pages 87–96, New York, New York, USA, may 2000. ACM Press.

[25] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, August 2021.

[26] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. 2000.

[27] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation. *SIAM Review*, 50(4):755–787, January 2008.

[28] M. Born and V. Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3–4):165–180, March 1928.

[29] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, Jan 2018.

[30] Masuo Suzuki. General decomposition theory of ordered exponentials. *Proceedings of the Japan Academy, Series B*, 69(7):161–166, 1993.

[31] Naomichi Hatano and Masuo Suzuki. *Finding Exponential Product Formulas of Higher Orders*, page 37–68. Springer Berlin Heidelberg, November 2005.

[32] Rebekah Herrman, Phillip C. Lotshaw, James Ostrowski, Travis S. Humble, and George Siopsis. Multi-angle quantum approximate optimization algorithm. 2021.

[33] Michelle Chalupnik, Hans Melo, Yuri Alexeev, and Alexey Galda. Augmenting qaoa ansatz with multiparameter problem-independent layer. 2022.

[34] V. Vijendran, Aritra Das, Dax Enshan Koh, Syed M. Assad, and Ping Koy Lam. An expressive ansatz for low-depth quantum approximate optimisation. *Quantum Science and Technology*, 9(2):025010, February 2024.

[35] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, page 51–67. Springer Netherlands, 1994.

[36] J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, March 1992.

[37] S. Bhatnagar, H.L. Prasad, and L.A. Prashanth. *Stochastic Recursive Algorithms for Optimization: Simultaneous Perturbation Methods*, pages 41–44. Springer London, 2013.

[38] Giovanni Acampora, Angela Chiatto, and Autilia Vitiello. Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm. *Applied Soft Computing*, 142:110296, July 2023.

[39] The PyPA recommended tool for installing Python packages. `https://pypi.org/project/pip/`. Accessed: 2024.04.28.

[40] Python Software Foundation. The Python Language Reference » 3. Data model. `https://docs.python.org/3.10/reference/datamodel.html`. Accessed: 2024.04.20.

[41] Guido van Rossum. Unifying types and classes in Python 2.2. `https://www.python.org/download/releases/2.2.3/descrintro/`, 2002. Accessed: 2024.04.20.

[42] scipy.optimize.minimize. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`. Accessed: 2024.04.20.

[43] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[44] Fuchang Gao and Lixing Han. Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51(1):259–277, May 2010.

[45] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing*, 3(3):1–50, September 2022.