QPLEX: Towards the Integration of Platform Agnostic Quantum Computation into Combinatorial Optimization Software

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

by

Juan Fernando Giraldo Botello

B.Eng. in Software Systems Engineering, Universidad Icesi, Colombia, 2021

We acknowledge and respect the lək^{w}əŋən peoples on whose traditional territory the university stands, and the Songhees, Esquimalt and W̱SÁNEĆ peoples whose historical relationships with the land continue to this day.

QPLEX: Towards the Integration of Platform Agnostic Quantum Computation into Combinatorial Optimization Software

by

Juan Fernando Giraldo Botello

B.Eng. in Software Systems Engineering, Universidad Icesi, Colombia, 2021

Supervisory Committee:

Dr. Hausi A. Müller, Supervisor
Department of Computer Science, University of Victoria

Dr. Norha M. Villegas, Supervisor
Department of Computer Science, University of Victoria

Dr. Nikitas J. Dimopoulos, Outside Member
Department of Electrical and Computer Engineering, University of Victoria

Abstract

Quantum computing has the potential to surpass the capabilities of current classical computers when solving complex problems. Combinatorial optimization has emerged as a pivotal target area for quantum computers, as problems in this field are renowned for their complexity and resource-intensive nature. Moreover, these challenges play a critical role in various industrial sectors, including logistics, manufacturing, and finance. This thesis explores the integration of quantum computation into classical software tools as a means to potentially address combinatorial optimization problems more efficiently and effectively.

This work introduces QPLEX, a Python software library that enables practitioners and researchers to implement the general mathematical formulation of a given combinatorial optimization problem once and execute it seamlessly on multiple quantum devices using various quantum algorithms. This software solution automatically adapts a general optimization model to the specific instructions utilized by the target quantum device's SDK. It offers a versatile execution workflow capable of running gate-based hybrid quantum-classical algorithms for combinatorial optimization in a platform-agnostic manner. This approach reduces the programming overhead required for modeling and experimenting with combinatorial optimization solutions.

Within this manuscript, we address and introduce the various aspects associated with the development of QPLEX in a clear and comprehensive manner. These aspects encompass the quantum algorithms and quantum hardware available in the library, along with QPLEX's system design and implementation. Additionally, we provide a guide on how to use the library and conduct a thorough evaluation of the software solution within a specific use case as part of this thesis.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This thesis couldn't have been finished without the support and guidance of many people throughout my master's degree. First and foremost, I would like to thank my supervisors, Dr. Hausi A. Müller and Dr. Norha M. Villegas, for the immense influence they had on my research and overall on my academic path. Their knowledge and experience were paramount to crafting my skill set as well as motivating me to pursue new projects and challenges. Additionally, I highlight the mentoring efforts and assistance of Dr. Ulrike Stege and Dr. Gabriel Tamura, who, in spite of not being direct supervisors of my work, played an important mentoring role over the past two years. Moreover, I express my gratitude to all the members of the Rigi Research Lab for all the insightful conversations we had and the projects we worked on. Especially to Tristan, José, and Felipe, it was a wonderful experience to work with you guys. Finally, I want to thank all my family and friends back in Colombia deeply; they were my main motivation to continue pursuing my academic goals and to always try my best at everything I do.

Chapter 1

Introduction

Quantum Computing (QC) is a new computing field, where calculations are performed by manipulating quantum bits (i.e., qubits) through the use of quantum mechanical properties such as superposition, entanglement, and interference [5]. Given the nature of these quantum operations, QC has the potential to surpass the capabilities of current classical computers when solving complex problems, including drug discovery and material design. The notion of a quantum computer was first introduced in the 1980s by Richard P. Feynman with the goal of developing a tool that could solve one of the most important problems in physics: simulating quantum mechanical systems [6]. During the following forty years, scientists and engineers greatly advanced the field of QC by inventing novel quantum algorithms [7], [8], [9] and building quantum computers capable of demonstrating an advantage in specific tasks when compared to their classical counterparts [10], [11], [12].

Nevertheless, today, there are still many technical challenges to overcome before quantum computers can be widely used. As a matter of fact, the current state of the field has been characterized as the Noisy Intermediate-Scale Quantum (NISQ) era [13], which refers to the limitations in the applicability and functionality of contemporary quantum devices due to their small number of qubits and a considerable amount of errors caused by noise. Generally, the execution of most quantum algorithms requires far better hardware resources than what is provided by NISQ devices. Nonetheless, the emerging field of hybrid quantum-classical computing offers a promising solution to these challenges. By using these limited quantum computers and incorporating classical resources into their execution, this approach effectively addresses some of the shortcomings of the quantum device, allowing the production of useful results. Hybrid algorithms are the prime candidates for demonstrating quantum advantage for real-world applications, such as molecular simulation [14].

The advantages of hybrid quantum-classical computing are particularly evident when applied to the field of optimization [15]. This area has emerged as one of the key target applications for QC, as problems found in this field are often computationally challenging and play a critical role in many different industrial application sectors, such as logistics [16], manufacturing [17], finance [18], among others. Recent advances in hybrid algorithms have provided techniques for solving complex optimization problems using universal gate-based quantum computers [2], [19], [20], [21]. Additionally, adiabatic computing, another QC paradigm, has also proven to be a proper method for tackling these types of

problems [22], [23], [24]. These techniques could potentially bring a speed up compared to classical approaches while delivering higher quality solutions.

The development of hybrid quantum-classical solutions that effectively use computing resources and are able to tackle large-scale real-world problems is one of the main goals of Quantum Software Engineering (QSE). This developing area of research aims at using knowledge from classical software engineering to bootstrap the transition from theoretical QC to practical quantum applications. Many crucial aspects of implementing hybrid software have begun to be addressed by QSE. These include the impact of QC on the classical software development lifecycle and the new stages to be considered for such applications [25], effective techniques for testing quantum software [26], and the evolution of classical development and operations practices to support quantum applications [27]. However, QSE is still in a very early stage, offering immense opportunities for innovative solutions and pivotal advancements. We believe that continuous development of quantum software engineering solutions is crucial in order to fully realize the potential benefits of QC applications in the near future. In this thesis, we contribute to the growth of this field by exploring the applicability of classical software engineering techniques for the formulation, modeling, and platform-agnostic execution of quantum combinatorial optimization problems.

1.1 Motivation

Over the last ten years, a large number of QC use cases have been explored for solving classically intractable computational problems. These use cases are aimed at providing practical and efficient near-term solutions for industries such as finance, logistics, and manufacturing. For instance, hybrid quantum-classical algorithms have been used as a tool to find an optimal combination of assets to include in an investment portfolio over a given period while considering market impact costs [28]. They have also been employed to optimize the routing of delivery vehicles over different time windows for various logistics scenarios [29]. Additionally, some QC approaches have shown to be an efficient option for modeling and distributing complex, large-scale energy supply chains, which are crucial for a range of industrial and civil operations [30]. This type of project, and the significant investment in research and development in quantum computing applications by leading companies, such as Bosch,[1] Airbus,[2] and BMW,[3] has sparked a widespread interest in this technology. According to Zapata Computing's 2021 Annual Report on Quantum Computing Adoption,[4] approximately 74% of enterprises have adopted or were planning to adopt QC in 2022, highlighting the growing recognition of its potential to transform a variety of industries.

With the imminent incorporation of quantum computing into most companies' technology stack, it is only natural that quantum combinatorial optimization applications begin to

[1] https://www.bosch.com/stories/future-of-quantum-computing/
[2] https://www.airbus.com/en/innovation/disruptive-concepts/quantum-technologies
[3] https://challenge.quantum.bmw.cloud/
[4] https://www.zapatacomputing.com/enterprise-quantum-adoption-2022/

be developed in-house to tackle industry-specific use cases. However, solving optimization problems using quantum computers is a nontrivial task; a certain level of proficiency in quantum algorithms and programming is necessary to develop such solutions. This adds friction to the programming experience, making it considerably more difficult for a software engineer to start experimenting with QC. Additionally, given the variety of QC providers and algorithm implementations for solving optimization problems, some techniques executed on certain devices may yield better results, achieve better performance, or be more efficient than others for a given problem [31] [32]. Thus, testing different algorithmic approaches and quantum processing units is essential in order to find the best alternative for a specific use case. This is especially important if one intends to benchmark against classical solutions.

Building on these insights, this thesis presents QPLEX, a software library that allows developers to implement a general mathematical formulation once for an optimization problem and execute it seamlessly on multiple quantum devices using a wide variety of quantum algorithms. The proposed solution enables optimization software developers to experiment with quantum resources selectively and assess performance improvements of hybrid quantum-classical optimization solutions. This work also dwells on how to formulate optimization problems so they are amenable to the library, explores different algorithms for quantum combinatorial optimization, and reviews the benefits and limitations of multiple quantum providers currently available in the market.

1.2 Problem Definition and Research Questions

Quantum combinatorial optimization holds immense potential for various industrial applications. However, this technology is still in its early stages, leaving uncertainty about the most suitable algorithms and quantum hardware for specific problems. Furthermore, the accessibility and utilization of QC resources pose challenges for most software practitioners, hindering their ability to explore the technology's potential and identifying practical use cases. To promote the widespread adoption of quantum combinatorial optimization in any industrial workflow, it is crucial to bridge the gap between non-quantum developers and the rapidly expanding array of QC resources, which includes algorithms, techniques, and platform providers. This requires the development of a software solution that facilitates the seamless interaction with these resources without imposing additional programming overhead. Our work constitutes a significant effort towards achieving this ambitious goal. This research aims to answer the following research questions:

RQ1: Which quantum algorithms are used for solving optimization problems, and what are their strengths and weaknesses?

RQ2: What are the most prevalent quantum computing hardware implementations, and how can the different offerings from platform providers be characterized to make an informed decision when selecting where to solve a combinatorial optimization problem?

RQ3: How can knowledge from classical software engineering be applied to develop a software library for platform-agnostic quantum combinatorial optimization?

1.3 Contributions

The contributions of this thesis, are as follows.

C1: An assessment of near-term quantum algorithms employed for solving combinatorial optimization problems.

C2: A comprehensive review of the most prevalent quantum hardware technologies and platforms. Including the description of characterization aspects employed when selecting a platform offering.

C3: A programming library for the modeling and seamless execution of optimization problems using different quantum computing platforms and optimization algorithms.

1.4 Research Methodology

In order to answer the research questions outlined above, we developed a Python software library for platform-agnostic quantum combinatorial optimization. This programming language was selected to be the base of our implementation given that it is among the most used in scientific research, more specifically the quantum computing field. We went through the main stages of building a software product, starting with problem analysis and formulation. This includes reviewing quantum algorithms for optimization and exploring the different quantum computing providers currently in the market, then moving to the design of the library and finalizing with the implementation of the solution.

To begin this work, we performed an assessment of the current state of quantum combinatorial optimization and studied the different algorithms that can be used to approach optimization problems. We analyzed their implementations and reported on their respective limitations. This stage helped us answer the first research question.

For the second research question, we described the three most well-known quantum hardware implementations while highlighting their strengths and weaknesses. Also, we reviewed three different quantum computing platform providers and their respective hardware and software offerings: IBM Quantum, D-Wave, and Amazon Braket. With this selection, we covered both QC paradigms (i.e., gate-based and annealing) and two different hardware implementations, superconducting and ion trap. Additionally, a set of characterization aspects is presented to aid potential users in making an informed decision when selecting a quantum computer for the solution of a given problem formulation. These aspects include the scale of the quantum computers, quality of execution, time-to-solution, and additional software features.

To answer the third research question, we took into account our findings and results from RQ1 and RQ2 in conjunction with tools and knowledge from classical software engineering (e.g., design patterns) to develop a platform-agnostic software library that effectively integrates QC into classical optimization workflows. To accomplish this goal, we identified requirements for the library and proceeded to design and implement a solution that addresses all of them. Finally, we proceeded to validate our software system by testing its modeling and execution capabilities.

1.5 Thesis Outline

This chapter presents the motivation of this research, outlines the research questions we addressed, and highlights the contributions of our work. The remaining chapters of this thesis are organized as follows.

Chapter 2: explains key concepts of this research and describes the background and state-of-the-art.

Chapter 3: introduces the concept of quantum combinatorial optimization and reviews different quantum algorithms used for this purpose.

Chapter 4: presents a review of quantum computing hardware implementations and popular quantum platform providers, while characterizing aspects for addressing the applicability of a given device for a specific combinatorial optimization problem.

Chapter 5: elucidates the requirements, design, and implementation of QPLEX.

Chapter 6: presents the validation and testing of the proposed platform-agnostic software library for quantum combinatorial optimization through a use-case.

Chapter 7: summarizes this thesis and discusses ideas for future work.

Chapter 2

Background and Related Work

The integration of quantum computation into the field of combinatorial optimization is a relatively recent endeavor within the scientific community. The widespread availability of larger and more robust quantum computers, along with the continuously increasing complexity of industrial problems, has facilitated the rapid development of this field. There is a strong interest in utilizing quantum computers to expedite and enhance solutions for complex industrial challenges, such as carbon fixation, ammonia synthesis, and battery development. Merging these two intricate domains is a formidable task, necessitating a profound understanding of both quantum computing (QC) and combinatorial optimization (CO). This chapter aims to provide an overview of both QC and classical CO while highlighting relevant work related to this thesis.

2.1 Quantum Computing

QC refers to a novel computational paradigm in which the units of information and the logical operations significantly differ from those used in classical computers. As implied by its name, quantum computing harnesses the principles of quantum mechanics to construct a new foundation for information theory and computation. Its primary goal is to tackle classically intractable problems that arise across various fields, with a particular focus on simulating quantum mechanical systems.

Quantum computation is achieved by employing qubits instead of conventional bits for storing and processing information. Qubits are the fundamental units of information in QC, used to store data in the form of quantum states and perform calculations by exploiting quantum phenomena. In contrast to classical bits, which have only two states (0 or 1), a qubit cam be in a any state that is a complex linear combination of $|0\rangle$ and $|1\rangle$. It is worth noting that the literature often employs Dirac notation to describe the possible quantum states that a qubit can assume. This notation is relatively straightforward, as it represents linear algebra elements, such as vectors and matrices, as well as operations like tensor products and inner products. The following equations present the most basic states a qubit can hold, represented in both Dirac and linear algebra notation.

$$\text{Pure State 0 (Ket-Zero): } |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.1}$$

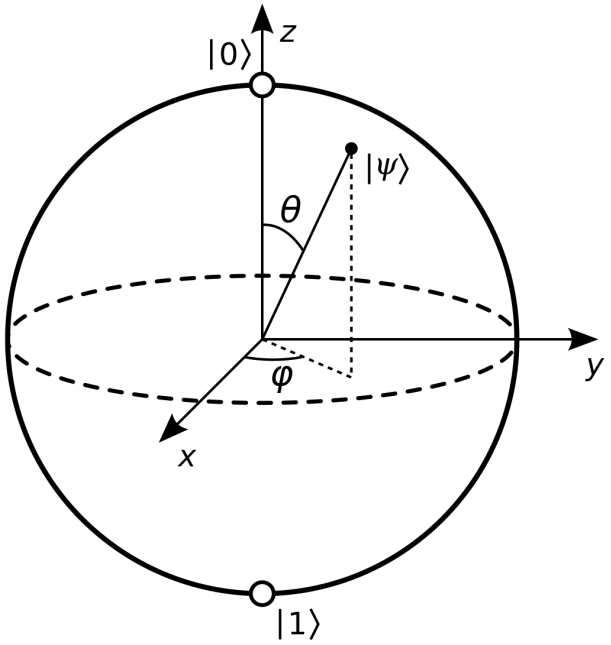

Figure 2.1|: Bloch Sphere representing the state of a single qubit.

$$\text{Pure State 1 (Ket-One): } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

$$\begin{aligned} &\text{Superposition state } (|\psi\rangle): \alpha|0\rangle + \beta|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &\text{Subject to Born's rule: } |\alpha|^2 + |\beta|^2 = 1 \rightarrow \alpha, \beta \in \mathbb{C} \end{aligned} \tag{2.3}$$

Two key quantum effects, superposition and entanglement, are harnessed to manipulate qubits. Superposition allows a qubit to exist in a linear combination of pure states (e.g, $|0\rangle$ and $|1\rangle$). Entanglement enables two qubits to be directly connected and instantly react to each other's actions. To visualize the current state of a qubit, the Bloch Sphere is often used, providing a geometrical representation of the state space of a two-level quantum system (as depicted in Figure 2.1). Here, the two computational bases (or pure states) are displayed in the Z axis, while superpositions of these states can be represented as a vector rotated about any of the three axes. Equation 2.4, a variant of Euler's equation, explains how the rotation angles on each axis affect the overall superposition state.

$$|\psi\rangle = cos(\frac{\theta}{2})|0\rangle + e^{i\phi} sin\frac{\theta}{2}|1\rangle \tag{2.4}$$

To achieve complex quantum computations, it is necessary to have an array of qubits organized on a single chip. This allows for the creation and manipulation of larger and more intricate quantum states. The resulting chip is analogous to a classical CPU and can

be referred to as a Quantum Processing Unit (QPU). However, the QPU alone does not constitute a complete quantum computer; it requires a large set of hardware and software components to fully enable quantum computation. These components include sensors, classical control logic, networking interfaces, and sometimes lasers or microwave emitters. When all these parts come together, they form a fully-fledged quantum computer. Quantum computers can be implemented using different computational paradigms, such as gate-based and adiabatic quantum computing. These paradigms determine how qubits are manipulated and, consequently, how quantum computations are performed. The following subsections will provide a more detailed explanation of both paradigms.

2.1.1 Gate-based Quantum Computing

This paradigm is the most prevalent among QC companies due to the simplicity of its computational model based on linear algebra. As implied by its name, quantum operations are performed on a QPU using quantum gates, much like classical operations are executed on CPUs through logical gates [5]. Each of these gates is represented by a unitary matrix, which applies a transformation in the form of a rotation to a vector containing the state of the current computation. Quantum gates can be applied to single or multiple qubits simultaneously. Below, a list of the most important quantum gates is presented, both in Dirac and matrix notation. Single-qubit gates are represented as 2x2 matrices, while two-qubit gates are 4x4 matrices.

$$
\begin{aligned}
&\text{Pauli-X: } |X\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\
&\text{Pauli-Y: } |Y\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\
&\text{Pauli-Z: } |Z\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
&\text{Hadamard: } |H\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
&\text{Phase: } |S\rangle = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \\
&\text{Controlled-NOT: } |CX\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\end{aligned}
\tag{2.5}
$$

Within this paradigm, quantum operations are expressed using quantum circuits. These circuits consist of the qubits to be manipulated and a sequence of quantum gates to be

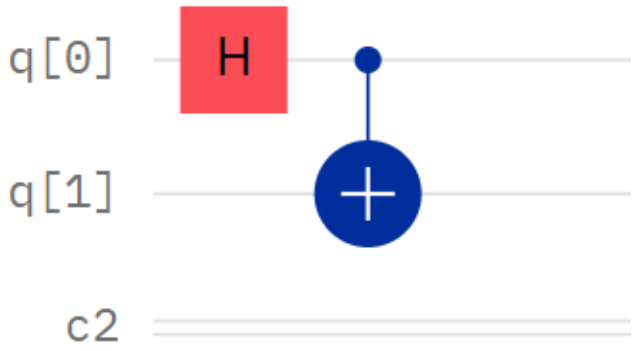

Figure 2.2|: Two-qubit quantum circuit. Generated using the Qiskit Circuit Composer.

applied to each of them. Since the information stored in qubits ultimately needs to be measured as classical bits, quantum circuits also include classical registers. In Figure 2.2, a simple two-qubit quantum circuit is depicted, where each horizontal line represents a qubit and lists all the quantum gates applied to it. In the case of qubit 0 (q[0]), a Hadamard gate is applied, while both qubit 0 and qubit 1 (q[1]) are subject to a controlled-NOT gate. At the bottom of the circuit, the classical registers (c2) are shown as a double horizontal line; these registers are used to record the measured information.

Larger sets of quantum gates can be employed within quantum circuits to construct quantum algorithms. These algorithms have specific inputs, outputs, and well-defined objectives. In comparison to classical algorithms, the scientific community recognizes a relatively small number of quantum algorithms. Among the most popular and influential are the Quantum Teleportation algorithm [33], the Deutsch–Jozsa algorithm [7], the Bernstein–Vazirani algorithm [34], and the Quantum Phase Estimation algorithm [35]. The field of quantum algorithms is still in its early stages of development, with significant potential for the creation of pivotal gate-based algorithms that could potentially offer substantial quantum advantages.

2.1.2 Adiabatic Quantum Computing

Similarly to GBQC, adiabatic quantum computing (AQC) is also a universal paradigm for quantum computation and it started as an approach to solving optimization problems [36]. This approach consists of encoding the solution of computational problems into the ground state of a time-dependent quantum Hamiltonian [37]. In this context, a Hamiltonian can be treated as a cost function that determines the energy of a given system, in this case, a quantum state. Unlike GBQC, AQC does not utilize quantum gates to perform computations. Instead, it leverages the quantum adiabatic theorem to decode the solution from the Hamiltonian. This theorem states that if a quantum system starts in the ground state of a time-dependent Hamiltonian $H(t)$ at time $t = 0$, and gets evolved slowly enough to a different Hamiltonian H_1 at time $t = n$, the ground state of the evolving Hamiltonian will

remain in its ground state [38], allowing us to find the desired solution. To fully harness this mechanism for quantum computation, specialized hardware systems, such as quantum annealers, can be implemented. These machines have the capability to map problem Hamiltonians onto the underlying architecture of the QPU, enabling the constructed quantum system to converge toward the desired solution through a time-dependent evolution. The specifics of how annealers work are discussed further below.

2.2 Hybrid Quantum-Classical Computing

This branch of QC aims to address the hardware limitations present in current quantum computers. The lack of error correction, the short coherence times (the duration for which a qubit can maintain its quantum properties), and the low gate fidelity make it difficult, and sometimes impossible, to obtain useful results of complex computations performed on near-term quantum computers [39]. Quantum circuits with a high number of qubits and quantum gates are especially challenging to address with present-day quantum hardware, heavily restricting the applications that these devices can be used for. By incorporating aspects of classical computation into quantum computing workflows, researchers have been able to implement hybrid quantum-classical solutions capable of addressing larger and more intricate problem instances [40]. This is especially relevant for this work as the main focus of these solutions is complex quantum simulations and combinatorial optimization. The following subsections introduce the two most important aspects involved in hybrid quantum-classical solutions: algorithms and systems.

2.2.1 Hybrid Quantum-Classical Algorithms

Hybrid algorithms combine quantum circuit execution and a classical optimization routine. This workflow is possible because these types of algorithms take advantage of variational quantum circuits, which are parameterized quantum circuits with a low number of quantum gates (shallow circuits). These circuits are easy to prepare and, in most cases, are short enough to not be considerably affected by decoherence times or low gate fidelity. In this context, "parameterized" means that the circuit contains rotation gates with a variational parameter corresponding to its rotation angle. The main idea of this approach is to find the best combination of angles so that a previously defined cost function is minimized. In most cases, the energy of the quantum system is used for this purpose. The process of finding these angles is performed through a classical optimizer, which, at each iteration, generates a new set of parameters to be evaluated with the cost function. When the optimization process has converged, it is safe to say that the optimal parameters have been found. This variational principle is employed by state-of-the-art hybrid quantum-classical algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) [15] and the Variational Quantum Eigensolver (VQE) [14].

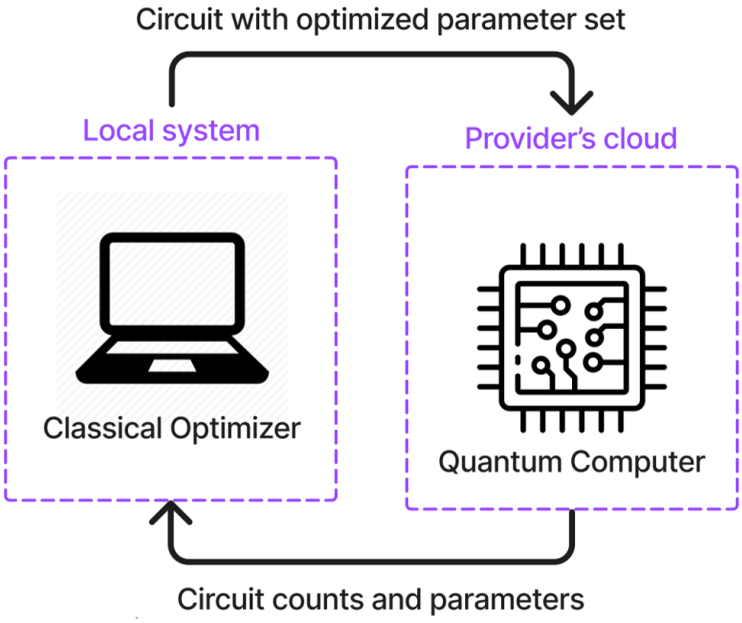

Figure 2.3|: Execution workflow for a variational algorithm on a classical system.

2.2.2 Hybrid Quantum-Classical Systems

Hybrid systems provide hardware infrastructure and software components to enhance and optimize the execution of hybrid algorithms. In a typical execution of a hybrid algorithm, the quantum section of the algorithm runs on a remote QPU, with results transmitted back to the local system. Subsequently, a classical optimization routine is performed, and this process repeats until the loss function converges. Due to the variational nature of these algorithms and the high number of iterations typically required to solve complex problems, any latency in communications or time spent in the execution queue of the provider's cloud can lead to considerable processing delays. The improvement offered by hybrid systems is realized by placing classical and quantum hardware in close proximity, virtually eliminating latency caused by data transfers. Furthermore, these systems feature software that allows for bundling multiple program executions into a single session, enabling users to maintain a queue priority until the entire workflow is completed. These two features alone provide a significant advantage over traditional execution workflows on non-hybrid systems. Figures 2.3 and 2.4 illustrate the main differences in the execution workflow for hybrid and classical systems.

At the present time, only two quantum providers enable execution on hybrid systems: IBM Quantum and Amazon Braket. The former does it through Qiskit Runtime,[1] while the latter uses Braket Hybrid Jobs.[2] Nevertheless, other quantum companies, such as Rigetti,[3] are planning to develop similar systems to provide an advantage to their customers.

[1] https://qiskit.org/ecosystem/ibm-runtime/
[2] https://docs.aws.amazon.com/braket/latest/developerguide/braket-jobs.html
[3] https://www.globenewswire.com/news-release/2022/02/16/2386276/0/en/Rigetti-and-Ampere-Announce-Strategic-Partnership-to-Develop-Cloud-native-Hybrid-Quantum-Classical-Computers.html

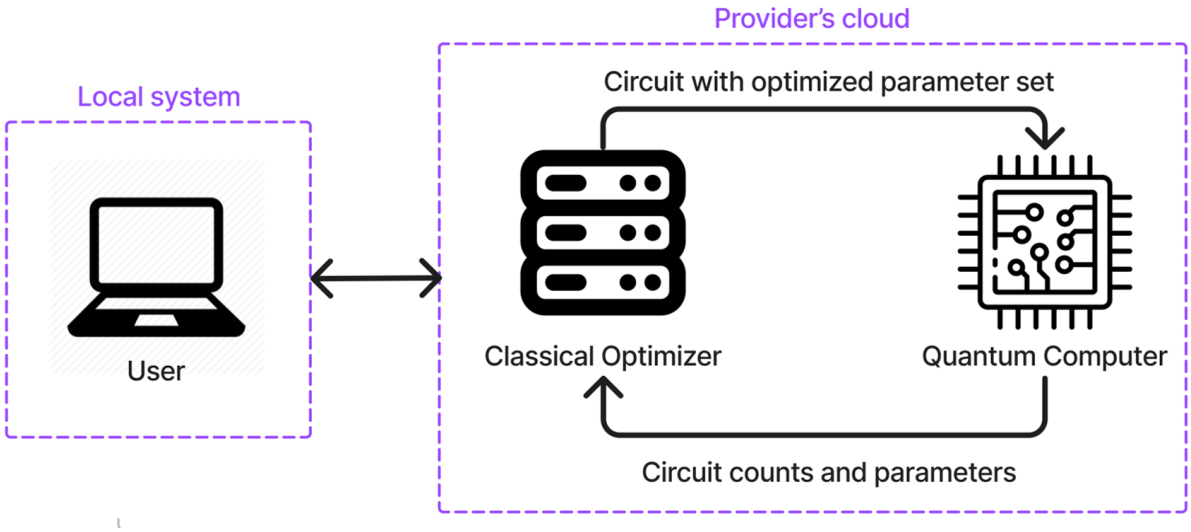

Figure 2.4|: Execution workflow for a variational algorithm on a quantum-classical hybrid system.

2.3 Combinatorial Optimization

Combinatorial optimization (CO) is a sub-field of mathematics, with its main purpose being to find an optimal object from a finite set of objects [41]. These objects are often represented as variables involved in an objective function. These variables are discrete (including binary) and contain all potential combinations of a finite set of possible options. It is expected that only a subset of the possible variable values would yield a feasible solution for the optimization function. In some cases, the variables within the objective functions can have higher-order interactions, such as binary or ternary, which increases the complexity of the problem. CO has been of great interest to the scientific community as numerous real-life problems can be formulated as abstract CO problems [42]. For instance, problems such as portfolio optimization and bin packaging, can be mapped to a knapsack formulation. This is a major advantage, as addressing the subset of abstract CO problems directly facilitates solutions for a broader range of specific formulations. The formal establishment of CO as a distinct subject occurred only around 70 years ago with the rise of operations research and linear programming. However, problems of this kind have been studied since the previous century in mathematics (e.g., the Traveling Salesman problem and the four colour theorem)[43]. As the field of operations research developed, there was a growing interest in characterizing a subfield that exclusively dealt with discrete problems, as opposed to the continuous problems typically encountered in linear programming.

CO problems are commonly heterogeneous, differing in their underlying structure, objective functions, constraints, solution techniques, and, most importantly, their computational complexity. Some can be efficiently solved by algorithms in polynomial time (i.e., belonging to the complexity class P) using traditional computational resources such as CPUs and GPUs. In contrast, others prove to be particularly challenging, sometimes falling into the NP-complete complexity class, implying that no algorithm can solve the problem in polynomial time. These problems can also be referred as classically intractable, as there is no classical algorithm that provides an efficient solution. A comprehensive description and analysis of most NP-complete combinatorial optimization problem formulations is provided by Karp in [44], where he demonstrated that each of these problems can be reduced to a Boolean satisfiability problem, previously proven to be NP-complete by Cook [45]. A few examples of these challenging problems include the Knapsack problem, Max-cut, Job Sequencing, and Vertex cover, among others [46].

As polynomial or "efficient" methods for solving NP-complete combinatorial optimization problems are yet to be discovered, multiple heuristics and techniques that yield approximate solutions have been developed to address them. Among the most prevalent methods are Simulated Annealing, Genetic Algorithms, and Tabu Search. These methods are commonly implemented within high-performance solvers that are accessible to the general public for solving their own problem formulations. Notable solvers such as the Gurobi Optimizer, IBM-CPLEX, MIPCL, and FICO Xpress are state-of-the-art software solutions currently utilized by many businesses and researchers. Access to these tools is typically provided through programming libraries that are straightforward to integrate into any

optimization workflow. For example, DOcplex is a Python programming library developed by IBM that provides access to the IBM-CPLEX solver.

The following subsections provide a more detailed description of two main types of combinatorial optimization problems: *unconstrained* and *constrained*.

2.3.1 Unconstrained Optimization

Unconstrained combinatorial optimization problems are formulations for which there are no infeasible solutions. All possible combinations for the problem's variables result in a valid solution; however, not all solutions correspond to a global minimum (or maximum) [41]. Problems such as max-cut, graph coloring, or the partition problem are just a few instances of unconstrained combinatorial optimization formulations. The general mathematical formulation for these types of problems is presented in equation 2.6, where h_i is the corresponding linear term for the variable x_i.

$$\text{minimize: } f(x) = \sum_i h_i x_i \tag{2.6}$$

A special case of unconstrained optimization that is relevant to our work is quadratic unconstrained optimization (QUBO) problems. These types of formulations involve pairwise or quadratic interactions between the function variables, increasing the search space and, therefore, the computational complexity of the problem. Equation 2.7 provides the details of how the objective function of these problems is formulated. The parameter h_i is called the *bias* on the i-th binary variable, $x_i \in \{0, 1\}$, and J_{ij} is called the *coupling* between binary variables i and j. The h_i and J_{ij} therefore define the problem to be solved. Given the similarity of QUBOs to the Ising model from ferromagnetism [47], QUBOs are a crucial part of quantum annealing and, consequently, gate-based variational quantum algorithms, including VQE and QAOA. The specific details about how QUBOs are involved in these algorithms are described in chapter 3.

$$QUBO(x) = \sum_i h_i x_i + \sum_{i>j} J_{ij} x_i x_j \tag{2.7}$$

2.3.2 Constrained Optimization

Contrary to the previously described type, constrained combinatorial optimization formulations do possess infeasible solutions, implying that only a subset of all potential solutions is considered when evaluating the maximum or minimum of the given objective function. Problem formulations of this nature frequently arise in fields such as finance or logistics, where objective functions are constrained by factors such as returns, costs, fuel, or physical space. Among the most pertinent use cases are portfolio optimization, vehicle routing, and bin packing. The usual mathematical formulation for these types of problems is presented

in equation 2.8, where $f(x)$ is subject to one or more equality or inequality constraints. For the provided equation, c_i corresponds to the constraint value for variable x_i, and c_{max} is some upper or lower bound for the associated constraint.

$$\begin{aligned} &\text{minimize: } f(x) = \sum_i h_i x_i \\ &\text{subject to: } \sum_i c_i x_i \leq c_{max} \end{aligned} \tag{2.8}$$

2.4 Related Work

At the moment of writing this thesis and to the best of our knowledge, there is no other software solution specifically tailored to modeling and executing combinatorial optimization formulations through different quantum platform providers and quantum algorithms. Classical optimization packages such as the ones previously described have to integrate any support for quantum technologies. A limited set of features tailored for combinatorial optimization is available in IBM's quantum development kit (Qiskit) through their optimization module (Qiskit Optimization).[4] They provide software features to model combinatorial optimization problems, convert them into formulations amenable to gate-based quantum computers, and allow access to a library of pre-made combinatorial optimization solutions. However, there is still a large amount of work needed to orchestrate a complete workflow for solving the modeled problem, including setting up the algorithmic solution and the parsing of the results. Additionally, Qiskit Optimization only supports backends from IBM Quantum and not from other quantum computing providers. Other quantum software solutions, such as T|ket⟩ [48] from Quantinuum and PennyLane [49] from Xanadu, provide platform-agnostic capabilities that enable the execution of any quantum algorithm on different quantum devices. Nonetheless, these libraries are for general purposes and do not provide custom functionalities for modeling combinatorial optimization problems, adding a great amount of overhead for executing and experimenting with quantum combinatorial optimization solutions.

2.5 Chapter Summary

This chapter describes the background and the relevant related work for this thesis. First, we provided a comprehensive description of the field of quantum computing and then described the most important aspects within the field of combinatorial optimization. Finally, we presented a brief review of related work in order to have an overview of the current quantum combinatorial optimization software ecosystem.

[4] https://qiskit.org/ecosystem/optimization/

Chapter 3

Quantum Combinatorial Optimization

Combinatorial optimization problems such as the Knapsack problem or the Traveling Salesperson problem play an essential role within the current industrialized world. Optimizing how to load a cargo plane, finding the best route to deliver packages, and even predicting how an RNA strand will fold are just a few examples of how these problems impact our everyday lives. As mentioned in the previous chapter, many instances of combinatorial optimization formulations are known to be NP-complete, limiting or even making it unfeasible to solve large-scale problems through classical resources. The evergrowing complexity and scale of industrial, financial, and logistical processes have pushed the field of optimization to adopt heuristics to solve such problems; more often than not, these solutions tend to be mere approximations or incomplete answers to the original formulation. With these limitations in mind, QC researchers have been able to employ the particular properties of quantum mechanics to successfully develop quantum algorithms and techniques that, in principle, address these shortcomings. Among the most fundamental and innovative QC algorithmic solutions for combinatorial optimization are Quantum Annealing, the Variational Quantum Eigensolver (VQE), and the Quantum Approximate Optimization Algorithm (QAOA). Even though there are still quantum hardware limitations, these techniques have shown, in some instances, to provide an advantage compared to classical solutions. The proposed software solution for this thesis integrates the aforementioned algorithms and allows the user to select the most appropriate for the given problem formulation. Hence, it is crucial for the reader to fully grasp the inner workings of these approaches. This chapter describes the three approaches to solving a combinatorial optimization problem with a quantum computer while highlighting their benefits and limitations.

3.1 Quantum Annealing

Quantum Annealing (QA) is a technique developed with the goal of finding the ground state of a given problem Hamiltonian with high accuracy in a short amount of time [50]. This method was inspired by a classical optimization heuristic called Simulated Annealing (SA), which uses thermal fluctuations to push a system towards its global minimum [51]. Analogously, QA uses quantum tunneling processes to help converge the optimization process. More specifically, this method employs the principle of adiabatic evolution, described in section 2, to optimize an energy function. This function is given in the form of

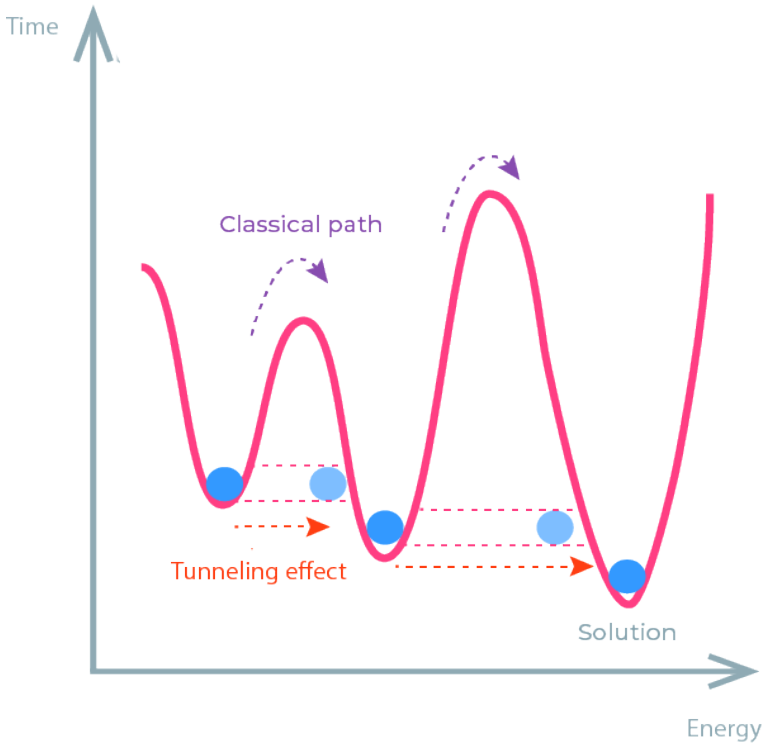

Figure 3.1|: The effect of quantum tunneling of Quantum Annealing at finding the global minimum compared to the adiabatic evolution of Simulated Annealing.

a problem Hamiltonian (H_p) whose ground state encodes the solution to the combinatorial optimization problem.

QA presents a clear advantage over SA in being able to find the lowest energy level for the specified function. In many cases, the thermal jumps from the adiabatic evolution performed in SA cannot surpass large barriers presented in the energy landscape, causing the algorithm to get stuck at local minima. Meanwhile, the quantum tunneling effects or fluctuations from QA can penetrate these barriers if they are thin enough, allowing this method to reach the better solution for a given problem formulation [52]. This process is illustrated in Figure 3.1.

The practicality of QA towards solving multiple types of combinatorial optimization problems comes from the ability of these types of problems to be represented as QUBO formulations, which in turn can be used to find a corresponding Ising Hamiltonian that can be used as H_p. More specifically, any given cost function $C(x_1, ..., x_N)$ in the form of a QUBO formulation can be expressed as a Hamiltonian H_c in Ising form [1]:

$$E_{Ising}(Z_1, ..., Z_N) = \sum_{i,j=0}^{N} J_{ij} Z_i Z_j + \sum_{i=0}^{N} h_i Z_i \tag{3.1}$$

where Z_i corresponds to discrete spin variables, subject to $Z_i \in \{-1, 1\}, \forall_i \in N$. Additionally, J_{ij} and h_i define the quadratic and linear coefficients of the function, respectively.

This mapping process has allowed QA providers, such as D-Wave, to develop user-friendly SDKs that allow for the definition of QUBOs for optimization problems that are then automatically converted into Ising Hamiltonians and fitted into the QPU without any programming overhead for the user. Due to the simplicity of this approach, numerous studies and publications have extensively applied QA to various fields and application areas, such as bio-informatics [53], manufacturing [54], logistics [55] and machine learning [56].

Besides offering a more straightforward and intuitive approach to solving combinatorial optimization problems, QA possesses additional advantages that contribute to its status as the most widely employed quantum algorithm for optimization. One of these benefits is the ease of scalability for quantum annealers, compared to their gate-based counterparts, enabling the implementation of larger systems capable of handling more complex and sizable problem instances [38]. However, the limited connectivity between the qubits in a quantum annealer presents a limit to the size and types of problems that can be addressed with these machines. Another advantage is the reduced scaling of the execution time for the size of the problem instance, which arises due to the time evolution process being regulated by a constant annealing time (t) provided by the user [24]. This means if the size of the problem doubles the increase in execution time is not going to be doubled but increased in a smaller factor, specified by the user to maintain the quality of the solution.

Nevertheless, there are still some drawbacks that affect QA and, in some cases, limit its usability. Firstly, most QA providers have a customized and restrictive implementation of the technique, allowing users to modify only a limited set of characteristics, such as the number of shots and annealing time. This constraint can limit the experimentation and evaluation of QA for complex use cases. Additionally, certain studies [1] have demonstrated that QA applied to large and complex problems may encounter challenges in finding the global minimum of the energy landscape compared to classical methods. Consequently, this can lead to the generation of sub-optimal results. This situation is illustrated in Figure 3.2.

Overall, QA is a highly practical and powerful approach to solving optimization problems. It stands as the most widely employed and researched quantum technique for addressing these types of problems. However, like any other quantum computing solution, it has flaws. Some progress is still needed before it can be extensively utilized for real-world, large-scale industrial applications.

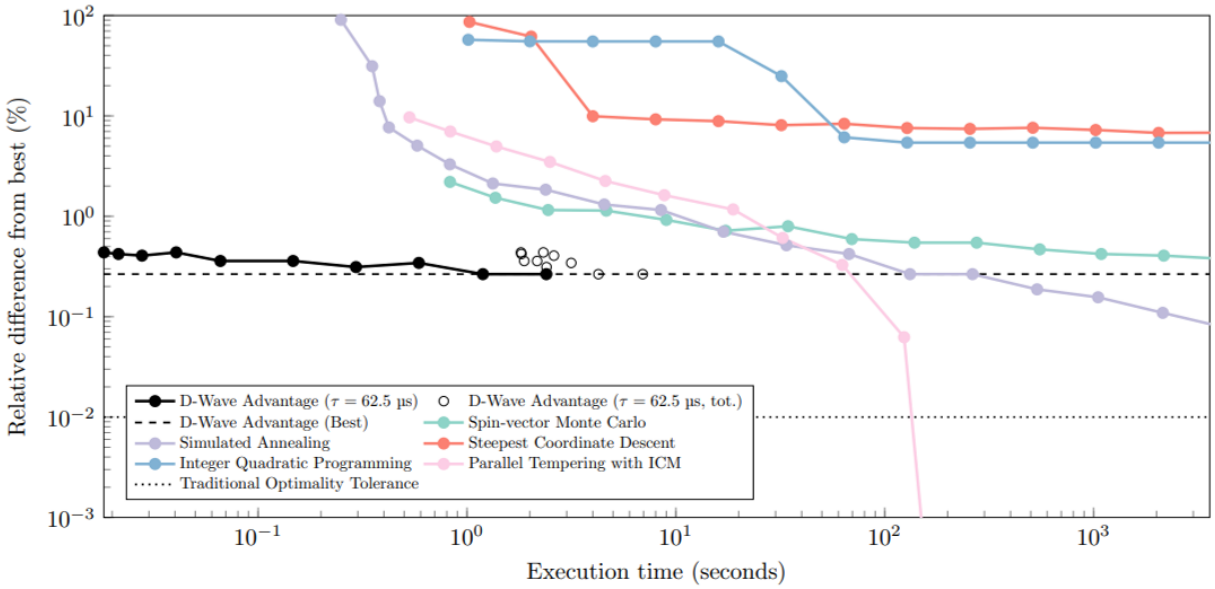

Figure 3.2|: Solution quality of multiple optimization techniques when solving a 5,387 decision variables problem. It is shown that QA is not able to converge toward the best-known solution. Taken from [1].

3.2 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) is a hybrid quantum-classical algorithm designed explicitly for gate-based quantum computers. Its primary objective is to approximate the eigenvalue of a given eigenvector by computing the lowest energy of the system [14]. This eigenvalue represents the ground state energy of a system, which corresponds to the most optimal configuration of that system. The VQE approach is inspired by the variational principle of quantum mechanics. Which states that the ground state energy (E_ϕ) of an observable described by a Hamiltonian $\hat{H}$ and a trial wave function $|\psi_\phi\rangle$ is always higher than or equal to the ground state energy (E_0) of the same Hamiltonian evaluated using an exact wave function $|\psi_0\rangle$ [57], as defined in equation 3.2.

$$E_0 = \frac{\langle\psi_0|\hat{H}|\psi_0\rangle}{\langle\psi_0|\psi_0\rangle} \quad E_\phi = \frac{\langle\psi_\phi|\hat{H}|\psi_\phi\rangle}{\langle\psi_\phi|\psi_\phi\rangle} \quad \text{fulfilling} \quad E_\phi \geq E_0 \tag{3.2}$$

Hence, the goal of VQE is to find the viable parameterized trial state that minimizes the ground state energy of the system. It is possible to characterize the VQE optimization process as described in equation 3.3. Where $|\psi_\phi\rangle$ is expressed as a parameterized unitary $U(\theta)$ applied to an N-qubit initial state ($|0\rangle^{\otimes N}$) [58]. $U(\theta)$ is also known as an Ansatz or an informed guess for the trial state, and it is composed of different parameterized quantum gates that will be updated by computing new parameters through the multiple iterations of a classical optimization algorithm.

$$E_{VQE} = \min_\theta \langle 0|U^\dagger(\theta)\hat{H}U(\theta)|0\rangle \tag{3.3}$$

The complete algorithmic approach for VQE is described in Figure 3.3. The pseudocode describes how the QPU and the CPU interact with each other to approximate the minimum eigenvalue of the operator $\hat{H}$. It is important to note that a crucial step before executing the algorithm is to define the variational Ansatz ($U(\theta)$); this step is analogous to designing the architecture of a neural network, as the structure of this trial state is going to determine the performance and accuracy of VQE. Choosing the correct Ansatz is not an easy task; many papers have studied how different choices affect the results, and some have even proposed certain ansatz for specific types of problems. For instance, Liu et al. [2] present a layered Ansatz that has proven to be adequate when dealing with combinatorial optimization problems; the specific structure of the quantum circuit is presented in Figure 3.4. Nevertheless, understanding and selecting the most appropriate trial state for different variational quantum algorithms is still a very active area of research [59] [60] [61].

The chosen Ansatz is used in the quantum subroutine of the algorithm to compute the expectation value ($E(\theta)$) of $\hat{H}$ evaluated on $U(\theta)$, as described in equation 3.4. This value is of utmost importance as it characterizes all the possible outcomes of a measurement weighted by their probabilities, which gives a sense of how well the calculation of the

Algorithm 1 Variational Quantum Eigensolver (VQE)

Require: Hamiltonian H, Variational Ansatz U, Initial parameter values θ
Ensure: Final parameter values θ and corresponding minimum energy $E(\theta)$

1: **Initialization**: Set convergence criteria, e.g., desired energy threshold, maximum number of iterations
2: Initialize iteration counter: $iter \leftarrow 0$
3: **repeat**
4: Increase iteration counter: $iter \leftarrow iter + 1$
5: Apply Variational Ansatz U with current parameter values θ to a quantum state $|\psi(\theta)\rangle$
6: Compute expectation value $E(\theta) = \langle\psi(\theta)|H|\psi(\theta)\rangle$ using quantum measurements or simulations
7: Update parameter values: $\theta \leftarrow Optimize(E(\theta), \theta)$
8: Check convergence criteria
9: **until** convergence or maximum # of iterations reached

Figure 3.3|: Pseudo-code for the VQE algorithm.

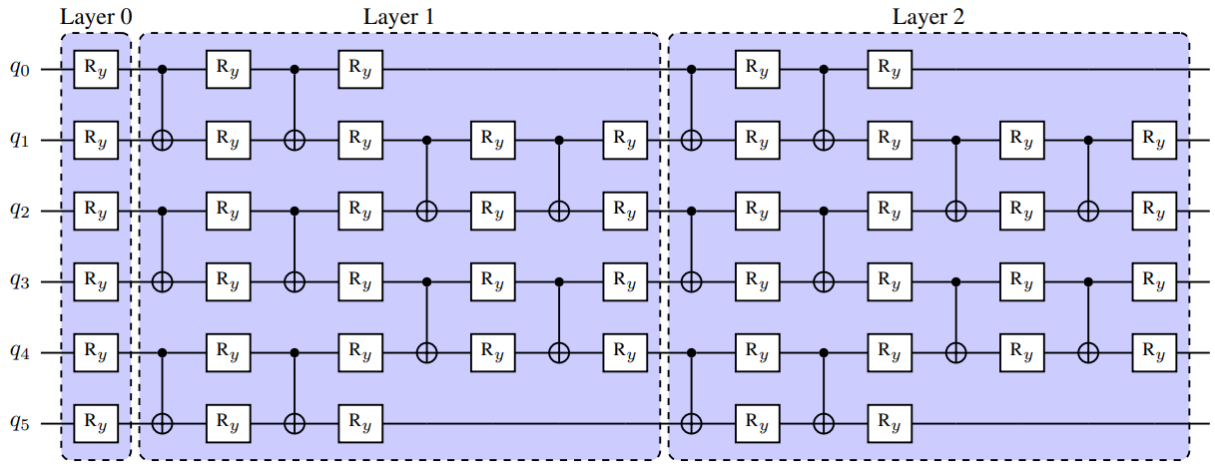

Figure 3.4|: Structure of a 6-qubit layered Ansatz suited for combinatorial optimization on noisy quantum computers. Extracted from [2].

minimum eigenvalue approximates the ground truth. In a sense, this step can be seen as the computation of the cost for the current configuration of θ evaluated on the Ansatz. Once $E(\theta)$ has been calculated, the classical subroutine of the hybrid algorithm is in charge of updating θ, so equation 3.4 will be minimized. This hybrid process between the two devices is repeated iteratively until either the expectation value stops changing or the maximum number of iterations is achieved; this approach is depicted in Figure 3.5. When the algorithm has finished its execution, the optimal values for θ are then used to do a final evaluation of $E(\theta)$. The result of this computation is going to be the lowest approximate eigenvalue of our problem, Hamiltonian.

$$E(\theta) = \langle U(\theta)|\hat{H}|U(\theta)\rangle \tag{3.4}$$

Since its inception, VQE has been primarily applied to quantum chemistry applications. However, numerous publications have utilized this approach for solving combinatorial optimization problems and have reported promising results [2] [31] [62]. VQE's applicability to these types of problems stems from the same procedure employed in Quantum Annealing (QA) for optimization. More specifically, the ability to map an optimization problem expressed in QUBO form to a corresponding Ising Hamiltonian $\hat{H}$, which ultimately will become the operator whose minimum eigenvalue is to be calculated. Similar to QA, the approximated ground state energy obtained from VQE also encodes the solution to the formulated optimization problem.

VQE is an algorithm known for its remarkable flexibility and wide applicability across various domains, as mentioned earlier. One of its key strengths lies in its high level of customization. The choice of Ansatz can be tailored and adjusted to suit specific use cases or problem characteristics. Furthermore, the classical optimization routine within VQE can be implemented using various optimization algorithms, such as COBYLA, ADAM, and SPSA, among others [63], providing further flexibility. Another significant advantage of VQE is its hybrid nature, enabling its utilization on near-term quantum devices while still producing

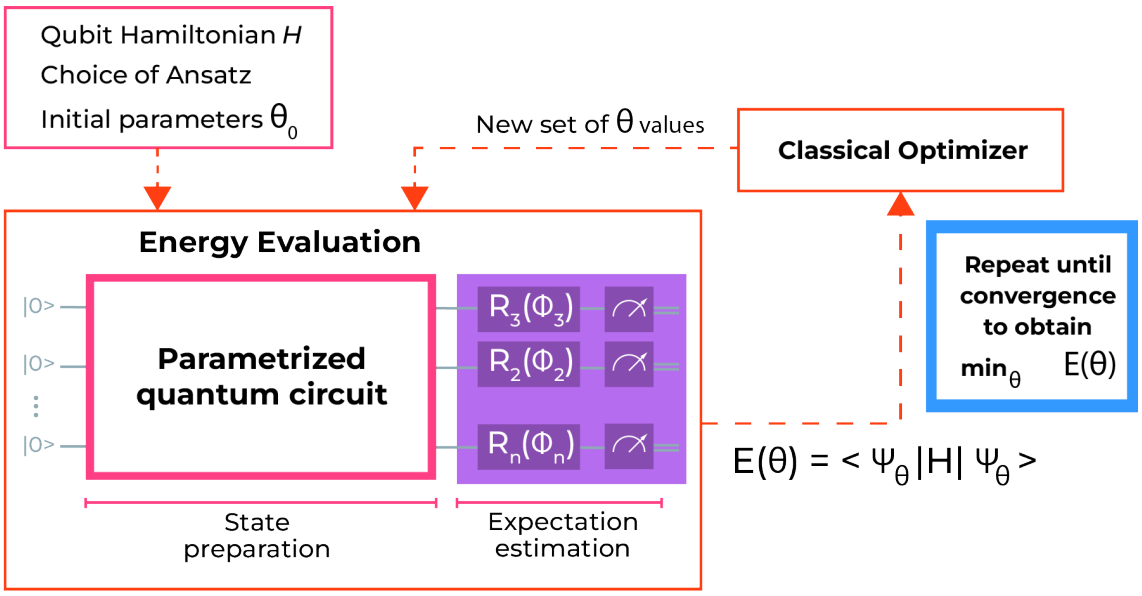

Figure 3.5 | : Overview of the variational approach followed by VQE.

valuable results. Additionally, leveraging specialized hardware, such as GPUs or TPUs for faster gradient computation, can accelerate the parameter optimization process, presenting an opportunity to gain efficiency from the classical side. Furthermore, the availability of a more extensive selection of gate-based devices from quantum providers, in comparison to Adiabatic Quantum Computing (AQC), offers more options to experiment with different hardware implementations [64]. This enables researchers to evaluate and identify the most suitable hardware configuration for specific problem domains, enhancing the exploration of potential quantum solutions.

When considering the limitations of VQE, two factors currently constrain its applicability and accuracy. Firstly, the scale of most general-purpose quantum computers restricts the size of problems that can be effectively addressed using VQE. Many optimization problems involve a significant number of variables, often reaching the order of hundreds or thousands, rendering VQE feasible for only a small subset of problem instances. Furthermore, the presence of high levels of noise in near-term devices, coupled with their limited qubit connectivity, negatively affects the algorithm's accuracy. Secondly, the variational approach employed by VQE is prone to becoming trapped in local minima, leading to suboptimal solutions. While specific papers [65] [66] have proposed potential improvements to alleviate this challenge, it remains an ongoing issue that has yet to be resolved entirely.

In conclusion, VQE is a foundational and groundbreaking quantum algorithm; its hybrid approach paved the path towards practical near-term quantum computing. It has multiple benefits that make it useful for a wide array of problems, including combinatorial optimization. However, being a relatively new algorithm, VQE is not without its limitations, particularly when it comes to scaling up to more significant and more intricate problem instances. Further research and development are imperative for VQE to fully realize its potential and offer tangible advantages in the field of optimization.

3.3 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical algorithm specifically designed to produce approximate solutions to combinatorial optimization problems [15]. QAOA follows a similar approach to VQE, as it also works by searching for the eigenstate that minimizes the eigenvalue of a given system in the form of a Hamiltonian $\hat{H}$. The ground state energy of $\hat{H}$ is also approximated by evaluating the operator on a parameterized Ansatz and optimizing the set of parameters in a variational fashion. Figure 3.6 presents the pseudocode of this algorithm. It could be said that QAOA is a more specific version of VQE, as the main difference between these two algorithms is that QAOA uses a problem-specific Ansatz. This parameterized quantum circuit is constructed using an initial N-qubit superposition state ($|+\rangle^{\otimes N}$) and on two layers, a cost layer, which encodes an approximation of the problem Hamiltonian (H_C), and a mixer layer, which represents a transverse field Hamiltonian (H_M), similar to the one used in QA. Both of these layers are repeated a p number of times to improve the level of approximation of the algorithm. The use of a large value for p comes with a trade-off as this increases the circuit depth and, therefore, the execution time. This Ansatz structure is depicted in Figure 3.7.

A crucial step in the QAOA algorithm is the encoding of the problem Hamiltonian into the cost layer. This process is performed by exponentiating H_C; this operation allows the discrete spin variables from the Ising formulation to be mapped into quantum gates that can be appended to our layer. For a given problem Hamiltonian H_C, its corresponding exponentiation is given by equation 3.5.

$$\begin{aligned} H_C &= \sum_{i,j=0}^{N} J_{ij} Z_i Z_j + \sum_{i=0}^{N} h_i Z_i \\ e^{-i\gamma H_c} &= \prod_{i,j=0}^{N} R_{Z_i Z_j}(J_{ij}\gamma) \prod_{i=0}^{N} R_{Z_i}(h_i \gamma) \end{aligned} \tag{3.5}$$

By examining the equation, we can observe that the quadratic terms of the problem Hamiltonian are encoded in the cost layer through the application of R_{ZZ} rotations. Similarly, the linear terms are represented by R_Z gates. Each of these gates relies on variational parameters (γ) and a corresponding coefficient derived from either the quadratic or linear terms of H_C.

The inclusion of a transverse field Hamiltonian, represented by H_M, is crucial in enabling QAOA to converge towards the global minimum of the system. This Hamiltonian plays a similar role to the "easy" Hamiltonian used in quantum annealing as it provides a known ground state to begin the exploration process. To incorporate H_M into the Ansatz, a similar exponentiation process as with H_C is performed, but this time using single-qubit Pauli X operators to represent the Ising formulation:

Algorithm 2 Quantum Approximate Optimization Algorithm (QAOA)

Require: Problem Hamiltonian H_C, Mixer Hamiltonian H_M, Number of layers p

Ensure: Approximate solution to the optimization problem

1: **Initialization**: Set initial parameter values for γ and β
2: Initialize quantum state $|\psi\rangle$ randomly
3: Apply Hadamard gates to all qubits in $|\psi\rangle$
4: **for** k in range(p) **do**
5: Set parameters γ_k and β_k
6: Apply $e^{-i\beta_k H_M}$ to all qubits in $|\psi\rangle$
7: Apply $e^{-i\gamma_k H_C}$ to all qubits in $|\psi\rangle$
8: Measure the qubits in $|\psi\rangle$ and obtain a classical bitstring
9: Update parameters γ_k and β_k based on the optimization objective and classical measurement results
10: **end for**
11: **return** Classical bitstring as the approximate solution

Figure 3.6|: Pseudo-code for the QAOA.

$$
\begin{aligned}
H_M &= \sum_{i=0}^{N} X_i \\
e^{-i\beta H_M} &= \prod_{i=0}^{N} R_{X_i}(2\beta)
\end{aligned}
\tag{3.6}
$$

Similarly to the previous exponentiation, Equation 3.6 demonstrates that the mixer Hamiltonian is encoded by applying parameterized R_X gates to each qubit in the Ansatz. These rotations also depend on variational parameters (β) that are distinct from the ones employed in the cost layer.

QAOA is often regarded as the gate-based counterpart to QA, employing a similar approach to approximate the optimal solution of a combinatorial optimization problem. Inspired by QA, QAOA requires the discretization of the time-dependent evolution of H_C to accommodate gate-based hardware [67]. This discretization process is possible thanks to the Trotter-Suzuki formula [68], a fundamental component of quantum simulation, it allows to approximate the exponential of a sum of non-commuting operators (e.g., $\hat{A}$ and $\hat{B}$) into a discrete number of steps (M), usually present during adiabatic evolution. As shown in equation 3.7.

$$
e^{\hat{A}+\hat{B}} = \lim_{M\to\infty} (e^{\frac{\hat{A}}{M}} e^{\frac{\hat{B}}{M}})^M \tag{3.7}
$$

As an algorithm primarily developed for solving combinatorial optimization problems, QAOA has been widely studied and tested across various application areas [32] [69]

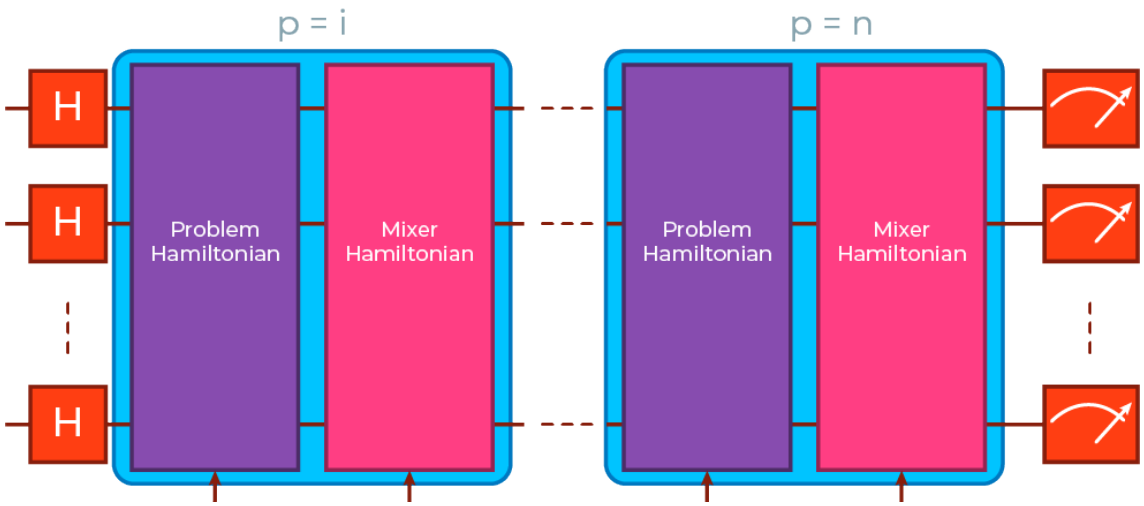

Figure 3.7|: Variational Ansatz for QAOA, depicting the problem and mixer layers.

[70]. This extensive research has been instrumental in gaining insights into the limitations of QAOA and has motivated the development of improved versions of the algorithm to achieve better approximations. For example, the Quantum Alternating Operator Ansatz (QAOAnsatz) [71] allows for alternation between more general families of operators: phase-separation operators, which depend on the objective function to be optimized, and mixing operators, which depend on the problem's domain and structure. This flexibility allows the Ansatz to go beyond the constraints of parameterized unitaries for a fixed local Hamiltonian, making it better suited for optimization problems with hard constraints. Another notable improvement from the base implementation is the Warmstart QAOA [72]. This approach enables the algorithm to leverage a pre-trained set of parameters obtained from the classical solution of a relaxed version of the optimization problem. The main idea behind this method is that starting QAOA from a better initial point can lead to faster convergence with fewer iterations. The expectation is that the time saved during the execution of the algorithm outweighs the time spent on pre-calculating the optimal starting point.

QAOA has multiple benefits that make it a prime candidate for demonstrating quantum advantage for combinatorial optimization problems with near-term devices. One of its most significant advantages is its ability to adapt to specific problems. As mentioned before, QAOA utilizes an Ansatz that encodes the cost function of the optimization problem and provides the option to use more general operators. This flexibility allows QAOA to handle complex constraints and problem structures, making it well-suited for a wide range of optimization problems. Another advantage of QAOA is its hybrid nature, which allows for the optimal utilization of current limited quantum computing resources in combination with classical optimization subroutines. As quantum computers continue to improve exponentially year after year, the performance of the algorithm is expected to undergo significant improvements. This means that as better hardware becomes widely available, the speed and accuracy of QAOA will naturally increase, leading to improved optimization results.

This algorithm, being very similar to VQE, also shares many of its limitations. Mainly, it is affected by the current state of gate-based quantum devices in terms of scale (number

of qubits), quality (level of noise), and connectivity. These factors have led to a subpar performance of QAOA compared to QA and even classical heuristics when studying it for large problem instances [73]. Additionally, as the name of the algorithm indicates, this method provides an approximation for the optimal solution. It is not guaranteed that QAOA will find the optimal solution for a problem. Even if the level of accuracy can be controlled by increasing p, it would be necessary to have infinite iterations in order to produce a perfect approximation to the true solution.

Finally, QAOA is a promising quantum algorithm that has gained significant interest and witnessed rapid advancements since its initial proposal. Its applicability to various fields of high interest has motivated active research aimed at overcoming limitations encountered when applying QAOA to large-scale and complex optimization problems. Moreover, the evolving landscape of quantum computing hardware provides promising prospects for the future development and application of QAOA. As a result, it is crucial to continue investing resources and efforts into research associated with this approach to further enhance its capabilities and explore its full potential.

3.4 Chapter Summary

This chapter describes in detail the three most important quantum algorithms and methods for quantum optimization: QA, VQE, and QAOA. We presented the motivation, implementation, and mathematical foundations for each of these. Additionally, the benefits and limitations of each approach are highlighted, taking into account their applicability for large-scale optimization problems and the current state of quantum hardware and software platforms.

Chapter 4

Quantum Computing Hardware and Platform Providers

The current ecosystem of quantum hardware is extensive, with over 20 quantum computing companies (i.e., quantum platform providers) offering access to their quantum computers. These devices vary in terms of qubit implementations, computing paradigms, and even SDKs, posing a challenge when selecting the most suitable platform for specific applications, particularly in the case of complex applications like quantum combinatorial optimization. This chapter aims to provide a comprehensive description of the most prevalent quantum hardware implementations and discuss a set of aspects that can characterize quantum computing platforms, allowing for a more informed evaluation of their capabilities. Finally, the three quantum providers available in QPLEX will be described, and the main characteristics of their devices will be presented. Similar to Chapter 3, a deep and clear understanding of the technologies and approaches described in this section of the thesis is paramount to fully exploiting the platform-agnostic capabilities of QPLEX.

4.1 Hardware implementations

The invention of quantum computing hardware is one of the most remarkable achievements of the last century. The ability to interact and perform computations on subatomic quantum mechanical systems is genuinely extraordinary. However, the development of quantum machines is a complex and challenging endeavor. It took scientists and engineers over 20 years to realize a functional prototype. Motivated by this significant milestone and driven by the pursuit of the most optimal way to build a quantum computer, multiple hardware implementations have been developed, each with unique characteristics and potential advantages. These implementations refer to the methods used to represent qubits within the quantum processing unit (QPU). The three most common qubit implementations are superconducting qubits, ion traps, and photonic qubits. The following subsections will describe and analyze each of these implementations in detail.

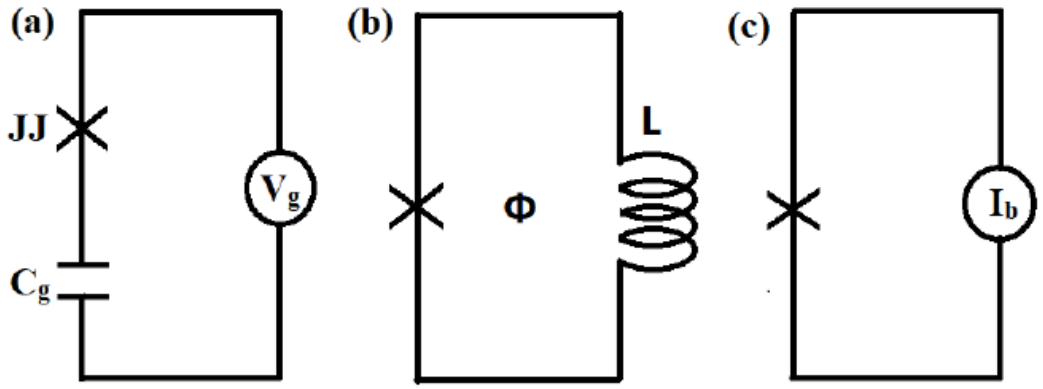

Figure 4.1|: Superconducting qubit circuit diagram. (a) Charge qubit. (b) Flux qubit. (c) Phase qubit. Taken from [3].

4.1.1 Superconducting

Superconducting quantum computing (SQC) is the most popular implementation of quantum computing hardware. Companies such as IBM, D-Wave, Rigetti, and Google are heavily investing in developing large-scale quantum computers based on this technology. This choice is driven by the complexity of building physical qubits from quantum particles (e.g., electrons and photons), as their small scale makes classical manipulation of these systems difficult. To overcome this challenge, SQC proposes using macroscopic systems that can behave as quantum particles and be easily manipulated for computations. The original idea for this method was presented back in 1999 [74], where the authors proposed the use of a superconducting circuit called a single-Cooper-pair box, in combination with a Josephson junction—a sheet of insulating material positioned between two superconducting plates. This setup creates an artificial two-level system capable of exhibiting superposition, also known as a charge qubit. Building upon the original implementation, two other types of superconducting qubits were developed: flux qubits [75] and phase qubits [76]. These three archetypes can be characterized by their method of controlling the transition between quantum states [3]. The charge qubit achieves this by controlling a voltage V_g, the flux qubit performs the transition by changing the bias flux Φ, and the phase qubit achieves it by adjusting the bias current I_b in the circuit. These implementations are depicted in Figure 4.1.

Within these qubit archetypes, there are more specific implementations that address different limitations of the original designs. Some notable examples include the Transmon qubit [77], the Fluxonium qubit [78], and the recently discovered Unimon qubit [79]. Regardless of the qubit type, all these quantum systems must interact with classical devices to receive instructions. Microwave pulses typically facilitate this interaction in superconducting quantum computers (SQC). These pulses are applied to the qubits using specialized devices that are coupled to the circuit, allowing for the manipulation of the system's state. When it comes to measuring the quantum states or performing qubit read-out, different approaches can be employed. Standard techniques include charge measurement, flux measurement, and inductance measurement. However, the most widely used process is dispersive read-

out. This method involves measuring the transmission coefficient of a readout resonator, which is also coupled to the circuit through capacitance or inductance [3]. By analyzing the changes in the transmission coefficient, the qubit's state can be detected and read out.

SQC offers two main advantages compared to other quantum hardware implementations: its fabrication process and the method of interacting with qubits. Firstly, SQC utilizes a macroscopic solid-state electrical circuit as its qubit implementation. This similarity to classical semiconductor microfabrication processes allows faster development and greater scalability. Secondly, using microwave pulses to manipulate the quantum state of the systems is highly advantageous. Commercial microwave equipment can be readily employed for this purpose, eliminating the need for specialized hardware development. This reduces costs and accelerates the implementation of more complex quantum systems. Nonetheless, a significant drawback poses a constraint on this type of implementation: the requirement to maintain these systems at extremely low temperatures to preserve the superconducting properties of the materials used in the circuits. To achieve and sustain the necessary temperature, the systems must be installed within a dilution fridge, a large and expensive apparatus explicitly designed for this purpose. Including this additional hardware significantly increases the implementation cost and complexity of superconducting quantum computers.

4.1.2 Trapped Ion

Even though Trapped Ion Quantum Computing (TIQC) is not as common as Superconducting SQC, it presents a promising and practical approach to building quantum computing hardware. Two major players, IonQ and Quantiniuum, are actively developing TIQC devices. These companies have successfully realized working QPU implementations using ion trap technology and provide access to these devices through cloud services like AWS Braket. This technique for building qubits was first proposed in 1995 by Ignacio Cirac and Peter Zoller. They demonstrated that a collection of cold ions manipulated with laser light and confined in a Paul trap, an electromagnetic confinement device, could serve as a realistic physical system for realizing a quantum computer [80]. However, not all ions can be used as qubits. The energy levels that characterize the ion determine the practicality and feasibility of preparing and manipulating a quantum state within itself. Generally, ions in Group II of the periodic table are good choices for qubit development due to the multiple states of the valence electrons in their atoms [81]. This flexibility allows for various qubit implementations within ion traps, such as Zeeman qubits, hyperfine qubits, optical qubits, and fine-Structure qubits. Each of these implementations specifies a way to represent the computational basis of the qubit and defines the methods for manipulating the quantum states.

Optical qubits and hyperfine qubits are two popular implementations of trapped ion qubits due to their efficiency and practicality. While both schemes utilize the ground state ($|g\rangle$) and excited state ($|e\rangle$) to represent the computational basis, they differ in the methods of transitioning between these states and applying quantum gates. Both approaches

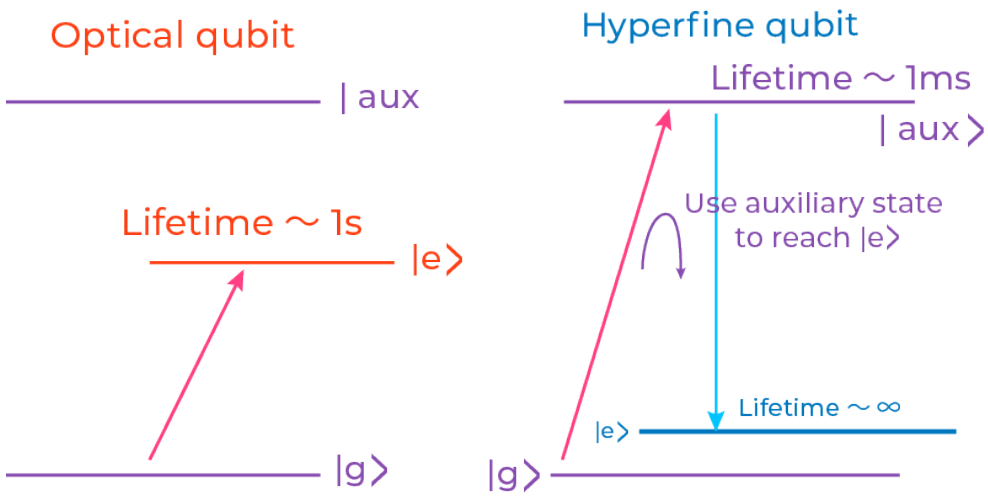

Figure 4.2|: Representation of state transition in Optical and Hyperfine qubits. Extracted from [4].

are illustrated in Figure 4.2. In the case of optical qubits, the transition from $|g\rangle$ to $|e\rangle$ can be achieved directly by exciting the system with a photon. The energy difference between the ground and excited state corresponds to a laser frequency of $729nm$, which can be generated using an infrared laser [4]. Similarly, a single resonating laser can apply quantum computing gates to optical qubits. On the other hand, hyperfine qubits cannot make the same jump, as the energy difference between the states is minimal; instead, an auxiliary state ($|aux\rangle$), reachable by an infrared laser, is used to transition the system to a higher energy state, in that point the coherence time of $|aux\rangle$ is short which causes the system to converge to $|e\rangle$. This setup allows for longer coherence times compared to optical qubits. Additionally, the execution of quantum gates on hyperfine qubits can be performed through microwaves or laser light, allowing for more flexibility for the hardware implementation.

Overall, TIQC is a promising technology for implementing quantum computers. Using quantum physical systems such as ions presents several advantages compared to other techniques. One advantage is the robustness of ion energy levels, which contributes to longer qubit coherence times. This improved coherence allows for the application of more sequential operations to these systems. Additionally, TIQC enables the implementation of single and two-qubit gates with high fidelity, thanks to the precision of lasers and the confinement of ions within the linear electromagnetic trap. However, a critical limitation of this approach is the slow time for applying quantum gates to the qubits. According to [81], two-qubit gates implemented on trapped ions have been demonstrated to be as fast as $1.6\mu s$, while two-qubit gates performed in superconducting qubits are in the order of nanoseconds.

4.1.3 Photonic

Photonic Quantum Computing (PQC) has seen a significant increment in research and development over the past decade. More specifically, Linear Optical Quantum Computing

(LOQC), a specific implementation of PQC, has become the most promising option for implementing a fully scalable and fault-tolerant quantum computer. Research labs and leading QC companies such as Xanadu and PsiQuantum are investing many resources to realize this ambitious vision. Xanadu, in particular, has produced multiple papers and reports showcasing their advancements [11][82], in addition to already providing cloud access to their state-of-the-art X-series nanophotonic chip [83].

The fundamental idea behind PQC is encoding qubits on single photons; given that this quantum particle does not easily interact with its environment, it makes for a very robust system. The encoding process itself can be achieved through many methods, including polarization, time bin, and path [84]. This is advantageous as one-qubit gates can be easily applied; for instance, a polarization-encoded qubit can be rotated through wave plates to execute a quantum gate. However, photons' non-interacting nature makes it difficult to implement entanglement operations, such as multi-qubit gates, which hinders the building of a universal quantum computer. Two formulations for universal quantum computation with these devices were proposed as a tool for overcoming this limitation: LOQC using the KLM protocol and Continuous-variable quantum computation (CVQC).

In 2001, Knill, Lafamme, and Milburn proposed the KLM protocol, which demonstrated that it is possible to implement reliable quantum algorithms using single photon sources, passive linear optics, and photo-detectors [85]. They achieved this by using ancillary photons, beam splitters, and photon detection to create a controlled-NOT gate. This first result was fundamental for establishing linear quantum optics as a feasible framework for achieving efficient quantum information processing. However, it has since been shown that this technique would require a large number of ancillary photons for large-scale quantum devices, making the system more complex and driving up fabrication costs. Following the contributions of KLM, four years later, Braunstein and van Loock explored the applicability of continuous-variable (CV) approaches for quantum information processing through the use of Gaussian states and operators [86]. The proposed implementation differs from traditional LOQC methods in that the qubits are encoded using the continuous quadrature amplitudes of the quantized electromagnetic field present in light rather than using single photons. This allows for an efficient implementation of all the necessary steps in quantum protocols, such as state preparation, manipulation, and measuring. Additionally, the entanglement process for CV qubits can be produced using squeezed light and linear optics. One major limitation of this approach is the low fidelity of multi-qubit gates, caused by the finite limit for squeezing light; after a few steps, this process destroys the information present in CV qubits [87]. Given the constraints present in both formulations, many photonic quantum hardware companies adopt a hybrid approach that leverages the advantages of each approach while mitigating their respective limitations.

4.2 Characterization aspects

The task of determining which quantum computing provider is the most appropriate for a specific use case or application is very complex. There are many factors that play an impor-

tant role in this selection process. Some of these are related to hardware characteristics, while others are subject to software features. Currently, there is no one metric or test that can provide a definitive solution to this situation. However, a large number of papers have discussed techniques for evaluating different aspects of quantum hardware, providing helpful insight for arriving at a final decision. This chapter will introduce and describe three different characterization aspects to provide guidelines for evaluating the feasibility of using a given quantum provider for a specific combinatorial optimization problem. This will allow practitioners to minimize the experimentation time and reduce the cost generated by trial and error approaches.

4.2.1 Scale

As previously discussed, the basic unit of computation for quantum computers is the qubit. Hence, the scale of these machines is determined by the number of qubits present in their QPUs. Naturally, the more qubits a quantum computer has, the better it is equipped to solve larger and more intricate problems. In fact, it is common to measure the progress and success of a quantum platform company by the number of qubits its largest quantum computer has. Most quantum computing providers possess a wide array of machines, each with a different number of qubits and other hardware specifications. However, for the purpose of this chapter, we will only concern ourselves with the quantum computers that have an equal or larger number of qubits than the ones needed to represent a given combinatorial optimization problem. This condition is essential because if the problem cannot be encoded into the QPU, other hardware characteristics, such as gate fidelity or decoherence time, are irrelevant as they will not be used.

The number of qubits needed for encoding a problem instance is given by the total number of variables needed to construct its corresponding QUBO formulation. If the combinatorial optimization problem is composed of only binary variables and does not have any constraints, then building a QUBO that represents it is straightforward. This is the case as each item (i.e., vertices, edges, etc.) in the problem can be represented by a single variable, and their properties and relationships are captured in the QUBO matrix itself. This is not the case when working with constrained combinatorial optimization problems or instances where discrete or continuous variables are present. For these types of problems, encoding techniques are needed to transform constraints into penalties and non-binary variables into binary. More often than not, this process requires the use of extra qubits in order to represent the additional information. Many encoding techniques are present in the literature, each with different advantages and limitations [88][89]. However, at the moment of writing this thesis, QPLEX uses one-hot encoding for handling non-binary variables and constraints. Therefore, it is advisable that when applying this approach to the characterization of a quantum computer, this specific encoding method is used to find an approximation to the total number of qubits needed.

4.2.2 Quality

The quality of the results from a quantum computer is objectively the most relevant metric from these devices. Even if the quantum machine is capable of executing an algorithm 10 times faster than a classical computer, it is not useful if the results are not accurate or if it cannot successfully solve the formulated problem. This is a major problem since, as previously stated, near-term quantum computers have a limitation when it comes to producing quality results for complex quantum circuits, usually needed when executing variational quantum algorithms. Intrinsic aspects of the QPU's hardware implementation, such as decoherence time, gate fidelity, and measurements fidelity, are essential properties that need to be evaluated when selecting a quantum provider and, more specifically, a quantum computer [90].

The aforementioned properties can be considered low-level, as these are directly related to the lowest layer in the QC stack. For instance, decoherence time refers to the time a given qubit in a quantum computer can preserve its state without losing any information. On the other hand, gate fidelity concerns with the closeness between the state of a physical qubit after applying a quantum gate and the ideal theoretical case. Similarly, measurement fidelity is related to how close the physical qubit's measurement is to the ideal outcome. Generally, a gate and measurement fidelity above 99% is necessary to achieve fault-tolerant quantum computing. It is possible to go up in the abstraction level of the quantum computing stack and use metrics that encapsulate the properties in the lower levels. One such metric is quantum volume; it quantifies the largest random circuit of equal width and depth that a quantum computer can successfully implement [91]. This evaluation method is designed to strictly depend on the lower-level characteristics of a quantum device. Hence a machine with a higher decoherence time, gate fidelity, and measurement fidelity is expected to achieve a larger quantum volume. This metric is currently used as an industry standard to measure the performance of gate-based quantum devices and to track the overall progress toward fault-tolerant quantum computing.

One evident limitation of the quantum volume metric is that it can only be used for assessing the performance of gate-based devices. Consequently, it cannot be utilized for drawing comparisons between other QC paradigms, such as annealing. One can address this situation by moving even higher into the QC stack and getting into the realm of quantum applications. At this level, the complete capabilities of a quantum computer and all its attributes are evaluated on how accurately the machine can solve a given problem. Some proposals from the literature are to use quantum chemistry [92] and optimization [93] applications to benchmark different quantum devices, regardless of their paradigm or hardware implementation. These studies have shown promising results; however, setting up a complete application in order to determine the better quantum device for a specific use case completely defeats the purpose of the described systematic approach. Instead, it is possible to constrain the benchmark to only the underlying quantum computation performed by both quantum chemistry and combinatorial optimization applications, the computation of the expectation value of a given Hamiltonian and a quantum state, as described in [94].

This method allows for a fast and low-cost alternative when characterizing the accuracy of a quantum computer.

4.2.3 Time-to-solution

The time it takes a quantum computer to solve a given problem is one of the main selling points for quantum advantage. Especially in the realm of combinatorial optimization in an industrial setting, where achieving a valid solution for a classically untractable problem formulation faster than a competitor can translate into a massive increase in revenue for a company. Hence, it is important to understand how to differentiate quantum devices in terms of how quickly they can arrive at a valid solution. Similarly to the previous characterization aspect, it is necessary to make the distinction between quantum annealers and gate-based computers, as their execution time depends on different properties and parameters.

On the one hand, quantum annealers' time-to-solution primarily depends on the annealing time selected for the execution. As mentioned before, this parameter determines how long the system's adiabatic evolution will take, and hence, it also controls the quality of the results. This is why the annealing time has to be considered a trade-off between speed and quality. It can not be dialed way down as it will affect the quality of the solution, and it shouldn't take a large value since we desire our execution time to be faster than solving the problem on a classical computer. Additionally, the annealing time is independent of the size of the problem, meaning that solving a problem formulation with 50 variables should take the same time as solving one with 200. Nevertheless, since the formulation with more variables creates a more complex energy landscape, it is possible that a larger annealing time is needed to achieve the most optimal solution. This leads us once again to the aforementioned trade-off. Generally, newer generation annealers, such as the D-Wave Advantage machines, require an overall lower annealing time to find an optimal solution than their predecessors. Allowing for quicker computations without losing any quality in the results.

On the other hand, general-purpose gate-based quantum computers derive their time-to-solution directly from lower-level physical characteristics of the QPU, for instance, gate speed. Also, this time is directly correlated with the complexity of the combinatorial optimization problem at hand. Contrary to annealers, gate-based computers take more time to compute a valid solution when more variables and/or constraints are present. It is possible to determine which gate-based quantum computer will execute a given problem formulation faster and, therefore, have a lower time-to-solution by employing a single metric that encapsulates the hardware characteristics responsible for the speed of computation. This metric is called "Circuit Layer Operations Per Second" (CLOPS) and is analogous to the FLOPS metric in classical computers. It was developed by IBM and measures how many parameterized quantum circuits with the same width and number of layers can be executed by a QPU per unit of time [90]. Commonly, a larger number of CLOPS would mean a faster time-to-solution for the problem at hand. More modern quantum computers with

faster gate speeds and better connectivity possess a higher number of CLOPS, making them preferable for time-sensitive industrial applications.

When assessing which quantum computer to select in terms of its time-to-solution within the context of solving a large-scale industrial problem, it is advisable to experiment with both annealers and gate-based devices before committing to a single quantum computing paradigm. A more pragmatic approach is needed for this characterization aspect since the time-to-solution for a problem can vary significantly with regard to its formulation. Nonetheless, some general suggestions are to begin the exploration with a quantum annealer to take advantage of the constant time complexity given by the annealing time. In case the annealer does not satisfy the time or quality requirements for the solution, gate-based devices should be used, prioritizing computers with a larger number of CLOPS.

4.3 Supported Providers

As previously mentioned, QPLEX is a hardware-agnostic programming library that allows the seamless execution of combinatorial optimization problems on a wide variety of quantum platform providers without adding any programming overhead to the user. Each of these providers has different quantum hardware offerings and unique features that must be fully understood to properly use the library and achieve the best results possible for a given problem. Therefore, this section will thoroughly describe all the available quantum computing providers within QPLEX while highlighting their benefits and limitations.

4.3.1 IBM Quantum

IBM Quantum is IBM's quantum computing division. In 2016 they decided to release the IBM Quantum Platform with the goal of offering public access to IBM's quantum computing devices. Along with the platform, IBM developed Qiskit, a Python library for programming and experimenting with their quantum computers. This platform also provides a visual quantum circuit composer (IBM Quantum Composer) as well as an online IDE (IBM Quantum Lab) for faster and simpler development of quantum solutions. Initially, only a few quantum devices were available to the public; these possessed a low number of qubits and had multiple limitations, including high noise levels and low coherence times. However, over 24 quantum computers, including their most advanced and powerful devices, are currently available in the IBM Quantum Platform.

IBM offers three different types of access to these devices[1]: Open plan, Pay-As-You-Go, and Premium. The Open Plan access is available for anyone who registers on the platform. This is the most basic level of access, and it only allows one to interact with the smaller quantum devices (usually up to 7 qubits). Additionally, since most people have this level of access and only a few computers are available, the execution queues are long. For the Pay-As-You-Go model, the user has to pay for the number of seconds their quantum programs

[1] https://www.ibm.com/quantum/access-plans

take to execute in exchange for getting access to the newer and larger devices (up to 127 qubits). Since this is a paid service, there are fewer users, which makes the execution queues considerably shorter, allowing for a quicker workflow. This plan also includes access to exclusive features such as Qiskit runtime. The last level of access is the Premium, and it is only available for selected users, including research labs and specialized companies. Among the premium features is access to exploratory devices that haven't been released to the public (currently up to 433 qubits), queue priority for any device, and hands-on support directly from IBM. Regardless of the access, all users require an API token to interact with IBM's cloud successfully.

The complete IBM quantum hardware line-up is based on superconducting qubits. Nonetheless, they differ in their chip architecture. Currently, only five architectures are available to the public on the IBM Quantum Platform: Falcon, Hummingbird, Eagle, Osprey, and Egret. Each of these has its own set of features and unique properties, although more often than not, the newer devices borrow many characteristics from the previous generations. The end goal of each new architecture is to increase at least one of the previously described characterization aspects, in other words, the number of qubits, the quantum volume, and the number of CLOPS. Each architecture can have different versions and can be implemented into one or more quantum computers. For instance, the Falcon architecture has seven different versions (i.e., r4t, r5.11H, r4P, r8, r5.11, r5.10, r5.11L) and has been implemented into 16 different quantum devices. Table 4.1 presents the main metrics of the flagship quantum computer for each chip architecture.

Table 4.1: Main metrics of some IBM Quantum's backends.

Backend	Chip Architecture	Qubits	Quantum Volume	CLOPS	Open Plan
ibmq_belem	Falcon r4T	5	16	2.5K	Yes
ibmq_manila	Falcon r5.11L	5	32	2.8K	Yes
ibmq_perth	Falcon r5.11H	7	32	2.9K	Yes
ibmq_guadalupe	Falcon r4p	16	32	2.4K	No
ibmq_mumbai	Falcon r5.10	27	128	1.8K	No
ibmq_algiers	Falcon r5.11	27	128	2.2K	No
ibmq_ithaca	Hummingbird r3	65	N/A	N/A	No
ibmq_sherbrooke	Eagle r3	127	32	904	No
ibmq_seattle	Osprey r1	433	N/A	N/A	No

IBM Quantum is at the forefront of quantum hardware development. At present time, they own the largest gate-based quantum computer on the market, the IBM Seattle, with 433 qubits, and are planning on almost tripling this size by the end of 2023 with their brand new Condor architecture. Additionally, IBM has also come up with significant innovations in the software realm, such as Qiskit runtime and Dynamic Circuits, both of which enable an optimized and higher-quality execution of quantum programs on the aforementioned

devices. The constant roll-up of features and the clear software and hardware roadmap[2] that IBM possesses make it a key player in the quantum industry. However, their limited access to the larger and more powerful devices, along with the long execution queues for the open access users, limits in a great manner the experimentation and applicability to large-scale use cases. This makes it necessary to upgrade to a paid plan in most cases when dealing with combinatorial optimization problems.

4.3.2 D-Wave Systems

D-Wave Systems is a Canadian full-stack quantum computing company. It was founded in 1999 and was the first company to develop and sell functional quantum computers. In contrast to most quantum hardware manufacturers, D-Wave does not develop gate-based computers. Since their conception, the construction of quantum annealers was their main objective. Nonetheless, they recently announced that developing general-purpose gate-based devices is within their plans. D-Wave allows public access to their annealers through Leap, the company's cloud platform, and enables the development of quantum solutions with the OceanSDK, a software development kit that provides all the necessary tools for building, executing, and testing programs. The software stack provided by D-Wave is one of the most user-friendly in the market, as it abstracts complex functionality from the user, such as problem encoding and hybrid workflow execution.

To gain access to Leap, a user has to be registered to one of the three available plans[3] in the platform: Free Developer access, Commercial access, and Research & Education access. The first plan is free of cost. It provides limited access to D-Wave's resources, more specifically one minute of execution time per month, and requires that all software developed under the plan is publically available to the open-source community. The remaining two plans are paid, and D-Wave assesses their cost on a case-to-case basis. For both plans, there is no obligation to open source projects or code, and there is direct support from the company. The main difference between these two plans is, on the one hand, that the Research & education plan has limited access to computational resources. This level of access is superior to the one provided by the free plan, but it is constrained by the agreement between the two parties. On the other hand, Commercial access does not have any limitations, and instead, it is charged in a pay-as-yo-go manner. All aforementioned plans have access to the same devices, and an API token is required in order to interact with them through Leap.

As previously mentioned, all D-Wave devices are quantum annealers, which allows them to have a larger amount of qubits compared to gate-based devices. As of the present day, the largest D-Wave QPU possesses over 5000 qubits, over ten times more than IBM's largest quantum computer. However, these qubits are not general purpose and can only be utilized to execute quantum annealing procedures. This condition is given by the QPU's architecture or topology, more specifically, how the qubits are organized in a graph and how they are connected through couplers. Currently, D-Wave supports devices with three different

[2] https://www.ibm.com/quantum/roadmap
[3] https://cloud.dwavesys.com/leap/plans/

architectures: Chimera, Pegasus, and Zephyr[4]. The Chimera topology is the oldest one among the three, and it consists of an interconnected lattice of unit cells (four horizontal qubits coupled to four vertical qubits). Each unit cell is connected to another one through an external coupler, while the qubits within the cell are coupled with internal couplers. For the Pegasus topology, D-Wave borrowed many aspects from Chimera, including the internal and external couplers. However, this new architecture supports odd couplers, which connect similarly aligned pairs of qubits to each other. This new feature enables a degree of connectivity of 15, a significant increase compared to Chimera's degree of six. Lastly, Zephyr, the newest topology, uses a similar configuration to Pegasus but increases the total number of couplers per qubit from 15 to 20. This new architecture adds four more internal couplers and one additional odd coupler. The higher connectivity allows the QPU to support unit cells of dimensions up to eight by eight qubits, increasing the scale of the annealer.

D-Wave's current quantum computer lineup consists of three machines: the D-Wave 2000Q, the Advantage system, and the Advantage2 system. Each of these uses a different chip architecture and has its own set of metrics. Table 4.2 summarizes this information. Moreover, Leap provides access to hybrid solvers. These systems consist of quantum and hybrid resources and allow users to execute large problem formulations that generally wouldn't fit on the raw QPUs. The solvers are categorized by the type of problems they support, for instance, formulations with binary variables, with discrete variables, and with constraints.

Table 4.2: Main metrics of D-Wave's QPU lineup.

Backends	Chip Architecture	Qubits	Connectivity Degree
D-Wave_2000Q	Chimera	+2000	6
Advantage_system	Pegasus	+5000	15
Advantage2_system	Zephyr	+5000	20

Due to the large scale of their devices, the ease of use of their SDK, and the high applicability of quantum annealing to relevant industrial problems, D-Wave has cemented itself as a top company in the quantum computing field. Furthermore, the usefulness and potential of D-Waves's devices have been demonstrated through the development of complex applications with multiple industry partners, such as Mastercard, Volkswagen, Deloitte, Lockheed Martin, and Save-on-Foods. The excellent reputation achieved from these projects has sparked the interest of many other companies in exploring quantum annealing for their own use cases. Given the user-friendly approach of D-Wave towards quantum computing and the free tier access to their top-of-the-line devices, it is very simple for any organization and even individuals to develop custom quantum solutions and assess the benefits and drawbacks of the service before committing to a paid plan.

[4]https://docs.dwavesys.com/docs/

4.3.3 Amazon Braket

Similar to IBM Quantum, Amazon Braket is Amazon Web Services'(AWS) quantum computing-as-a-service platform. However, unlike IBM, Amazon does not build its own quantum computing devices; they collaborate with hardware partners to provide access to their computers through the AWS cloud platform. This makes Amazon Braket a hardware-agnostic provider, as different devices can be used to execute algorithms. Braket was made available to the public back in 2020 with the goal of providing a fully managed AWS service that allows for the design, development, and testing of quantum algorithms. In addition to enabling access to quantum hardware, AWS has built a complex software stack with tools to support all the needs of developers. Among these tools are the Amazon Braket SDK, AWS' own quantum computing software development kit, Braket Hybrid Jobs, a quantum-classical runtime for optimizing the execution of hybrid algorithms, and managed notebooks, a complete quantum computing development environment in the cloud. Additionally, Braket is seamlessly integrated with other AWS services, such as S3 for storing execution results, CloudWatch for monitoring the execution of quantum jobs, and EC2 for the execution of hybrid workflows. This makes it very simple for a cloud practitioner to start working with this quantum service.

As is expected from any AWS service, Braket also operates through a freemium access model. There is a free tier that only allows one hour of execution time on available quantum simulators per month. The use of any other service or quantum device will incur extra costs. If the free tier resources are surpassed, a pay-as-you-go model will be used for charging users. The pricing for each device and service are clearly laid out in the AWS portal[5], and depending on which is being used, a different unit is employed for calculating the total cost of use. In the case of quantum computers, the number of tasks and the number of shots for a given algorithmic execution will be employed for the pricing. For simulators, Hybrid Jobs, and Managed Notebooks, the number of minutes the execution of a task takes is the chosen cost unit. To gain access to the service, it is necessary to have an AWS account and to generate a development key pair. Then, these credentials have to be configured into the AWS CLI so the Amazon Braket SDK can successfully use the resources in the cloud.

Four quantum computing hardware providers are currently partnered with AWS to provide access to their machines: Rigetti, Oxford Quantum Circuits (OQC), IonQ, and QuEra Computing. Each of these companies brings to the table different qubit implementations, architectures, and metrics. In the case of Rigetti, one gate-based superconducting quantum computer is provided, the Aspen M-3. This device has 79 qubits and implements Rigetti's latest chip architecture for enhanced readout and better circuit fidelity. OQC also offers access to a gate-based superconducting device through Lucy, their latest quantum system. Lucy possesses eight qubits, and it's built using a brand-new type of qubit, the Coaxmon, which allows the QPU to have increased simplicity, flexibility, ease of engineering, and scalability[6]. Regarding IonQ, this company supplies ion-trap quantum processors capable of implementing gate-based solutions. At present, there are three IonQ devices available in

[5] https://aws.amazon.com/braket/pricing/
[6] https://aws.amazon.com/braket/quantum-computers/oqc/

Amazon Braket: the Aria1, Aria2, and Harmony. The first two share the same Aria architecture and have a total of 25 qubits. These are IonQ's latest devices and feature improved gate operations as well as reduced 1-qubit and 2-qubit gate errors. The latter device supports 11 qubits and provides an efficient execution for shallower circuits and smaller-scale proof of concept work[7]. QuEra's Computing has a particular offering as it provides access to a neutral atom-based quantum computer, Aquila, which functions under the Analog Hamiltonian Simulation paradigm. This is the company's first generation QPU, and it possesses 256 qubits in analog mode, enabling long lifetimes and increased scaling. Table 4.3 contains a summary of the aforementioned quantum backends.

Table 4.3: Main metrics of Amazon Braket Quantum backends.

Backend	Provider	Chip Architecture	Qubits
Aspen-M-3	Rigetti	Aspen	79
Lucy	OQC	Coaxmon	8
Aria-1	IonQ	Aria	25
Aria-2	IonQ	Aria	25
Harmony	IonQ	Harmony	11

Amazon Braket is an excellent quantum computing provider for users who do not want to commit to a single hardware vendor or a single paradigm and are already familiar with AWS' cloud offering. Additionally, the aforementioned software features give added value as they make the overall development experience so much easier for novel users and enable them to optimize the more complex workflows commonly employed by quantum experts. The pricing model employed for Braket allows the exploration of the platform and the service without paying upfront. However, if one wants to interact with real quantum devices and benefit from more advanced software features, it would be necessary to pay for the resources used.

4.4 Chapter Summary

In Chapter Four, the three most prevalent quantum hardware implementations (SCQ, TIQC, PQC) were described in a clear and approachable manner for anyone who is not familiar with this technology. Three characterization aspects for quantum providers were presented with the goal of helping users reduce guessing and allowing them to shortlist the most appropriate options for quantum hardware. Finally, a thorough examination of the offerings from the quantum platform providers supported by QPLEX was provided.

[7] https://ionq.com/quantum-systems/harmony

Chapter 5

QPLEX: System Design and Implementation

Taking as a foundation the knowledge about quantum algorithms and quantum platforms compiled from chapters 3 and 4, the current chapter aims to discuss the conception of QPLEX in detail. The following sections present the specified requirements for the QPLEX programming library, its system's architecture, and its main components. We also provide a description of how to use the main system features of QPLEX.

5.1 Requirements

Following traditional software engineering practices, we collected a set of functional requirements for the QPLEX system. These helped guide the design and implementation phases of the software solution. The requirements formulation was directly influenced by the motivation and the problem statement previously discussed. This section outlines the chosen requirements for the implementation of QPLEX.

Requirement 1: The system allows the modeling of complex combinatorial optimization formulations through a declarative API.

Requirement 2: The solution has support for the platform-agnostic execution of combinatorial optimization problem formulations on different quantum computing devices.

Requirement 3: The system has the capability to be a customizable and extensible software solution for supporting multiple quantum algorithms and quantum platforms.

5.2 System Overview

Given the nature of the problem at hand and the requirements above, the conceived solution is a complex and intricate software system. To allow the reader to fully understand the inner workings of QPLEX, the following section presents the intended functional workflow as well as describe each of the main modules that make up the larger system.

5.2.1 Workflow

Following requirement one, we decided to design QPLEX as an extension for DOcplex, a widely used classical combinatorial optimization library written in Python. This would enable the simple modeling of a combinatorial optimization problem formulation while providing a familiar interface for optimization practitioners and researchers. DOcplex is a complete software library, and it allows for the construction of many different types of formulations, including unconstrained and constrained (i.e., equality and inequality constraints) problems, with the possibility of using binary, discrete, and continuous variables. The "Model" class of this library allows the user to specify all the details about the problem, such as name, variables, constraints, and the objective function. Additionally, as a built-in feature within DOcplex, there is a high-performance classical solver named CPLEX, which allows a simple execution of the formulated optimization problems on your own computer or in the cloud.

To maintain the benefits of both the "Model" class and the classical solver, we decided to create a wrapper class that would extend the already existing behavior of DOcplex while enabling the desired quantum platform-agnostic execution. This new workflow is depicted in Figure 5.1, where a new class named "QPLEX Model" (QModel) represents the wrapper or superclass for the DOcplex model, and it provides all the functionalities for the "quantum" branch of the execution workflow to work properly. The new QModel allows the user to choose which type of execution will be used to solve the problem formulation, either classical or quantum. If classical is chosen, the CPLEX solver is used; otherwise, supported quantum solvers will be employed with the help of a solver factory. Furthermore, within the quantum execution, there is the option to specify which quantum provider will be used. At the end of the execution, the user should expect the exact same structure for the solved model as they would expect from the base DOcplex library, no matter which solver was employed.

5.2.2 Modules

The proposed software solution is organized into three main modules (i.e., *Solvers*, *Algorithms*, and *Commons*) to allow for a cohesive code base while ensuring low coupling and few dependencies within the system. Figure 5.2 presents the general overview of the modules and the dependencies among them. The respective details of the three modules are discussed in the following subsections.

Solvers

The Solvers module is in charge of managing solver classes. Each class corresponds to a quantum provider (e.g., IBM Quantum, D-Wave, or Amazon Braket), and they abstract vital functionalities for the platform-agnostic execution of optimization models. For instance, the parsing of a problem formulation into a format amenable to the provider's SDK, the necessary instructions for executing the problem, the function for reading and interpreting the results sent back from the QPU, and the logic for selecting a physical backend for the

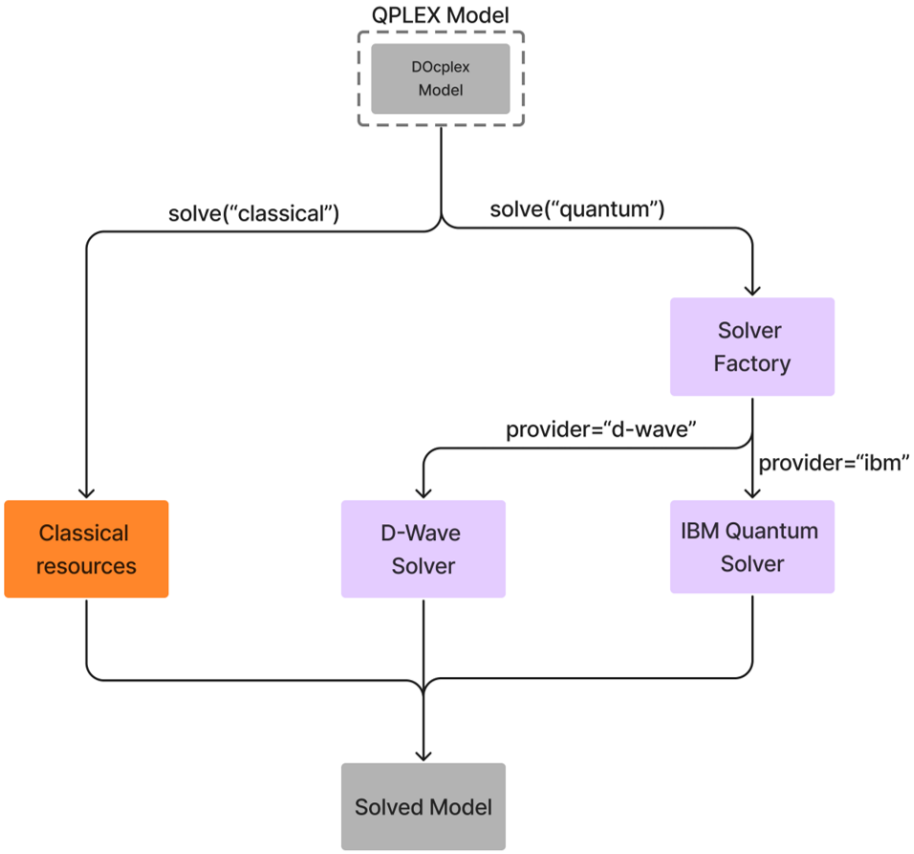

Figure 5.1|: Execution workflow for a combinatorial optimization problem using QPLEX.

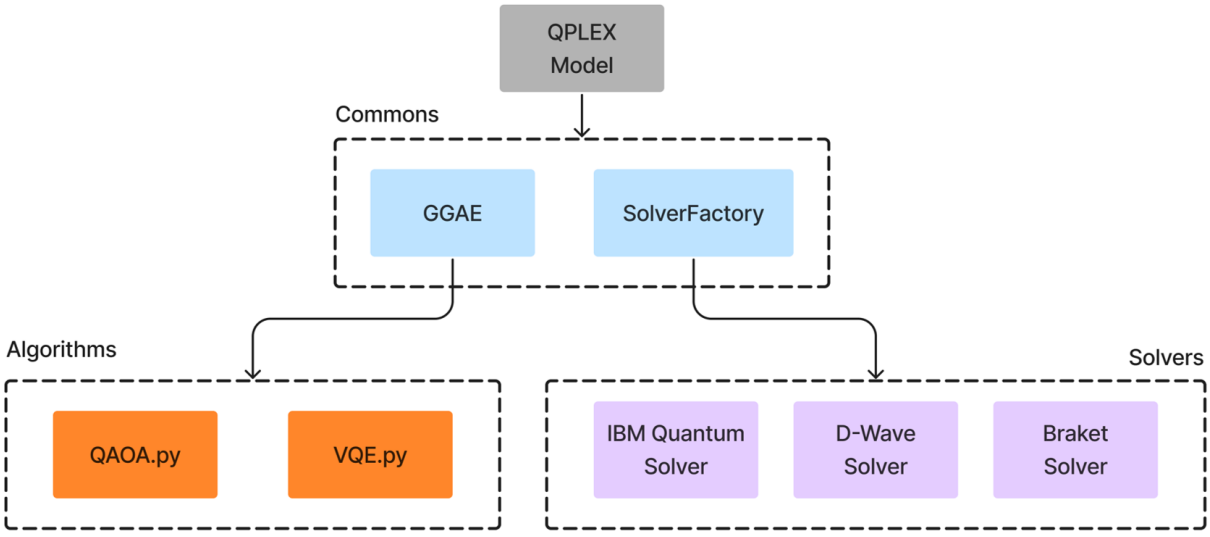

Figure 5.2|: System design overview of QPLEX.

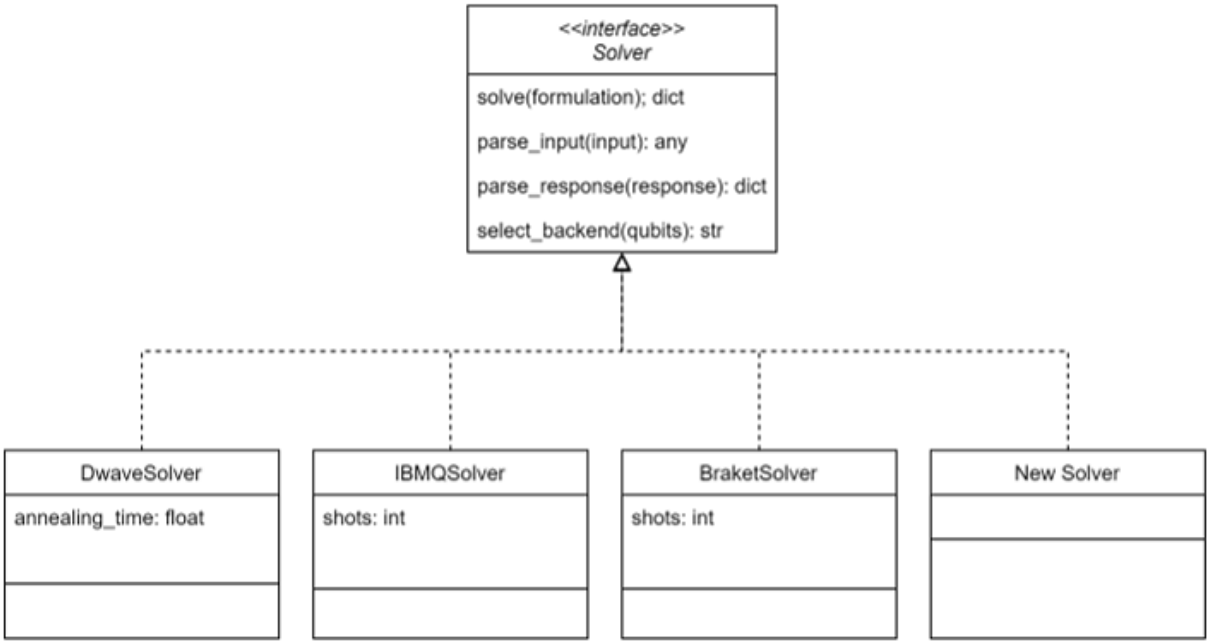

Figure 5.3|: Class diagram of the Solvers module.

execution. All the instructions within a Solver class are SDK-specific, meaning they only work for their corresponding quantum platform provider. This would be the case with Qiskit and IBM Quantum. To enforce that all Solver classes have the same functionality, a "Solver" software interface is present in the system, as shown in Figure 5.3. The interface specifies the common behavior among all solvers, but it allows each Solver class to have its own attributes as needed, such as annealing time in the case of the D-Wave solver and the number of shots for the others.

Algorithms

This module has the responsibility to provide the algorithmic specification for all the gate-based quantum algorithms supported by QPLEX (i.e., VQE and QAOA). Each class in the Algorithms module provides a template, represented by a parameterized OpenQASM3 string, for implementing the corresponding algorithm. In a similar manner as the Solvers module, a software interface is also present in the Algorithms module with the goal of enforcing all quantum algorithms to have the necessary methods for execution. These include the logic for updating the parameters and the definition of the cost function for the classical optimization subroutine. In addition, each Algorithm class can have custom attributes depending on the underlying algorithmic implementation. For instance, VQE has a $layers$ attribute while QAOA has a p attribute. This feature allows for a high level of experimentation as the users can explore different configurations for each algorithm without incurring extra development time. The corresponding class diagram for the described module is depicted in Figure 5.4.

Commons

The commons module is not as homogeneous as the previous ones. Here, different classes

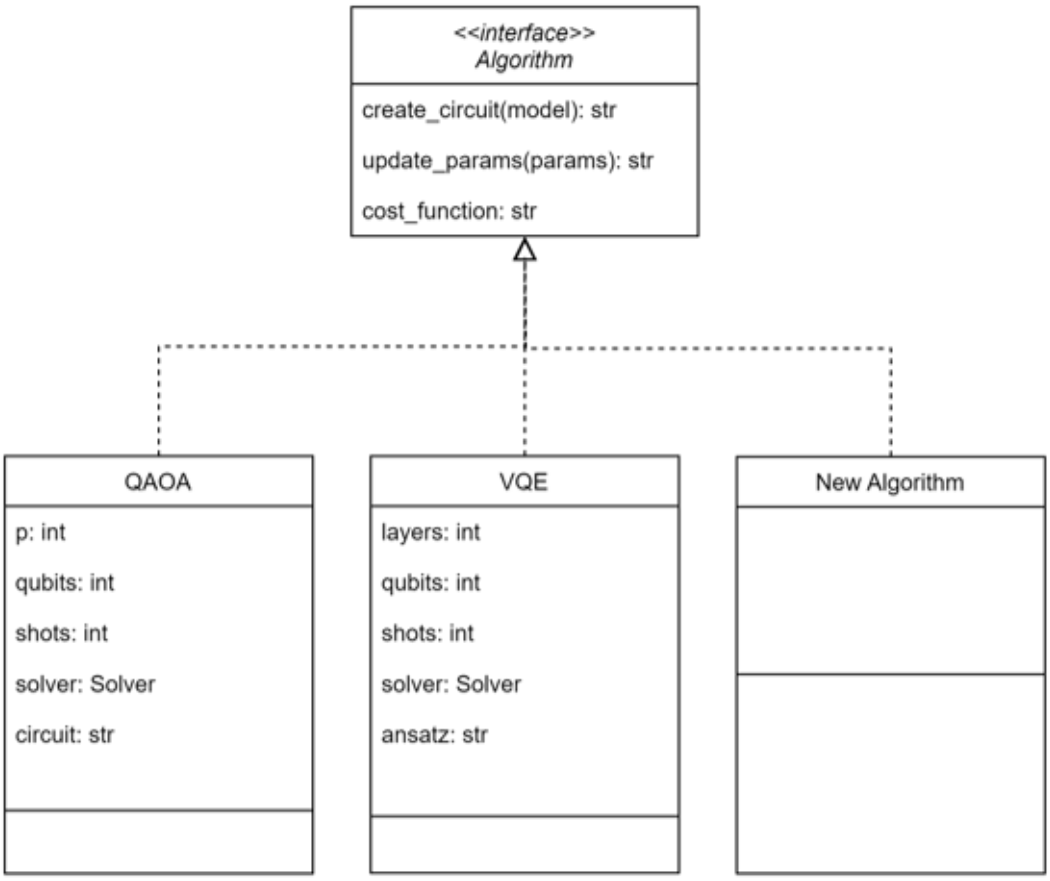

Figure 5.4|: Class diagram of the Algorithms module.

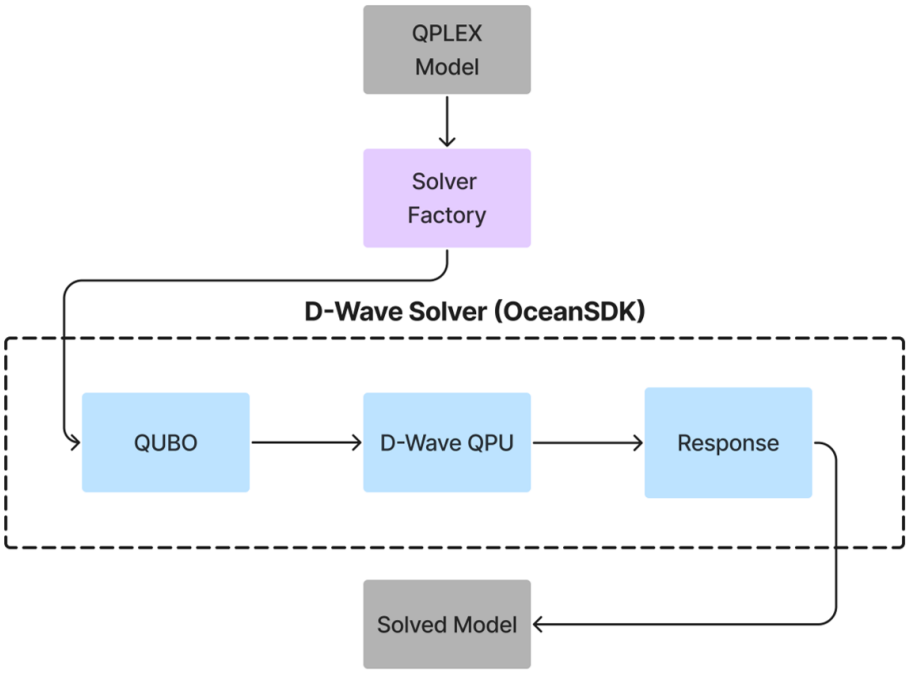

Figure 5.5|: Model execution using the D-Wave solver.

that support the platform-agnostic execution of quantum algorithms are present. Currently, there are only two classes in this module: the Solver Factory and the Generalized Gate-based Algorithms Execution Manager (GGAEM). However, each of these plays a fundamental role in the system. On the one hand, the Solver Factory handles the instantiation of Solver classes for when the user is required to solve a problem formulation with quantum resources, as shown in Figure 5.5. On the other hand, the GGAEM enables the workflow for seamlessly executing gate-based hybrid quantum algorithms on gate-based devices and classical computing resources. It is worth mentioning that QModel directly depends on functionality from the Commons module, specifically the Solver Factory. Also, this module has two dependencies, one created by the GGAEM class and another one by the Solver Factory. More specifically, the former requires access to the gate-based quantum algorithmic specifications within the Algorithms module to enable the platform-agnostic nature of the system, and the latter has to interact with the Solvers module to return the necessary instances for the execution. We will dive deeper into implementing both the Solver Factory and the GGAEM classes in the following sections.

In addition to providing a simple workflow for execution, the design of this system greatly minimizes the coupling between the solvers, the algorithms, and the rest of the system, as all the logic for each of these is contained within its own module and used through common software interfaces. The Solvers module, in particular, provides a high level of encapsulation. This is useful if a quantum provider decides to update how to access their machines, as it would only be necessary to modify the corresponding solver class; the remaining sections of code within the system are not affected by the change. This also applies when a new provider has to be integrated into the library, as it is only needed to create the new solver class with the necessary execution logic and instantiate it within the

solver factory, heavily reducing the amount of time and complexity required to provide access to a new quantum platform. The same is true for the Algorithms module. Adding support for a new gate-based quantum algorithm is just as simple as creating a new class that implements the common interface. Moreover, with the presented architecture, when a new Solver is added, all the algorithms will automatically have access to it, and when a new algorithm is supported, all the solvers can use it for executing a problem formulation.

5.3 General Combinatorial Optimization Model Adaptation

As previously stated, the QModel class from QPLEX allows users to build general combinatorial optimization models that any of the supported solvers can execute. However, the user-specified formulation is not amenable to the quantum providers, and hence, it is necessary to perform a number of transformations beforehand. First, the problem formulation from the QModel has to be converted into a QUBO by encoding the non-binary variables and converting constraints into penalty terms and slack variables. This process is handled in our system automatically by Qiskit converters. Once the QUBO formulation is created, it must be mapped to the target device. For gate-based solvers, this is handled by the GGAEM. Here, the QUBO formulation is mapped to the corresponding parameterized circuit and then executed through the solver's interface. In the case of other solvers, such as D-Wave, the QUBO formulation is directly handled by the provider's SDK, which then is able to map the problem into the QPU and return the results, just as shown in Figure 5.5. For any solver, the solution will always be given as the solved QModel.

5.4 Solver Instances Management

The different quantum solvers play a fundamental role within QPLEX, as they allow the connection to the quantum providers and the eventual execution of the desired algorithm. Therefore, it is paramount to manage them in the most efficient and structured way possible. This is accomplished in our system by the aforementioned Solver Factory class, present in the commons module. It was implemented by following the software engineering Factory design pattern, and it functions by instantiating the correct solver when a given quantum provider is selected in the execution workflow. The UML diagram of the implementation is depicted in Figure 5.6. This component is employed every time a QModel is created, as it allows the system to understand which set of instructions has to be executed to solve the specified combinatorial optimization problem. The Solver Factory can be directly used within the QModel or through the GGAEM, depending on whether the selected provider employs gate-based devices or not. If we are dealing with the latter case, then the instantiation of the required solver only happens inside the GGAEM workflow. The specifics of this process are described in the following section.

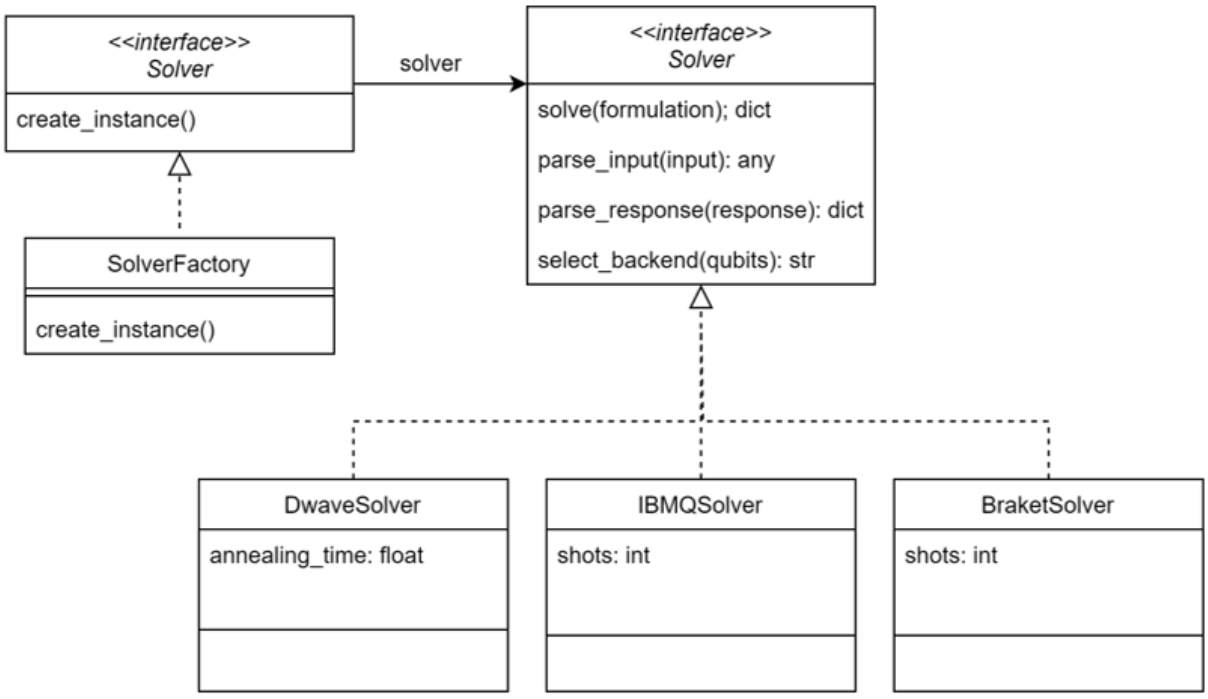

Figure 5.6|: UML diagram of the Solver Factory implementation.

5.5 Generalized Gate-Based Algorithms Execution Workflow

Contrary to annealers, which can use QUBO formulations directly to solve the underlying optimization problem, gate-based devices are required to implement a quantum algorithm in the form of a quantum circuit every time an optimization problem is addressed. Most of the time, these implementations are provider-specific, which makes it necessary to have multiple versions of the same approach when experimenting on different quantum computers, making the process of adding support for more devices and quantum algorithms cumbersome. Therefore, to enable the capability to run these gate-based quantum algorithms on different platforms seamlessly, it was necessary to devise a novel algorithmic execution workflow when building QPLEX, referred to as the Generalized Gate-Based Algorithms Execution Workflow (GGAEW). All the logic for the workflow is contained within the GGAEM class, and it's divided into three main sections, which are described below. The general overview of this logic is depicted in Figure 5.7.

5.5.1 Quantum Intermediate Language Compilation

QPLEX, and more specifically, the GGAEW, heavily relies on a quantum intermediate representation language for building a common algorithmic representation that multiple quantum providers can transpile into their platform-specific instructions. This representation corresponds to a quantum circuit written using OpenQASM3 directives. The building process of this circuit consists of the compilation of a set of quantum circuit specifications provided by an algorithm instance, the problem-specific data given by the QUBO formulation generated from the QModel, and a number of randomly chosen parameters that will be optimized in the further sections of the workflow. More specifically, the specifications are the different sections of the corresponding parameterized quantum circuit for the chosen algorithm. For instance, if QAOA is selected, the QAOA Algorithm instance returns the circuits for the problem Hamiltonian and the Mixer layers, built using parameterized rotation gates. Additionally, the data from the QUBO formulation corresponds to the linear and quadratic terms needed for computing the angles of the aforementioned rotation gates. The parameters are also used in the calculation of the angles. Once the rotation angles are mapped into the circuits, and the final OpenQASM3 string has been compiled, the workflow continues to the next section.

5.5.2 Platform-Agnostic Quantum Circuit Execution

To successfully execute the already compiled quantum circuit containing the problem formulation on a quantum computer, this workflow uses the Solver Factory to retrieve the corresponding solver for the quantum provider requested by the user. Once the correct solver instance is made available to the GGAEM, it proceeds to use the provided provider-specific logic to parse the OpenQASM3 circuit into an object supported by the provider's SDK for its eventual execution. In the case of IBM Quantum, Qiskit's $qiskit.qasm3.loads(string)$

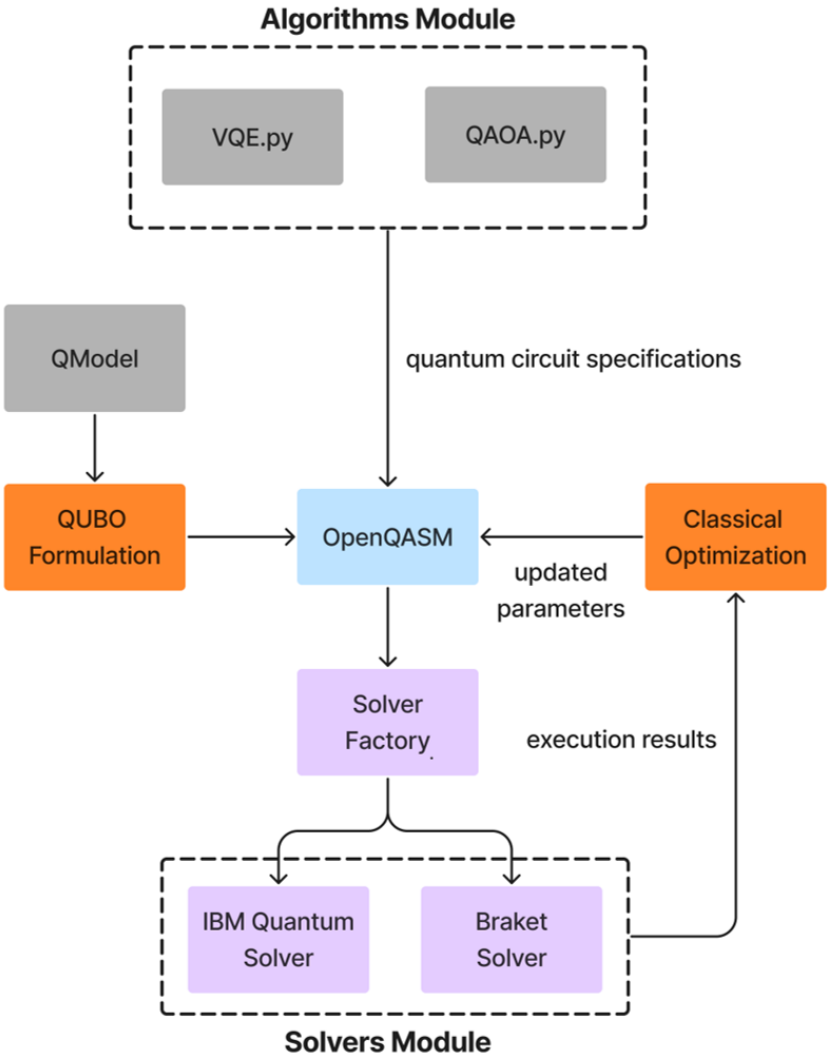

Figure 5.7|: Overview of the generalized gate-based algorithm execution workflow.

function is used to generate a $QuantumCircuit$ instance containing all the details from the original problem formulation, which can then be executed as a regular Qiskit circuit. The same process is performed for the Amazon Braket provider but using the Braket SDK. The SDK-specific instructions for executing the parsed circuit are also provided within the solver instance, making it only necessary for the GGAEM to call the corresponding method to begin the execution. This method also includes the logic for finding the platform backend with the least queue, in case the user did not specify one, and the steps for transforming the results into the appropriate format to continue the next step of the execution workflow. An example of how this transformation would work is converting the counts received by an IBM machine, which are organized in the little-endian format, to the most common and widely used, big-endian format.

5.5.3 Classical Optimization Routine

Given that the GGAEM was designed for the execution of hybrid quantum-classical algorithms, such as QAOA and VQE, it was necessary to include an optimization routine within the workflow for finding the best set of parameters for the variational circuits that represent the quantum section of the algorithm. This step takes place once the counts from the circuit execution are retrieved and parsed into the designated format. For practicality and future-proofing purposes, the Python library SciPy was integrated into QPLEX to gain access to a large set of powerful classical optimizers that can successfully tune the circuit parameters. In order to make use of the optimizers, it was necessary to define a cost function to minimize, which in the context of hybrid quantum-classical optimization algorithms is the expectation value of the problem Hamiltonian evaluated on the quantum circuit. This quantity can be easily calculated by computing the retrieved counts from the previous step. Each optimization step finds a new set of parameters, which forces the workflow to return to the second stage and execute the same circuit with updated parameters to obtain the new expectation value, which hopefully would be smaller. As mentioned in Chapter 3, these types of algorithms are iterative and require multiple optimization steps for the cost function to converge towards a global minimum. The maximum number of optimization steps and the learning rate can be specific by the user before the algorithm execution begins. When the expectation value has converged, the final results are mapped onto the QModel, and it is returned to the user for verification, along with a report of the execution metrics.

5.6 QPLEX Usage

QPLEX provides a user-friendly programmatic interface that allows practitioners and researchers to model a wide range of combinatorial optimization problem formulations and execute them seamlessly on quantum devices. In the following subsections, we show how to go about modeling, executing, and retrieving results and useful information about the system.

```python
from qplex import QModel
import numpy as np
import networkx as nx
# Problem definition
graph = nx.Graph()
graph.add_nodes_from(np.arange(0, 6, 1))
edges = [(0, 1, 2.0), (0, 2, 3.0), (0, 3, 2.0),
         (0, 4, 4.0), (0, 5, 1.0), (1, 2, 4.0),
         (1, 3, 1.0), (1, 4, 1.0),(1, 5, 3.0),
         (2, 4, 2.0), (2, 5, 3.0), (3, 4, 5.0),
         (3, 5, 1.0)]
graph.add_weighted_edges_from(edges)
weight_matrix = nx.adjacency_matrix(graph)
shape = weight_matrix.shape
size = shape[0]
qubo_matrix = np.zeros((size, size))
qubo_vector = np.zeros(size)
for i in range(size):
    for j in range(size):
        qubo_matrix[i, j] -= weight_matrix[i, j]
for i in range(size):
    for j in range(size):
        qubo_vector[i] += weight_matrix[i, j]
# Instantiate the model
model = QModel('max_cut')
# Add variables
x = model.binary_var_list(6, name="x")
# Get linear terms
linear_terms = sum(qubo_vector[i] * x[i] for i in range(size))
# Get quadratic terms
quadratic_terms = sum(qubo_matrix[i][j] * x[i] * x[j]
                      for i in range(size)
                      for j in range(size))
# Build the objective function
obj_fn = linear_terms + quadratic_terms
model.set_objective('max', obj_fn)
```

Figure 5.8|: QModel implementation for a Max-cut problem instance.

```python
from qplex import QModel
# Problem definition
weights = [4, 2, 5, 4, 5, 1, 3, 5]
values = [10, 5, 18, 12, 15, 1, 2, 8]
max_weight = 15
n = len(values)
# Instantiate the model
knapsack_model = QModel('knapsack')
# Add variables
x = knapsack_model.binary_var_list(n, name="x")
# Add constraints
knapsack_model.add_constraint(sum(weights[i] * x[i] for
                                  i in range(n)) <= max_weight)
# Build the objective function
obj_fn = sum(values[i] * x[i] for i in range(n))
knapsack_model.set_objective('max', obj_fn)
```

Figure 5.9|: QModel implementation for a Knapsack problem instance.

5.6.1 Problem Modelling

The QModel class provides all the necessary logic for the user to model the desired formulation. To begin, this class has to be instantiated by providing a name for the current problem instance. Then, the variables are added to the QModel object along with the objective function and constraints if the problem requires them. The objective function can have both linear and quadratic terms. Figure 5.8 presents how to create a QModel to solve a Max Cut problem formulation with linear and quadratic terms and without constraints. Figure 5.9 shows how to create a QModel to address a Knapsack problem [95] in QPLEX, with constraints and only linear terms.

5.6.2 Problem execution

The execution of the previously modeled problem formulation is achieved through the QModel's $solve$ method. This function receives one required argument and, optionally, a dictionary of keyword arguments ($kwargs$) for defining different execution properties. The required argument determines which type of resources will be used to solve the problem; it can be "classical" or "quantum". If the execution argument is not given by the user, then the execution will default to use the CPLEX classical solver. Additionally, all the $kwargs$ have default values and can be modified by the user during the call to the solve method. A complete list of all possible arguments for the solve method and their default values are

```
18 execution_arguments = {
19     "provider": "ibmq",
20     "backend": "ibmq_perth",
21     "algorithm": "qaoa",
22     "p": 3,
23     "max_iter": 500,
24     "shots": 2048
25 }
26
27 knapsack_model.solve('quantum', **execution_arguments)
```

Figure 5.10|: Execution of the solve method using keyword arguments.

presented in Table 5.1. Additionally, Figure 5.10 depicts how the solve method can used to run a given problem.

Table 5.1: List of keyword arguments available for the $solve$ method.

Argument	Purpose	Default Value
provider	The quantum platform provider	"d-wave"[1]
backend	The specific quantum device	Calculated
algorithm	The quantum algorithm to use	"qaoa"
ansatz	The ansatz circuit for VQE	Layered Ansatz
p	The p value for the QAOA algorithm	2
layers	The number of layers for the VQE algorithm	2
optimizer	The classical optimizer	"cobyla"
tolerance	The tolerance value for the optimizer	$1e-10$
max_iter	The maximum number of optimizer iterations	1000
penalty	The penalty constant used for the QUBO	Calculated
shots	The total number of shots	1024
seed	The execution random seed	1

A few of the keyword arguments have dynamic default values, meaning that these are calculated during the execution based on the information provided by the user, the problem formulation, and the current state of the quantum devices in the cloud. In the case of the backend, QPLEX selects the one with the shortest execution queue and enough qubits to map the problem formulation. Additionally, the penalty constant value is calculated by the Qiskit converters module based on the modeled problem.

[1]When the D-Wave API token is available.

It is worth mentioning that for users to access the resources of a quantum provider, it is necessary for them to provide the corresponding API keys or tokens to the library. This can be achieved by setting an environment variable for each token. For instance, $DWAVE\_API\_TOKEN$ in the case of D-Wave and $IBMQ\_API\_TOKEN$ for IBM Quantum. If a user decides to use a backend from a provider for which its token wasn't provided beforehand, QPLEX will not allow the execution. This also affects the calculation of the default provider, as the system will only consider the ones for which the user has access. Hence, in case the $DWAVE\_API\_TOKEN$ is not provided, the default value for the provider now will be "ibmq".

5.6.3 Results and information

Retrieving the results of the problem execution in QPLEX is achieved in the same way as in DOcplex. This means all code that currently retrieves solution information from DOcplex models will work perfectly for QPLEX QModels. For instance, the information about the overall solution can still be retrieved by calling $qmodel.print\_solution()$, and attributes such as $qmodel.objective\_value$ are also available to retrieve specific information about the solution. The main difference is that QPLEX adds execution-specific information to provide a better understanding of how the solution process was carried out. This information includes the type of resources used, the quantum provider, the backend, and execution time. An example output is shown in Figure 5.11.

QPLEX also provides a number of methods that can be used for retrieving information about the library. This is a list of all the methods with their corresponding description.

qplex.info.providers: Prints a list of all the supported quantum providers.

qplex.info.backends(provider): Prints a list of all the accessible backends for a given quantum provider.

qplex.info.algorithms: Provides a list of all the quantum algorithms currently supported.

5.7 Chapter Summary

This chapter described in detail the design and implementation details of QPLEX. Firstly, the functional requirements of the system were stated to be used as guidelines for the development of the library. Afterward, a system overview was presented to introduce the workflow and main components of QPLEX. Additionally, every relevant execution aspect of the library was described in its own section. Finally, we provided a guide on how to use and interact with QPLEX.

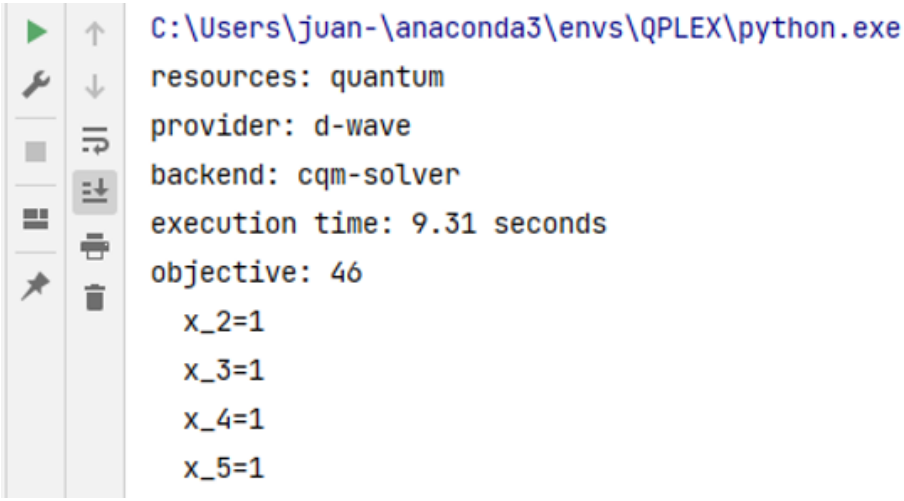

Figure 5.11|: Execution results for a solved QModel.

Chapter 6

Validation and Discussion

The main purpose of this chapter is to present a formal methodology for validating the research efforts undertaken in this work. The research questions formulated at the beginning of the thesis in Chapter 1 serve as guidelines for structuring the validation process. However, given the nature of both RQ1 and RQ2, it is not necessary to include them in the validation framework presented here. Chapters 3 and 4 themselves provide all the necessary information in a concise and organized manner to address these research questions. This leaves us with the more complex RQ3, which can be validated similarly to how one would assess a traditional software system, given that our research methodology stipulated that this research question would be addressed through the development of QPLEX. Software engineering principles dictate that the validation of a software solution involves reviewing the fulfillment of the functional requirements presented during the project planning phase. The set of functional requirements for QPLEX is outlined in Chapter 5 and their validation is articulated in the following sections.

6.1 Functional Requirements validation

In software engineering, functional requirements play a crucial role in specifying the expected set of features and functionalities when delivering a software product to a client or stakeholder. In the context of this thesis, these requirements determine the capabilities of QPLEX. It is possible to assign one capability to each of the previously described functional requirements and establish a method for validating their fulfillment as presented below.

Requirement 1:

Capability: Combinatorial optimization formulation modeling through a classical interface.

Validation method: Select an optimization problem use case and utilize QPLEX to construct an optimization model. Solve it with classical resources and verify its correctness.

Requirement 2:

Capability: Platform-agnostic execution of optimization models.

Validation method: Employ the previously created optimization model and execute it using all the quantum providers available in QPLEX. Verify the correctness of the solutions by comparing them with the results of the classical execution.

Requirement 3:

Capability: Customizability and extensibility for quantum algorithms and platform providers.

Validation method: Repeat the execution of the optimization model using quantum resources three more times, employing different quantum algorithms and combinations of execution keyword arguments.

The subsequent section outlines the specifics of each of the validation methods, including the description of the selected use case as well as the combination of algorithms and execution keyword arguments.

6.2 Validation Set-Up

This thesis describes the software tools to be used to realize the aforementioned validation methods and the combinatorial optimization problem to be addressed. First, an Anaconda development environment[1] with the Python programming language (Version 3.10) installed will be used for performing all the necessary executions on QPLEX (Version 1.0.0). Regarding the selected use case, a large set of problems was reviewed, but the current one was picked given its applicability to the industry and the relatively simple approach needed for its modeling. This problem was proposed as one of the challenges for the Fall 2021 IBM Quantum Challenge, and was titled "Battery Revenue Optimization"[2].

The optimization challenge revolves around maximizing the revenue generated by battery-based energy storage systems used in the integration of renewable energy sources into power grids. This involves selecting the optimal markets for the battery's energy supply within specified time windows to maximize returns. However, a critical consideration is the degradation of the battery's capacity over time due to charge/discharge cycles, which impacts its overall lifetime performance. To optimize revenue, the challenge is to balance market returns (value) based on price forecasts with the expected battery degradation (cost) when selecting markets for energy supply in each time window. The problem setting specified by IBM was extracted from [96] and is as follows:

[1] https://www.anaconda.com/

[2] https://github.com/qiskit-community/ibm-quantum-challenge-fall-2021/blob/main/content/challenge-4/challenge-4.ipynb

Considering two markets M_1, M_2, during every time window (typically a day), the battery operates on one or the other market for a maximum of n time windows. Every day is considered independent, and the intraday optimization is a standalone problem: every morning, the battery starts with the same level of power, so we don't consider charging problems. Forecasts on both markets are available for the n time windows. We assume known for each time window t (day) and for each market:

- the daily returns λ_1^t, λ_2^t.
- the daily degradation, or health cost (number of cycles), for the battery c_1^t, c_2^t.

We want to find the optimal schedule, that is, optimize the lifetime return with a cost less than C_{max} cycles. We introduce $d = max_t\{c_1^t, c_2^t\}$.

We introduce the decision variable z_t, $\forall_t \in [1, n]$ such that $z_t = 0$ if the supplier chooses M_1, $z_t = 1$ if the supplier chooses M_2, with every possible vector $z = [z_1, ..., z_n]$ being a possible schedule. The previously formulated problem can then be expressed as:

$$\max_{z\in\{0,1\}^n} \sum_{t=1}^{n} (1 - z^t)\lambda_1^t + z_t\lambda_2^t \tag{6.1}$$

$$s.t. \sum_{t=1}^{n} [(1 - z_t)c_1^t + z_t c_2^t] \leq C_{max} \tag{6.2}$$

Upon reviewing the problem description, it becomes apparent that at its core lies a knapsack problem, which can be readily modeled using QPLEX. To construct the knapsack formulation, we need to identify the decision variables, their corresponding values and costs, and the cost constraint. Equation 6.2 indicates that the vector z_t will serve as our decision variable vector. However, determining the values and costs for all variables isn't straightforward because the problem provides two sets of values or returns (λ_1^t and λ_2^t) and two sets of costs (c_1^t and c_2^t). Therefore, it's necessary to compute new cost and value vectors that accurately represent the data in the problem formulation. In [X], the authors propose transforming the data by introducing $p_t = (\lambda_2^t - \lambda_1^t)$ and $w_t = (c_2^t - c_1^t)$ and rewriting the problem as:

$$\max_{z_t\in\{0,1\}^n} \sum_{t=1}^{n} p_t z_t \tag{6.3}$$

$$s.t. \sum_{t=1}^{n} w_t z_t \leq C'_{max} = C_{max} - \sum_{t=1}^{n} c_1^t \tag{6.4}$$

Using this new formulation, it's possible to construct the corresponding QModel for the Battery Revenue Optimization problem in the form of a linear knapsack. One final consideration for this setup is the number of qubits required to map the formulation onto a quantum computer. The problem statement from the IBM Quantum Challenge suggests

seven time steps ($t = 7$). Therefore, seven binary variables are necessary, each of which can be encoded into a single qubit. If the formulation had no constraints, additional qubits wouldn't be necessary. However, the presence of the cost constraint necessitates the inclusion of ancillary qubits and penalty terms. As previously mentioned, QPLEX manages this transformation through Qiskit converters, employing one-hot encoding. Given that the cost constraint for this specific formulation is 16, we can deduce that to represent this number, an additional five qubits are required, bringing the total qubit count to 12 for this specific problem instance.

The qubit count for this validation setup poses a limitation on execution using real quantum devices. As demonstrated in Chapter 4, IBM Quantum provides free access to quantum computers with up to 7 qubits; any devices with a higher qubit count require a premium plan. Similarly, with Amazon Braket, executing on any quantum device falls outside the free-tier plan. Moreover, due to the high demand for these large-scale quantum devices, executing the aforementioned problem would take a large amount of time. Given that one of the primary objectives of the validation process is to showcase QPLEX's ability to execute optimization models across different quantum providers, it was decided to utilize the quantum simulators hosted by each provider in their respective cloud environments. This approach still allows us to validate our solution while not incurring any costs and receiving results within a reasonable timeframe due to less demand.

To showcase the customizability of QPLEX, a set of experimental executions will be performed, Table 6.1 summarizes the characteristics of each of them, including quantum providers, algorithms, and more specialized parameters. The source code for the experiment setup is presented in figures 6.1-6.3. These include the combinatorial optimization problem set-up, the QModel building process, and the list of experiments.

Table 6.1: Experiment set-up for validation

Execution #	Provider	Algorithm	Parameters
E1	D-Wave	Quantum Annealing	N/A
E2	IBM Quantum	QAOA	p=6, shots=5120, max_iter=5000
E3	IBM Quantum	QAOA	p=8, shots=10240, max_iter=10000
E4	IBM Quantum	VQE	l=9, shots=10240, max_iter=10000
E5	Amazon Braket	QAOA	p=6, shots=5120, max_iter=5000
E6	Amazon Braket	QAOA	p=8, shots=10240, max_iter=10000
E7	Amazon Braket	VQE	l=9, shots=10240, max_iter=10000

```python
def main():
    # Problem set-up
    # --------------
    t = 7  # Time steps
    l1 = [5, 3, 3, 6, 9, 7, 1]  # Daily return for battery 1
    l2 = [8, 4, 5, 12, 10, 11, 2]  # Daily return for battery 2
    c1 = [1, 1, 2, 1, 1, 1, 2]  # Daily degradation for battery 1
    c2 = [3, 2, 3, 2, 4, 3, 3]  # Daily degradation for battery 1
    c_max = 16  # Maximum degradation

    # Knapsack definition
    # -------------------
    values = list(map(lambda x, y: x - y, l2, l1))
    weights = list(map(lambda x, y: x - y, c2, c1))
    max_weight = c_max - sum(c1)
    knapsack_model = build_knapsack_model(values, weights, max_weight)
```

Figure 6.1|: Code snippet for the problem set-up.

```python
def build_knapsack_model(values: List, weights: List, max_weight: int) -> QModel:
    # Total number of items
    n_items = len(values)
    # Instantiating the QModel
    knapsack_model = QModel('Battery_Optimization')
    # Adding the decision variables
    x = knapsack_model.binary_var_list(n_items, name="x")
    # Adding the degradation constraint
    knapsack_model.add_constraint(sum(weights[i] * x[i]
                                      for i in range(n_items)) <= max_weight)
    # Creating and adding the objective function
    obj_fn = sum(values[i] * x[i] for i in range(n_items))
    knapsack_model.set_objective('max', obj_fn)

    return knapsack_model
```

Figure 6.2|: Code snippet for problem modeling using the QModel from QPLEX.

```
# Run experiments
# ---------------
experiments = [
    {"provider": "d-wave"},
    {"provider": "ibmq", "algorithm": "qaoa", "p": 6,
     "shots": 5120, "max_iter": 10000},
    {"provider": "ibmq", "algorithm": "qaoa", "p": 8,
     "shots": 10240, "max_iter": 10000},
    {"provider": "ibmq", "algorithm": "vqe", "layers": 9,
     "shots": 10240, "max_iter": 10000},
    {"provider": "braket", "algorithm": "qaoa", "p": 6,
     "shots": 5120, "max_iter": 10000},
    {"provider": "braket", "algorithm": "qaoa", "p": 8,
     "shots": 10240, "max_iter": 10000},
    {"provider": "braket", "algorithm": "vqe", "layers": 9,
     "shots": 10240, "max_iter": 10000},
]
```

Figure 6.3|: Code snippet for the experiment's setup.

6.3 Validation Results

The execution of the validation experiments was sequential, and key metrics and results were recorded. These include execution time, the objective value achieved, and the resulting decision variables. All this information was recorded in plain text files. Table 6.2 summarizes the results. Additionally, Figures 6.4-6.6 provide proof that with each execution, the cloud resources of the chosen provider are being utilized.

Table 6.2: Execution results for the validation experiments.

Execution #	Execution time (s)	Objective value	Decision variables
Baseline	0.04	16	x_0, x_2, x_3, x_5, x_6
E1	12.03	16	x_0, x_2, x_3, x_5, x_6
E2	176.38	11	x_3, x_4, x_5, x_6
E3	273.71	16	x_0, x_2, x_3, x_5, x_6
E4	4698.41	16	x_0, x_2, x_3, x_5, x_6
E5	178.05	4	x_1, x_2, x_6
E6	273.20	16	x_0, x_2, x_3, x_5, x_6
E7	3948.51	16	x_0, x_2, x_3, x_5, x_6

It is possible to identify that multiple experiments performed with quantum resources achieved the target objective value with the correct combination of values for the variables involved. Nevertheless, there is a large difference in execution time between the validation cases. The details and explanation for these results are provided in the following section.

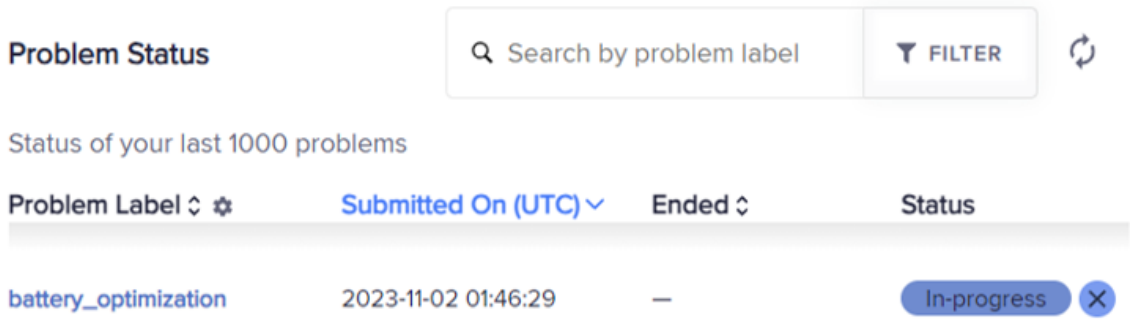

Figure 6.4|: Execution of the battery optimization problem on D-Wave's cloud Leap.

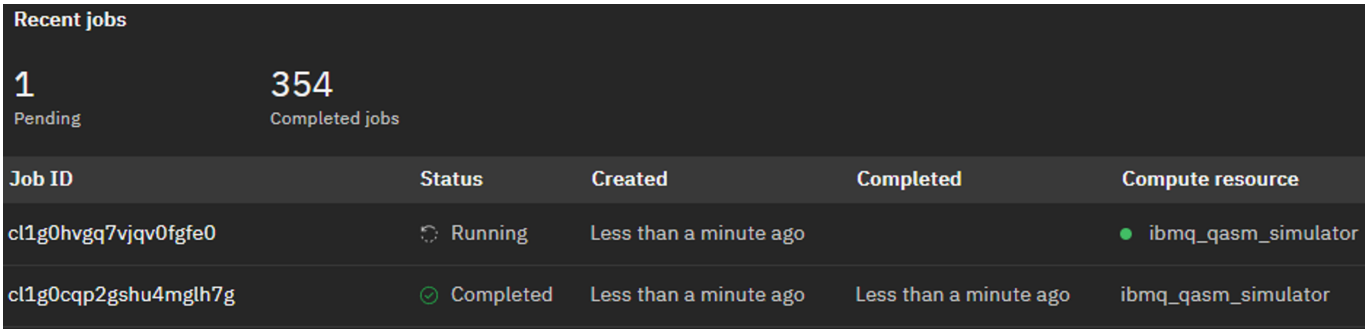

Figure 6.5|: Execution of the jobs associated with the battery optimization problem formulation on the IBM Quantum cloud, using the openQASM simulator.

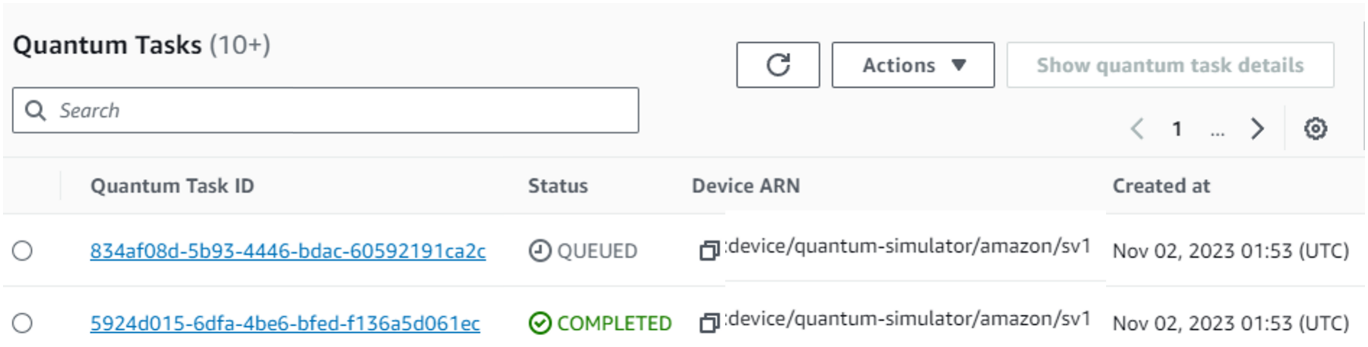

Figure 6.6|: Execution of the battery optimization tasks on the Amazon Braket cloud, using the state vector simulator.

6.4 Discussion and Limitations

Upon analyzing the execution results from the proposed experiments, we can confidently conclude that all three functional requirements set for QPLEX have been successfully met. Each of these requirements underwent one validation method, demonstrating the presence of the expected capabilities within the developed software solution.

For Requirement 1, we can utilize both Figures 6.1 and 6.2, in conjunction with the results from the baseline execution, to confirm that QPLEX enables the modeling of combinatorial optimization formulations through its classical interface. The correctness of the solution provided by the baseline execution confirms the proper modeling of the problem.

Requirement 2 has been validated through the successful execution and result retrieval from different runs using the three supported quantum platform providers: D-Wave, IBM Quantum, and Amazon Braket. Furthermore, it can be verified that these executions were appropriately conducted within a cloud environment by reviewing the provided figures. The retrieval of correct results from these executions establishes that the providers' SDKs and software tools are correctly utilized for addressing combinatorial optimization problems.

Finally, the satisfaction of Requirement 3 is confirmed through an evaluation of the variability in execution results based on the algorithmic customization provided by the parameters of each experiment. It is evident how the value of p for QAOA or the number of layers for VQE, in conjunction with the number of shots and optimizer iterations, impacts the final solution. While executions with larger values may require more time to reach a valid solution, the resulting solutions tend to be of higher quality, as observed in E3, E4, E6, and E7. The parameter configurations provided in E2 and E3, as well as in E5 and E6, are especially relevant for validating this requirement. They demonstrate how using the same provider and the same algorithm can yield substantially different solutions when a higher level of customization for the specified algorithm is applied. The flexibility to customize how the problem is solved allows the user to achieve more optimal results, as evident in Table 6.2.

Even though all the requirements were achieved, it is evident that the proposed software solution in its current state has some limitations. This becomes clear when comparing the execution times using quantum resources as opposed to using a classical solver. There are two main limiting factors within QPLEX that directly affect this execution time: lack of more specialized quantum algorithms and the direct use of the QPUs without using a hybrid system such as Qiskit Runtime or AWS Hybrid jobs. As previously mentioned, QPLEX only supports two gate-based quantum algorithms, QAOA and VQE; however, these solutions are not optimal for dealing with more complex optimization problems such as the one proposed for validation. Other algorithms, such as the Quantum Alternating Operator Ansatz (QAOAnsatz), provide a method for reaching valid solutions for constrained problems in a shorter amount of time. However, the lack of such an approach will be addressed in future iterations of the project. Also, the multiple executions performed by QPLEX to solve the

provided problem formulation are not within a hybrid system session, causing the execution queue to reset every time a new execution is going to be performed, resulting in a longer execution time. Given the current software architecture employed by QPLEX, it is a complex task to integrate access to hybrid systems to boost the performance of the library. Nonetheless, this is part of the future work. Additionally, the previously mentioned limitations inherent to quantum devices are also present within the system; however, these are out of the scope of our software solution.

6.5 Chapter summary

This chapter introduced the validation framework for QPLEX. It highlights the functional requirements expected from the solution, describes the necessary software capabilities, and outlines the validation methodology to be employed. Furthermore, it provides a detailed overview of the validation setup, encompassing the description of the use case and the formulation of the combinatorial optimization problem. The particulars on how to conduct experiments are also provided. Finally, the results, accompanied by a thorough discussion, are also presented, followed by a discussion of QPLEX's limitations.

Chapter 7

Conclusions

This thesis represents a significant endeavor aimed at advancing the domains of quantum software engineering and quantum combinatorial optimization. It accomplishes this by seamlessly integrating quantum computation into optimization software through the utilization of tools offered by classical software engineering. This ambitious task was successfully realized through the development of QPLEX, a Python programming library designed for platform-agnostic quantum combinatorial optimization. The development process of this solution adhered to the primary phases of the software engineering life cycle, including analysis, design, and implementation. Each of these stages not only facilitated the answer to the research questions at hand but also produced essential artifacts crucial for the library's development. The analysis phase, in particular, offered a profound understanding of the advantages and limitations of three distinct quantum algorithms tailored for combinatorial optimization: Quantum Annealing, VQE, and QAOA. It also enabled a comprehensive evaluation of the leading quantum platform options in the market, with a focus on three prominent quantum providers: IBM Quantum, D-Wave, and Amazon Braket. The design phase played a pivotal role in identifying the classical software engineering concepts and tools that could be harnessed to achieve our objectives. It also yielded the design artifacts that described the various modules and workflows. Lastly, the implementation stage provided a finalized product that was used for validating the correctness of our software solution and, therefore, the successful integration of quantum computing into combinatorial optimization software.

7.1 Contributions

This section restates the research questions and the contributions that address them.

Q1: Which quantum algorithms are used for solving optimization problems, and what are their strengths and weaknesses?

C1: Chapter 3 describes in detail three different quantum algorithms utilized for solving combinatorial optimization problems. It also highlights their implementation, their benefits, and limitations.

Q2: What are the most prevalent quantum computing hardware implementations, and how can the different offerings from platform providers be characterized to make an informed decision when selecting where to solve a combinatorial optimization problem?

C2: Chapter 4 presents a comprehensive review of the most popular quantum hardware implementations, analyzed the offerings of three different quantum providers, and introduced various characterization aspects for evaluating the suitability of a specific quantum device for solving a given combinatorial optimization problem.

Q3: How can knowledge from classical software engineering be applied to develop a software library for platform-agnostic quantum combinatorial optimization?

C3: Chapter 5, lays out the specifics of the architectural design and details about the implementation of QPLEX and articulates which classical software engineering tools and knowledge were used for conceiving our software solution.

7.2 Future work

This section discusses potential improvements to be further explored for extending this research.

Wider variety of quantum algorithms for combinatorial optimization

As mentioned earlier, QPLEX currently supports three distinct quantum computing algorithms for addressing combinatorial optimization problems. However, the literature reports a wider array of algorithmic techniques that can be harnessed to achieve more optimized solutions for combinatorial formulations. Examples include Quantum walks [97], the Quantum Alternating Operator Ansatz [71], and the CVaR VQE [98]. Expanding support to include these algorithms would offer additional options for experimentation and, in some cases, may lead to improved results and reduced execution times for solutions.

Larger number of quantum providers

Expanding access to a broader array of quantum computing platform providers offers greater flexibility for testing and exploring optimal solutions. Different providers offer various hardware implementations with unique low-level optimization routines and distinct device metrics. This extended accessibility facilitates experimentation for algorithm researchers, as they can seamlessly access a larger pool of QPUs through QPLEX.

Support for hybrid systems

Customizing QPLEX to use quantum-classical hybrid systems like Qiskit Runtime and Amazon Braket Hybrid Jobs would grant access to provider-specific features that, in turn, could enhance both execution speed and solution quality within the library. These features would encompass various noise mitigation levels and access to execution sessions in the case of

Qiskit Runtime. Likewise, Braket Hybrid Jobs would offer unified job execution and cloud monitoring capabilities.

Reduction of Qiskit library dependencies

Our software solution currently leverages Qiskit to enable various essential functionalities in the context of solving combinatorial optimization problems. Specifically, we employ the 'converters' module from 'qiskit-optimization' to parse combinatorial model instances into QUBOs, which can then be utilized by the supported algorithms. To enhance the efficiency and robustness of QPLEX's code base, it's necessary to reduce these dependencies, beginning with the development of custom converters designed for problem formulations involving constraints and integer variables.

Bibliography

[1] B. Tasseff, T. Albash, Z. Morrell, M. Vuffray, A. Y. Lokhov, S. Misra, and C. Coffrin, "On the emerging potential of quantum annealing hardware for combinatorial optimization," 2022. [Online]. Available: https://arxiv.org/pdf/2210.04291.pdf

[2] X. Liu, A. Angone, R. Shaydulin, I. Safro, Y. Alexeev, and L. Cincio, "Layer VQE: A variational approach for combinatorial optimization on noisy quantum computers," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022.

[3] H.-L. Huang, D. Wu, D. Fan, and X. Zhu, "Superconducting quantum computing: a review," *Science China Information Sciences*, vol. 63, no. 8, 2020. [Online]. Available: https://arxiv.org/pdf/2006.10433.pdf

[4] A. Ballon, "Trapped ion quantum computers," 2022. [Online]. Available: https://pennylane.ai/qml/demos/tutorial_trapped_ions

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. Cambridge, 2011.

[6] R. P. Feynman, "Quantum mechanical computers," *Foundations of Physics*, vol. 16, no. 6, pp. 507–531, 1986. [Online]. Available: https://link.springer.com/article/10.1007/BF01886518

[7] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Procedings Royal Society London*, 1992.

[8] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, p. 1484–1509, 1997. [Online]. Available: https://dl.acm.org/doi/10.1137/S0097539795293172

[9] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://dl.acm.org/doi/10.1145/237814.237866

[10] J. Golden, A. Bärtschi, S. Eidenbenz, and D. O'Malley, "Evidence for super-polynomial advantage of QAOA over unstructured search," 2022. [Online]. Available: https://arxiv.org/abs/2202.00648

[11] L. S. e. a. Madsen, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, pp. 75–81, 2022. [Online]. Available: https://www.nature.com/articles/s41586-022-04725-x

[12] A. Glos, M. Kokainis, R. Mori, and J. Vihrovs, "Quantum speedups for dynamic programming on n-dimensional lattice graphs," in *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, ser. LIPIcs, vol. 202. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 50:1–50:23.

[13] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018. [Online]. Available: https://quantum-journal.org/papers/q-2018-08-06-79/

[14] A. P. et al., "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, 2014.

[15] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[16] J. Geunes and P. M. Pardalos, *Supply chain optimization*. Springer, 2005.

[17] K. Gupta and M. K. Gupta, *Optimization of Manufacturing Processes*. Springer Nature, 2020.

[18] G. Cornuejols, J. F. Peña, and R. Tütüncü, *Optimization Methods in Finance*. Cambridge University Press, 2018.

[19] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. A. Negre, I. Safro, S. M. Mniszewski, and Y. Alexeev, "A hybrid approach for solving optimization problems on small quantum computers," *Computer*, vol. 52, no. 6, pp. 18–26, 2019.

[20] E. Zahedinejad and A. Zaribafiyan, "Combinatorial optimization on gate model quantum computers: A survey," 2017. [Online]. Available: https://arxiv.org/abs/1708.05294

[21] A. Ajagekar, K. A. Hamoud, and F. You, "Hybrid classical-quantum optimization techniques for solving mixed-integer programming problems in production scheduling," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–16, 2022.

[22] S. Mukherjee and B. Chakrabarti, "Multivariable optimization: Quantum annealing and computation," *The European Physical Journal Special Topics*, pp. 17–24, 2015.

[23] C. C. Mcgeoch and C. Wang, "Experimental evaluation of an adiabiatic quantum system for combinatorial optimization," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '13, vol. 13. ACM, 2013.

[24] 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, 2014.

[25] B. Weder, J. Barzen, F. Leymann, M. Salm, and D. Vietz, "The quantum software lifecycle," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software*. Association for Computing Machinery, 2020, p. 2–9. [Online]. Available: https://dl.acm.org/doi/10.1145/3412451.3428497

[26] A. García de la Barrera, I. García-Rodríguez de Guzmán, M. Polo, and M. Piattini, "Quantum software testing: State of the art," *Journal of Software: Evolution and Process*, vol. 35, no. 4, p. e2419, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2419

[27] I.-D. Gheorghe-Pop, N. Tcholtchev, T. Ritter, and M. Hauswirth, "Quantum devops: Towards reliable and applicable nisq quantum computing," in *2020 IEEE Globecom Workshops (GC Wkshps)*, 2020, pp. 1–6.

[28] G. Rosenberg, P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, and M. L. de Prado, "Solving the optimal trading trajectory problem using a quantum annealer," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 6, pp. 1053–1060, 2016.

[29] S. Harwood, C. Gambella, D. Trenev, A. Simonetto, D. Bernal, and D. Greenberg, "Formulating and solving routing problems on quantum computers," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–17, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9314905

[30] A. Ajagekar and F. You, "Quantum computing for energy systems optimization: Challenges and opportunities," *Energy*, vol. 179, pp. 76–89, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544219308254

[31] A. Awasthi, F. Bär, J. Doetsch, H. Ehm, M. Erdmann, M. Hess, J. Klepsch, P. A. Limacher, A. Luckow, C. Niedermeier, L. Palackal, R. Pfeiffer, P. Ross, H. Safi, J. Schönmeier-Kromer, O. von Sicard, Y. Wenger, K. Wintersperger, and S. Yarkoni, "Quantum computing techniques for multi-knapsack problems," in *Intelligent Computing*. Springer Nature Switzerland, 2023, pp. 264–284. [Online]. Available: https://arxiv.org/pdf/2301.05750.pdf

[32] J. Obst, J. Barzen, M. Beisel, F. Leymann, M. Salm, and F. Truger, "Comparing quantum service offerings: A case study of qaoa for maxcut," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2023. [Online]. Available: https://arxiv.org/pdf/2304.12718.pdf

[33] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Phys. Rev. Lett.*, vol. 70, pp. 1895–1899, 1993. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.70.1895

[34] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997. [Online]. Available: https://doi.org/10.1137/S0097539796300921

[35] A. Y. Kitaev, "Quantum measurements and the abelian stabilizer problem," *Electron. Colloquium Comput. Complex.*, vol. TR96, 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:17023060

[36] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, 2018. [Online]. Available: https://arxiv.org/abs/1611.04471

[37] W. Vinci and D. A. Lidar, "Non-stoquastic hamiltonians in quantum annealing via geometric phases," *npj Quantum Information*, vol. 3, no. 1, 2017. [Online]. Available: https://arxiv.org/abs/1701.07494

[38] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of quantum annealing: methods and implementations," *Reports on Progress in Physics*, vol. 83, no. 5, 2020. [Online]. Available: https://arxiv.org/pdf/1903.06559.pdf

[39] S. Endo, Z. Cai, S. C. Benjamin, and X. Yuan, "Hybrid quantum-classical algorithms and quantum error mitigation," *Journal of the Physical Society of Japan*, vol. 90, no. 3, p. 032001, 2021. [Online]. Available: https://journals.jps.jp/action/showCitFormats?doi=10.7566%2FJPSJ.90.032001

[40] M. Suchara, Y. Alexeev, F. Chong, H. Finkel, H. Hoffmann, J. Larson, J. Osborn, , and G. Smith, "Hybrid quantum-classical computing architectures," *Proceedings of the 3rd International Workshop on Post-Moore Era Supercomputing, 2018.*, 2018. [Online]. Available: https://par.nsf.gov/biblio/10084839

[41] D.-Z. Du, P. M. Pardalos, X. Hu, and W. Wu, *Introduction to combinatorial optimization*. Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-10596-8?utm_medium=referral&utm_source=springer&utm_content=RM&utm_term=null&utm_campaign=HSCR_ALLPR_AWA1_GL_MPAS_005JU_MATH-COMB

[42] B. Korte and J. Vygen, *Combinatorial optimization: Theory and Algorithms*. Springer Berlin Heidelberg, 2018. [Online]. Available: https://link.springer.com/book/10.1007/978-3-662-56039-6

[43] A. Schrijver, "On the history of combinatorial optimization (till 1960)," in *Discrete Optimization*, ser. Handbooks in Operations Research and Management Science, K. Aardal, G. Nemhauser, and R. Weismantel, Eds. Elsevier, 2005, vol. 12, pp. 1–68. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705120015

[44] R. M. Karp, "Reducibility among combinatorial problems." Springer US, 1972, pp. 85–103. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4684-2001-2_9

[45] S. A. Cook, "The complexity of theorem-proving procedures." ACM, 1971, p. 151–158. [Online]. Available: https://dl.acm.org/doi/10.1145/800157.805047

[46] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, ser. Mathematical Sciences Series. Freeman, 1979. [Online]. Available: https://books.google.ca/books?id=fjxGAQAAIAAJ

[47] E. Ising, "Beitrag zur theorie des ferromagnetismus," *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, 1925. [Online]. Available: https://link.springer.com/article/10.1007/BF02980577

[48] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "T|ket⟩: a retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020. [Online]. Available: https://arxiv.org/abs/2003.10611

[49] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," 2022. [Online]. Available: https://arxiv.org/abs/1811.04968

[50] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Phys. Rev. E*, vol. 58, pp. 5355–5363, 1998. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.58.5355

[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[52] S. Mukherjee and B. Chakrabarti, "Multivariable optimization: Quantum annealing and computation," *The European Physical Journal Special Topics*, vol. 224, no. 1, pp. 17–24, 2015. [Online]. Available: https://arxiv.org/pdf/1408.3262.pdf

[53] T. Zaborniak, J. Giraldo, H. Muller, H. Jabbari, and U. Stege, "A QUBO model of the RNA folding problem optimized by variational hybrid quantum annealing," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 174–185. [Online]. Available: https://arxiv.org/pdf/2208.04367.pdf

[54] P. Schworm, X. Wu, M. Glatt, and J. C. Aurich, "Solving flexible job shop scheduling problems in manufacturing with quantum annealing," *Production Engineering*, vol. 17, no. 1, pp. 105–115, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s11740-022-01145-8

[55] S. J. Weinberg, F. Sanches, T. Ide, K. Kamiya, and R. Correll, "Supply chain logistics with quantum and classical annealing algorithms," *Scientific Reports*, vol. 13, no. 1, p. 4770, 2023. [Online]. Available: https://www.nature.com/articles/s41598-023-31765-8

[56] R. K. Nath, H. Thapliyal, and T. S. Humble, "A review of machine learning classification using quantum annealing for real-world applications," *SN Computer Science*, vol. 2, no. 5, p. 365, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42979-021-00751-0

[57] F. M. Fernández, "On the rayleigh-ritz variational method," 2022. [Online]. Available: https://arxiv.org/pdf/2206.05122.pdf

[58] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, and J. Tennyson, "The variational quantum eigensolver: A review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022. [Online]. Available: https://arxiv.org/pdf/2111.05176.pdf

[59] H. R. Grimsley, S. E. Economou, E. Barnes, and N. J. Mayhall, "An adaptive variational algorithm for exact molecular simulations on a quantum computer," *Nature Communications*, vol. 10, no. 1, p. 3007, 2019. [Online]. Available: https://www.nature.com/articles/s41467-019-10988-2

[60] H. L. Tang, V. Shkolnikov, G. S. Barron, H. R. Grimsley, N. J. Mayhall, E. Barnes, and S. E. Economou, "Qubit-ADAPT-VQE: An adaptive algorithm for constructing hardware-efficient ansätze on a quantum processor," *PRX Quantum*, vol. 2, no. 2, 2021. [Online]. Available: https://arxiv.org/pdf/1911.10205.pdf

[61] R. Watanabe, K. Fujii, and H. Ueda, "Entangled embedding variational quantum eigensolver with tensor network ansatz," 2023. [Online]. Available: https://arxiv.org/pdf/2305.06536.pdf

[62] D. Amaro, C. Modica, M. Rosenkranz, M. Fiorentini, M. Benedetti, and M. Lubasch, "Filtering variational quantum algorithms for combinatorial optimization," *Quantum Science and Technology*, vol. 7, no. 1, p. 015021, 2022. [Online]. Available: https://arxiv.org/pdf/2106.10055.pdf

[63] A. Pellow-Jarman, I. Sinayskiy, A. Pillay, and F. Petruccione, "A comparison of various classical optimizers for a variational quantum linear solver," *Quantum Information Processing*, vol. 20, no. 6, 2021. [Online]. Available: https://arxiv.org/pdf/2106.08682.pdf

[64] A. Bentellis, A. Matic-Flierl, C. B. Mendl, and J. M. Lorenz, "Benchmarking the variational quantum eigensolver using different quantum hardware," 2023. [Online]. Available: https://arxiv.org/pdf/2305.07092.pdf

[65] A. V. Uvarov and J. D. Biamonte, "On barren plateaus and cost function locality in variational quantum algorithms," *Journal of Physics A: Mathematical and Theoretical*, vol. 54, no. 24, p. 245301, 2021. [Online]. Available: https://arxiv.org/pdf/2011.10530v2.pdf

[66] Z. Holmes, K. Sharma, M. Cerezo, and P. J. Coles, "Connecting ansatz expressibility to gradient magnitudes and barren plateaus," *PRX Quantum*, vol. 3, no. 1, 2022. [Online]. Available: https://arxiv.org/pdf/2101.02138v2.pdf

[67] L. Binkowski, G. Koßmann, T. Ziegler, and R. Schwonnek, "Elementary proof of qaoa convergence," 2023. [Online]. Available: https://arxiv.org/pdf/2302.04968.pdf

[68] M. Suzuki, "Generalized trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems," *Communications in Mathematical Physics*, vol. 51, no. 2, pp. 183–190, 1976. [Online]. Available: https://link.springer.com/article/10.1007/BF01609348

[69] S. Boulebnane and A. Montanaro, "Solving boolean satisfiability problems with the quantum approximate optimization algorithm," 2022. [Online]. Available: https://arxiv.org/pdf/2208.06909.pdf

[70] S. Brandhofer, D. Braun, V. Dehn, G. Hellstern, M. Hüls, Y. Ji, I. Polian, A. S. Bhatia, and T. Wellens, "Benchmarking the performance of portfolio optimization with qaoa," 2022. [Online]. Available: https://arxiv.org/pdf/2207.10555.pdf

[71] S. Hadfield, Z. Wang, B. O'Gorman, E. 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. [Online]. Available: https://arxiv.org/pdf/1709.03489.pdf

[72] D. J. Egger, J. Mareč ek, and S. Woerner, "Warm-starting quantum optimization," *Quantum*, vol. 5, p. 479, 2021. [Online]. Available: https://arxiv.org/pdf/2009.10095.pdf

[73] E. Pelofske, A. Bärtschi, and S. Eidenbenz, "Quantum annealing vs. QAOA: 127 qubit higher-order ising problems on NISQ computers," in *Lecture Notes in Computer Science*. Springer Nature Switzerland, 2023, pp. 240–258. [Online]. Available: https://arxiv.org/pdf/2301.00520.pdf

[74] Y. Nakamura, Y. A. Pashkin, and J. S. Tsai, "Coherent control of macroscopic quantum states in a single-cooper-pair box," *Nature*, vol. 398, no. 6730, pp. 786–788, 1999. [Online]. Available: https://arxiv.org/pdf/cond-mat/9904003.pdf

[75] T. P. Orlando, J. E. Mooij, L. Tian, C. H. van der Wal, L. S. Levitov, S. Lloyd, and J. J. Mazo, "Superconducting persistent-current qubit," *Physical Review B*, vol. 60, no. 22, pp. 15 398–15 413, 1999. [Online]. Available: https://arxiv.org/pdf/cond-mat/9908283.pdf

[76] J. M. Martinis, "Superconducting phase qubits," *Quantum Information Processing*, vol. 8, no. 2, pp. 81–103, 2009. [Online]. Available: https://link.springer.com/article/10.1007/s11128-009-0105-1

[77] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, 2007. [Online]. Available: https://arxiv.org/pdf/cond-mat/0703002.pdf

[78] V. E. Manucharyan, J. Koch, L. I. Glazman, and M. H. Devoret, "Fluxonium: Single cooper-pair circuit free of charge offsets," *Science*, vol. 326, no. 5949, pp. 113–116, 2009. [Online]. Available: https://arxiv.org/pdf/0906.0831.pdf

[79] E. Hyyppä, S. Kundu, C. F. Chan, A. Gunyhó, J. Hotari, D. Janzso, K. Juliusson, O. Kiuru, J. Kotilahti, A. Landra, W. Liu, F. Marxer, A. Mäkinen, J.-L. Orgiazzi, M. Palma, M. Savytskyi, F. Tosto, J. Tuorila, V. Vadimov, T. Li, C. Ockeloen-Korppi, J. Heinsoo, K. Y. Tan, J. Hassel, and M. Möttönen, "Unimon qubit," *Nature Communications*, vol. 13, no. 1, 2022. [Online]. Available: https://arxiv.org/pdf/2203.05896.pdf

[80] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.74.4091

[81] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, p. 021314, 2019. [Online]. Available: https://arxiv.org/pdf/1904.04178.pdf

[82] J. E. Bourassa, R. N. Alexander, M. Vasmer, A. Patil, I. Tzitrin, T. Matsuura, D. Su, B. Q. Baragiola, S. Guha, G. Dauphinais, K. K. Sabapathy, N. C. Menicucci, and I. Dhand, "Blueprint for a scalable photonic fault-tolerant quantum computer," *Quantum*, vol. 5, p. 392, 2021. [Online]. Available: https://arxiv.org/pdf/2010.02905.pdf

[83] J. M. Arrazola, V. Bergholm, K. Brádler, T. R. Bromley, M. J. Collins, I. Dhand, A. Fumagalli, T. Gerrits, A. Goussev, L. G. Helt, J. Hundal, T. Isacsson, R. B. Israel, J. Izaac, S. Jahangiri, R. Janik, N. Killoran, S. P. Kumar, J. Lavoie, A. E. Lita, D. H. Mahler, M. Menotti, B. Morrison, S. W. Nam, L. Neuhaus, H. Y. Qi, N. Quesada, A. Repingon, K. K. Sabapathy, M. Schuld, D. Su, J. Swinarton, A. Száva, K. Tan, P. Tan, V. D. Vaidya, Z. Vernon, Z. Zabaneh, and Y. Zhang, "Quantum circuits with many photons on a programmable nanophotonic chip," *Nature*, vol. 591, no. 7848, pp. 54–60, 2021. [Online]. Available: https://www.nature.com/articles/s41586-021-03202-1

[84] J. L. O'Brien, "Optical quantum computing," *Science*, vol. 318, no. 5856, pp. 1567–1570, 2007. [Online]. Available: https://arxiv.org/pdf/0803.1554.pdf

[85] E. Knill, R. Laflamme, and G. J. Milburn, "A scheme for efficient quantum computation with linear optics," *Nature*, vol. 409, no. 6816, pp. 46–52, 2001. [Online]. Available: https://www.nature.com/articles/35051009

[86] S. L. Braunstein and P. van Loock, "Quantum information with continuous variables," *Reviews of Modern Physics*, vol. 77, no. 2, pp. 513–577, 2005. [Online]. Available: https://arxiv.org/pdf/quant-ph/0410100.pdf

[87] S. Takeda and A. Furusawa, "Toward large-scale fault-tolerant universal photonic quantum computing," *APL Photonics*, vol. 4, no. 6, jun 2019. [Online]. Available: https://arxiv.org/pdf/1904.07390.pdf

[88] A. Montanez-Barrera, D. Willsch, A. Maldonado-Romo, and K. Michielsen, "Unbalanced penalization: A new approach to encode inequality constraints of combinatorial problems for quantum optimization algorithms," 2023. [Online]. Available: https://arxiv.org/pdf/2211.13914.pdf

[89] Y. Chatterjee, E. Bourreau, and M. J. Rančić, "Solving various np-hard problems using exponentially fewer qubits on a quantum computer," 2023. [Online]. Available: https://arxiv.org/pdf/2301.06978.pdf

[90] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers," 2021. [Online]. Available: https://arxiv.org/pdf/2110.14108.pdf

[91] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, 2019. [Online]. Available: https://arxiv.org/pdf/1811.12926.pdf

[92] A. J. McCaskey, Z. P. Parks, J. Jakowski, S. V. Moore, T. D. Morris, T. S. Humble, and R. C. Pooser, "Quantum chemistry as a benchmark for near-term quantum computers," *npj Quantum Information*, vol. 5, no. 1, p. 99, 2019. [Online]. Available: https://www.nature.com/articles/s41534-019-0209-0

[93] T. Lubinski, C. Coffrin, C. McGeoch, P. Sathe, J. Apanavicius, and D. E. B. Neira, "Optimization applications as quantum performance benchmarks," 2023. [Online]. Available: https://arxiv.org/pdf/2302.02278.pdf

[94] E. Pelofske, J. Golden, A. Bartschi, D. O'Malley, and S. Eidenbenz, "Sampling on NISQ devices: "who's the fairest one of all?"," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021. [Online]. Available: https://arxiv.org/pdf/2107.06468.pdf

[95] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: A survey," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800220110

[96] P. D. de la Grand'rive and J.-F. Hullo, "Knapsack problem variants of qaoa for battery revenue optimization," 2019. [Online]. Available: https://arxiv.org/pdf/1908.02210.pdf

[97] S. Marsh and J. B. Wang, "Combinatorial optimization via highly efficient quantum walks," *Physical Review Research*, vol. 2, no. 2, 2020. [Online]. Available: https://doi.org/10.1103%2Fphysrevresearch.2.023302

[98] P. K. Barkoutsos, G. Nannicini, A. Robert, I. Tavernelli, and S. Woerner, "Improving variational quantum optimization using CVaR," *Quantum*, vol. 4, p. 256, 2020. [Online]. Available: https://doi.org/10.22331%2Fq-2020-04-20-256