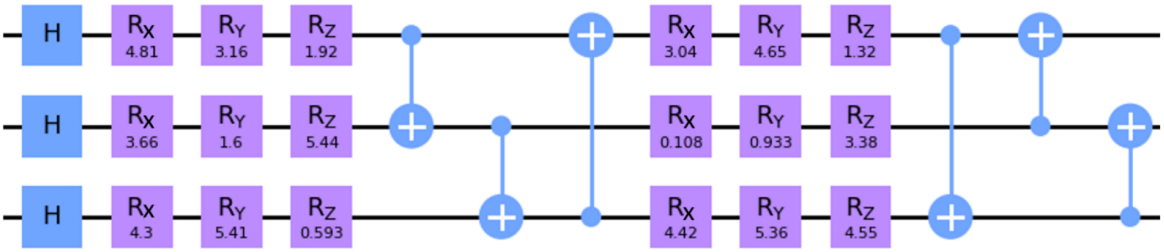
TÉCNICO LISBOA



Development of a new variational quantum algorithm for MaxCut: QAOA and QEMC hybrid algorithm

Afonso Sequeira Azenha

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisor(s): Prof. Yasser Rashid Revez Omar
Prof. Joao Carlos Carvalho de Sá Seixas

Examination Committee

Chairperson: Prof. Full Name 1
Supervisor: Prof. Full Name 2
Member of the Committee: Prof. Full Name 3

June 2024

Dedicated to someone special...

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

A few words about the university, financial support, research advisor, dissertation readers, faculty or other professors, lab mates, other friends and family...

This work is being developed in collaboration with Zoltán Zimborás and Bence Bako, from the Wigner Research Centre for Physics (Hungary), in the context of project: *HQCC – Hybrid Quantum-Classical Computing*, supported by the EU QuantERA ERA-NET Co-fund in Quantum Technologies and by FCT – Fundação para a Ciência e a Tecnologia (QuantERA/004/2021).

Resumo

Inserir o resumo em Português aqui com um máximo de 250 palavras e acompanhado de 4 a 6 palavras-chave.

Palavras-chave: palavra-chave1, palavra-chave2, palavra-chave3,...

Abstract

Insert your abstract here with a maximum of 250 words, followed by 4 to 6 keywords.

Keywords: keyword1, keyword2, keyword3,...

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

$|\psi\rangle$ Arbitrary pure state

ρ Density matrix or performance guarantee

σ_x, σ_y, σ_z Pauli matrices

Roman symbols

$\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ Pauli matrices (alternative notation)

H_C or H_P Cost or problem Hamiltonian.

H_M or H_B Mixer or bias Hamiltonian.

$R_x(\theta), R_y(\theta), R_z(\theta)$ Rotation (of angle θ) gates around the x, y and z axes, on the Bloch sphere.

S_n n-dimensional unit sphere

Tr Trace [of a matrix]

Tr_A Partial trace [over subsystem A]. "Tracing out subsystem A."

Subscripts

x, y, z Cartesian components

Superscripts

† Hermitian adjoint

T Transpose

List of acronyms

AR	Approximation Ratio
GW	Goemans-Williamson Algorithm
HQCC	Hybrid Quantum-Classical Computing
iQAQE	Interpolated QAOA/QEMC Hybrid Algorithm
MaxCut	Maximum Cut Problem
NISQ	Noisy Intermediate-Scale Quantum
PQC	Parameterized Quantum Circuit
QAOA	Quantum Approximate Optimization Algorithm
QEMC	Qubit-Efficient MaxCut Heuristic Algorithm
QKD	Quantum Key Distribution
QML	Quantum Machine Learning
SAT	Boolean Satisfiability Problem
SDP	Semidefinite Programming
TSP	Traveling Salesman Problem
VQA	Variational Quantum Algorithm
VQE	Variational Quantum Eigensolver

Chapter 1

Introduction

Over the last few decades, significant progress has been achieved in the field of quantum computing. This is an entirely new computing paradigm, based on exploiting the fundamental principles of quantum mechanics to our advantage. Although the concept of a quantum computer has been around for a little over 40 years [1], only rather recently, in the present century, have we successfully engineered elementary prototypes for such cutting-edge devices. Scientists and engineers worldwide are now working towards the development of practical quantum computers, which are expected to revolutionize the way we solve complex problems in various fields, such as cryptography, optimization, drug discovery, material science, machine learning, and many more.

However, with the current generation of quantum machines, so-called Noisy Intermediate Scale Quantum (NISQ) devices, we are still far from achieving the promised quantum advantage. This refers to the point at which a quantum computer can outperform its classical counterpart by solving specific problems considerably faster or more efficiently. It's the "holy grail" of quantum computing and what drives the research in this field forwards.

Presently, the most promising candidates for achieving meaningful quantum advantage are anchored in what has become known as "Hybrid Quantum-Classical Computing" (HQCC). This approach aims to merge the strengths of both worlds: the computational power of quantum and the stability of classical computing. In this context, Variational Quantum Algorithms (VQA) have been at the forefront of research, as they are particularly well-suited for near-term quantum devices, requiring fewer qubits and featuring shallower circuit depths. These algorithms are hybrid by design, using a classical optimizer to adjust the parameters of a Parameterized Quantum Circuit (PQC).

Amid the many prospective applications of quantum computing, notable advances have been made in recent years in solving combinatorial optimization problems. VQAs like the Quantum Approximate Optimization Algorithm (QAOA) [2] have demonstrated potential for tackling the Maximum Cut (MaxCut) problem, a challenging graph-based NP-hard problem in computer science. Despite its promise, however, QAOA has a significant drawback: it requires a large number of qubits, exceeding the capacity of current quantum devices, when scaled to larger problem instances. This constraint has prompted the development of alternative approaches such as the Qubit-Efficient MaxCut Heuristic Algorithm (QEMC)

[3], designed to address the MaxCut problem using fewer qubits. Meanwhile, the search for better algorithms to solve this problem continues, and the development of new hybrid quantum-classical methods is crucial to achieving quantum advantage.

1.1 Motivation

The search for efficient algorithms to solve combinatorial optimization problems is critical in numerous fields, such as logistics, finance, and telecommunications. The MaxCut problem, for example, has broad applications in areas like machine learning [4], statistical physics [5], circuit design [5], and data clustering [6]. Creating more efficient algorithms to solve this problem can improve the performance of these applications, driving substantial progress in their respective fields.

Moreover, there is a strong interest from the computer science community in terms of computational complexity. The MaxCut problem is recognized as NP-hard, and the search for efficient algorithms to solve it may offer valuable insights into the boundaries between classical and quantum computing. It might even contribute to unraveling one of the most perplexing questions in theoretical computer science: the $P = NP$ problem. Imagining a world where the $P = NP$ conjecture is proven true, albeit improbable, is intriguing. It would mean that every problem in NP could be solved in polynomial time, including MaxCut. This would also extend to problems like the Traveling Salesman Problem (TSP) and the Knapsack Problem, among others. The implications would be profound, as this would dramatically increase our ability to solve previously difficult optimization problems, with direct applications in areas like vehicle routing, job scheduling, and broader logistics. Additionally, the impact on cryptography would be as significant – if not more so – since many cryptographic techniques rely on the complexity of NP problems. For example, integer factorization is a key component of RSA (Rivest-Shamir-Adleman) encryption, a popular asymmetric encryption algorithm used extensively in public-key cryptography for secure message transmission over the internet. If $P = NP$, RSA encryption could easily be broken, leading to major security risks. Hence, the importance of studying these problems to fully understand their complexity.

The aforementioned considerations drive our efforts to develop a new algorithm for solving the MaxCut problem with greater efficiency and accuracy. This algorithm, the Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE), will be the focus of this thesis.

1.2 Topic Overview

In this project, we propose a new VQA, the Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE). This algorithm combines the strengths of two existing VQAs, QAOA and QEMC, for improved performance in solving the MaxCut problem. As previously mentioned, QAOA requires a qubit for each graph vertex, making it difficult to scale. In contrast, QEMC uses exponentially fewer qubits by assigning one basis state to each graph node, requiring only $\log_2(n)$ qubits (for n graph vertices). However, this compression leads to limitations in QEMC's results. By interpolating both VQAs, we aim to create an

algorithm that utilizes fewer qubits than QAOA and performs better than QAOA and QEMC. The new algorithm, iQAQE, assigns multiple basis states to each node, unlike QEMC's single basis state approach. This design tentatively allows for a more practical implementation on present-day NISQ devices, thanks to its reduced qubit requirements compared to QAOA, fewer measurement shots than QEMC, and potentially greater trainability than QAOA.

1.3 Objectives and Deliverables

The primary objective of this thesis is to develop and analyze the iQAQE algorithm. The algorithm will be implemented and tested using classical simulations of quantum machines. The deliverables for this project include the iQAQE algorithm's code, the results obtained from the simulations, and the analysis of these results. The expected outcomes are improvements in the performance of the iQAQE algorithm compared to QAOA and QEMC, with a focus on accuracy, efficiency, and scalability.

1.4 Thesis Outline

This thesis is structured as follows: Chapter 2 provides an overview of the background concepts related to quantum computing, variational quantum algorithms, and the MaxCut problem. Chapter 3 introduces the base hybrid quantum-classical algorithm, iQAQE, explaining its design, advantages, and potential applications. Chapter 4 details the numerical implementation of the iQAQE algorithm, including the individual algorithms QAOA and QEMC. It also describes the benchmarking and testing methods used to evaluate the algorithm's performance. Chapter 5 presents the schemes and results obtained from the simulations, comparing iQAQE with QAOA and QEMC. It also outlines the various iQAQE variations that were explored. Finally, Chapter 6 concludes the thesis, summarizing the main findings and suggesting future research directions.

Chapter 2

Theoretical background

This chapter covers the key concepts and theoretical background necessary to understand the work presented in this thesis. It begins by describing the MaxCut problem and discussing elements of computational complexity, reinforcing the rationale behind this research. The state-of-the-art algorithms for solving the MaxCut problem, both classical and quantum, are then examined. After a brief introduction to quantum computing, the chapter explores hybrid quantum-classical computing and variational quantum algorithms in greater depth. Specifically, the Quantum Approximate Optimization Algorithm (QAOA) and the Qubit-Efficient MaxCut Heuristic (QEMC) are analyzed, as they play a pivotal role in this study.

2.1 The Maximum Cut Problem & Computational Complexity

The MaxCut problem, a fundamental problem in graph theory and combinatorial optimization, involves partitioning a graph $G = (V, E)$ into two disjoint subsets, S_1 and S_2, such that the number of edges connecting vertices from different subsets is maximized. Formally, the objective is to find a partition (S_1, S_2) that maximizes:

$$\text{Cut}(S_1, S_2) = \sum_{(u,v)\in E} \chi(u, v), \tag{2.1}$$

where $\chi(u, v) = 1$ if u and v belong to different subsets, and $\chi(u, v) = 0$ if they belong to the same subset. This quantity is referred to as the ”cut” of the partition (S_1, S_2). Pictorially, this can be represented as cutting the edges of the graph (Figure 2.1), hence the name MaxCut. What we describe here is the undirected, un-weighted MaxCut problem. A more general formulation would involve the specific graph's adjacency matrix, W_{ij}.

Since the MaxCut problem is NP-hard, finding an optimal solution efficiently is a significant computational challenge, especially as the graph size grows. However, researchers have developed various approximation algorithms and heuristics to approach near-optimal solutions in a reasonable time, including both classical and quantum approaches (cf. subsections 2.1.1 and 2.1.2). These methods are designed to tackle the intrinsic complexity of the problem, providing practical solutions that have real-

world applications. As mentioned earlier, the MaxCut problem finds use in a variety of fields, including machine learning [4], statistical physics [5], circuit design [5], and data clustering [6].

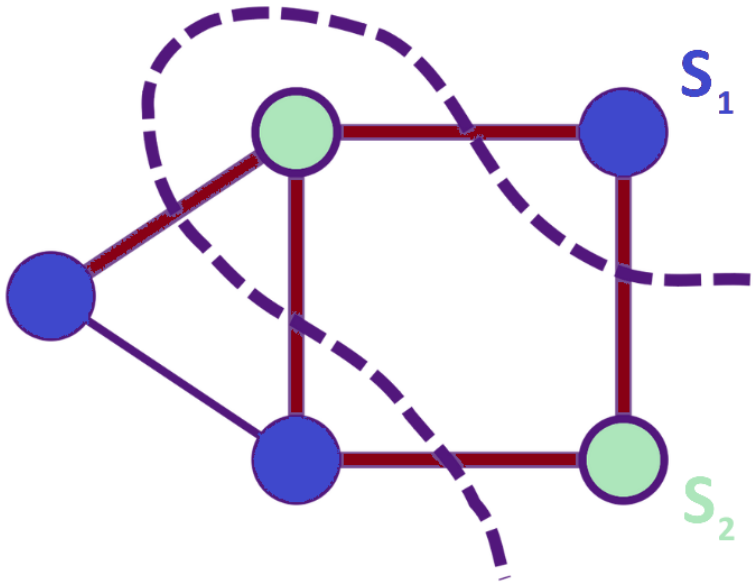


Figure 2.1: An example of a graph with a partition that maximizes the number of cut edges (red). Note that the MaxCut partition might not be unique.

We previously mentioned that the MaxCut problem is well-known to be NP-hard. (Indeed, it [decision version] even appears on Karp's original list of NP-complete problems, from 1972 [7].) Let's explore what this means in more detail. The computational complexity class of a problem is defined by the computational resources required to solve it, such as time or memory. Although there are many complexity classes (see the Complexity Zoo), the following four are key to our discussion:

1. **P (Polynomial time)**: Problems that can be solved in polynomial time by a deterministic Turing machine, indicating they are computationally efficient.
2. **NP (Non-deterministic Polynomial time)**: Problems that can be verified in polynomial time if given a solution, but finding the solution may not be as straightforward. Therefore, if you guess a solution, you can confirm its correctness quickly.
3. **NP-hard (Non-deterministic Polynomial time hard)**: Problems that are at least as challenging as the most difficult problems in NP. If you could solve an NP-hard problem in polynomial time, you would be able to solve all problems in NP efficiently.
4. **NP-complete (Non-deterministic Polynomial time complete)**: Problems that belong to NP but are also NP-hard. If you can solve one NP-complete problem efficiently, all problems in NP would become efficiently solvable.

The MaxCut problem in its **optimization form** – "Given a graph G, find the maximum cut" – is NP-hard. This implies that if you could find an efficient solution to MaxCut, you could also solve all other NP

problems reliably. That would include, e.g., the boolean SATisfiability problem (SAT), knapsack problem, decision version of the traveling salesman problem, clique problem, and more [8]. This potential for broad application drives interest in developing high-performance algorithms for MaxCut. Alternatively, the MaxCut problem can be formulated as a **decision problem** – "Given a graph G and an integer k, determine if there is a cut of size at least k in G." This formulation is NP-complete. NP-complete status indicates that a problem is both in NP, allowing polynomial-time verification, and at least as hard as any other problem in NP, making it central to computational theory and offering crucial insights into the boundary between efficiently solvable and intractable problems. Although our focus is on the NP-hard version of the problem, it's important to understand the significance of the NP-complete formulation.

2.1.1 Classical State-of-the-Art Algorithms for MaxCut

Over the years, a variety of algorithms have been proposed to solve the MaxCut problem, ranging from exact solutions to various approximations, including heuristics and other techniques. In this work, we focus on scaling these algorithms to handle bigger graphs, which is why exact algorithms are not discussed – they quickly become impractical for large instances. Instead, our attention is on approximation algorithms, which are more suitable for larger scales[1].

Approximation algorithms aim to find solutions that are close to optimal, often with a guaranteed Approximation Ratio (AR). This is a measure of the performance of an approximation algorithm, representing the worst-case deviation between the algorithm's solution and the optimal solution. Formally, this can be expressed as: (For the Goemans-Williamson (GW) Algorithm [9], e.g.)[2]

$$r_A^{GW} = \frac{\text{MaxCut}_{\text{GW}}}{\text{MaxCut}_{\text{Opt.}}} \approx 0.87856,$$

where $\text{MaxCut}_{\text{GW}}$ is the worst-case MaxCut value obtained by the Goemans-Williamson Algorithm, and $\text{MaxCut}_{\text{Opt.}}$ is the optimal MaxCut value. The AR is a crucial metric for evaluating the quality of an approximation algorithm, providing insight into its effectiveness. The closer the AR is to 1, the better the algorithm's performance. In literature, this worst-case approximation ratio might also be referred to as the *performance guarantee* ρ, often appearing alongside designations such as ρ-approximation algorithms. Using this nomenclature, the Goemans-Williamson Algorithm is a 0.87856-approximation algorithm for MaxCut (i.e., $\rho = 0.87856$).

Building on this, two commonly used **approximation algorithms** are:

- **Goemans-Williamson Algorithm** [9]: This popular approximation algorithm achieves a performance guarantee of approximately 0.87856, leveraging Semidefinite Programming (SDP) and a randomized rounding technique. (This will be further developed below, in subsubsection Goemans-Williamson Algorithm.)
- **Spectral Methods** [10]: These techniques employ eigenvalues and eigenvectors of the adjacency matrix to identify effective cuts. While spectral methods may offer quicker computational speeds

[1] In this work, "larger/bigger graphs" will often refer to graphs with many hundreds to a few thousand nodes.
[2] The actual value is $\alpha = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} > 0.878$.

compared to other approximation strategies (e.g. GW), they may not consistently provide the optimal approximation ratios. Performance-wise, the leading spectral algorithm, Trevisan's algorithm, using Soto's refined analysis [11], features a worst-case approximation ratio of 0.614.

Heuristic and **metaheuristic algorithms** offer another approach, focusing on finding solutions more quickly, albeit without guaranteed approximation ratios. They are typically employed when computational efficiency is prioritized over achieving the best possible approximation ratio. These algorithms include:

- **Greedy Algorithms** (Described in [10]): These build solutions incrementally, following a greedy approach. Mathematically, we start with $S_1, S_2 = \emptyset$. In each step of the algorithm, we choose a vertex $v \in V$ yet to be assigned to S_1 or S_2. Then, we add v to S_1 or S_2 by choosing the larger of the two cuts $(S_1 \cup \{v\}, S_2)$ and $(S_1, S_2 \cup \{v\})$ at this step. Repeat until all vertices are assigned to a set.
- **Simulated Annealing** [12]: Employs a probabilistic optimization approach, inspired by metallurgical annealing, to iteratively explore the solution space, gradually reducing the acceptance of worse solutions ("temperature") over time to converge towards near-optimal solutions.
- **Genetic Algorithms** [13]: These algorithms, rooted in evolutionary principles, employ selection, crossover, and mutation techniques to iteratively generate solutions for the MaxCut problem, evolving from an initial population of diverse partitions towards improved solutions.

Reiterating, these approximation and heuristic methods offer scalable solutions for the MaxCut problem, making them more suitable for larger graphs where exact methods are not feasible.

Of all the mentioned approaches, we will now elaborate on the Goemans-Williamson algorithm, which currently stands as the most effective approximation method for MaxCut, boasting the best-known performance guarantee to date (0.87856). Its detailed explanation is particularly relevant as we plan to utilize it extensively in benchmarking our proposed algorithms later on. We opt for GW over other options for one obvious reason: Our aim is to push the boundaries of MaxCut algorithms, thus it is logical to compare against the top-performing method. (We do not use any of the heuristic/metaheuristic algorithms, as they do not provide performance guarantees, and we are interested in having a benchmark to compare against.)

2.1.1.1 Goemans-Williamson Algorithm

In this subsubsection, we shall present the Goemans-Williamson Algorithm in greater detail. This approximation algorithm, developed by Michel X. Goemans and David P. Williamson in 1995 [9], is a cornerstone in the field of combinatorial optimization, particularly for the MaxCut problem. The algorithm leverages semidefinite programming to construct a solution that is then rounded to provide a near-optimal cut. Semidefinite programming involves optimizing a linear function of a symmetric matrix while adhering to linear equality constraints and the requirement that the matrix be positive semidefinite. This programming paradigm finds utility across diverse fields such as control theory, nonlinear programming,

geometry, and combinatorial optimization, showcasing its versatility and applicability. (These applications are detailed in [9] and references therein.)

Formally, the Goemans-Williamson Algorithm can be described as follows. (We will use the original formulation, from [9], presenting a distilled version of the most important aspects of their description.) Given a graph, with vertex set $V = \{1, ..., n\}$ and non-negative weights $w_{ij} = w_{ji}$ for each pair of vertices i and j, the weight of the maximum cut $w(S_1, S_2)$ is given by the following integer quadratic program[3]:

$$\begin{aligned} & \text{Maximize } \frac{1}{2}\sum_{i<j} w_{ij}(1 - y_i y_j) \\ (Q) \qquad & \text{Subject to: } y_i \in \{-1, 1\} \qquad \forall i \in V. \end{aligned} \tag{2.2}$$

It is clear that the sets $S_1 = \{i | y_i = 1\}$ and $S_2 = \{i | y_i = -1\}$ correspond to a cut of weight $w(S_1, S_2) = \frac{1}{2}\sum_{i<j} w_{ij}\,(1 - y_i y_j)$. This represents the same quantity as the aforementioned Cut$(S_1, S_2) = \sum_{(u,v)\in E} \chi(u, v)$ (Eq. 2.1), in the un-weighted case. Explicitly, in words, the sum in Eq. 2.2 is to be taken over all the graph's edges (i, j), with y_i representing node-i's associated set: $y_i = +1$, if node-$i \in S_1$, or $y_i = -1$, when node-$i \in S_2, \forall i \in V$.

Given that solving this integer quadratic program is NP-complete [9], we explore relaxations of (Q). These are obtained by relaxing some of the constraints of (Q), and extending the objective function to the larger space. Consequently, all solutions of (Q) are feasible for the relaxation, and the optimal value of the relaxation serves as an upper bound for (Q)'s optimal value. We interpret (Q) as constraining y_i to be a 1-dimensional unit vector. A number of possible relaxations involve allowing y_i to be a multidimensional vector v_i of unit Euclidean norm. As the linear space spaned by the vectors v_i has dimension at most n (number of graph nodes), we can assume that these vectors belong to $\mathbb{R}^n$. More precisely, they belong to the n-dimensional unit sphere, S_n. At this point, so as to make sure that the resulting optimization problem is, indeed, a relaxation, we must define the cost function such that it reduces to $\frac{1}{2}\sum_{i<j} w_{ij}(1 - y_i y_j)$ for vectors lying in a 1-dimensional space. One way to do this is to replace $y_i y_j$ with the inner product $v_i \cdot v_j$ in the expression for $w(S_1, S_2)$. The resulting relaxation, (P), reads:

$$\begin{aligned} & \text{Maximize } \frac{1}{2}\sum_{i<j} w_{ij}(1 - v_i \cdot v_j) \\ (P) \qquad & \text{Subject to: } v_i \in S_n \qquad \forall i \in V. \end{aligned} \tag{2.3}$$

In the context of this semidefinite programming relaxation, the GW algorithm follows these steps:

1. Solve (P), thus obtaining an optimal set of vectors v_i;
2. Let r be a vector uniformly distributed on the unit sphere, S_n;
3. Set $S_1 = \{i | v_i \cdot r \geq 0\}$ and $S_2 = \{i | v_i \cdot r < 0\}$.

[3]Note that we use the general formulation here, contemplating the possibility of weighted graphs, i.e., we do not restrict ourselves to un-weighted graphs.

In other words, we choose a random hyperplane in n dimensions, crossing through the origin, and partition the vertices according to whether the vectors v_i lie "above" ($v_i \cdot r \geq 0$) or "below" ($v_i \cdot r < 0$) this hyperplane. Those above the hyperplane are assigned to S_1, and those below to S_2.

Furthermore, it can be shown that the GW algorithm has a performance guarantee of:

$$\alpha = \min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} > 0.878. \tag{2.4}$$

A detailed proof elucidating the appearance of this 0.878 value is present in section 3 of [9].

Computationally, one defines the positive semidefinite matrix $X \in \mathbb{R}^{n \times n}$, with $X_{ii} = 1$ and $X_{ij} = v_i \cdot v_j = v_i^T v_j, \forall i, j \in V$. Then, we can solve the following semidefinite program:

$$\begin{aligned} &\text{Maximize } \frac{1}{2} \sum_{\text{Edges}} w_{ij}(1 - X_{ij}) \\ (N) \qquad &\text{Subject to: } X_{ii} = 1, X \succeq 0. \end{aligned} \tag{2.5}$$

We do possess effective tools and techniques for solving semidefinite programs, enabling the efficient determination of the optimal X matrix. From there, we can extract the vectors v_i, by performing the square root operation on the matrix X. The resulting matrix's columns, or lines – X is symmetric – will correspond to the desired vectors v_i. This outlines the process for solving (P) as listed earlier.

2.1.2 Hybrid Quantum-Classical State-of-the-Art Algorithms for MaxCut

A number of different hybrid quantum-classical algorithms have been proposed to solve the MaxCut problem, leveraging the unique properties of quantum computing to potentially outperform purely classical algorithms. They are designed to exploit quantum superposition and entanglement to explore the solution space more efficiently. In this section, we will briefly mention the state-of-the-art hybrid quantum-classical algorithms for MaxCut, focusing on the Quantum Approximate Optimization Algorithm (QAOA) and the Qubit-Efficient MaxCut Heuristic Algorithm (QEMC). Due to their variational nature, these algorithms do not have a theoretical performance guarantee, but they have shown promising results in practice.

Presently, the Quantum Approximate Optimization Algorithm (QAOA), initially introduced in [2], is one of the top candidates for achieving meaningful quantum advantage with near-term quantum devices. Although, MaxCut-wise, it has a substantial limitation: it requires a number of qubits equal to the number of graph nodes. This is a significant drawback, as it severely limits the algorithm's scalability to bigger graphs. Hence, why it is believed that "QAOA for Max-Cut requires hundreds of qubits for quantum speed-up" [14]. After all, our classical algorithms work just fine for small graphs, which is why we do not present performance metrics for basic QAOA. Alternatively, different QAOA variations have been proposed to address the issue of high qubit requirements, such as QAOA-in-QAOA [15], which partitions a large graph into many smaller subgraphs, each of which is easily solved using a separate QAOA instance. Afterwards, these results are joined together by working through another, this time **weighted**, MaxCut problem. This approach in particular has proven to yield competitive or even better performance

over the best known classical algorithms, i.e. GW, for graphs up to 2000 nodes.

A separate problem one might face when implementing QAOA, that also hinders its application to larger graphs, has to do with qubit connectivity[4], as QAOA might require specific connectivity patterns that are not natively available in present-day NISQ devices. To implement these, we are often required to utilize a number of 2-qubit SWAP gates, which degrade the algorithm's performance by introducing extra noise in the system, and hence errors in our results. Therefore, a parallel line of work has been to develop QAOA variations that have lesser connectivity requirements, in the sense that they do not depend on arbitrary qubit connectivity. For example, Parity-QAOA [16, 17] was designed for quantum chips with planar lattice geometry (Figure 2.2), requiring only nearest-neighbour connectivity. Although it necessitates more qubits, it entirely solves this problem.

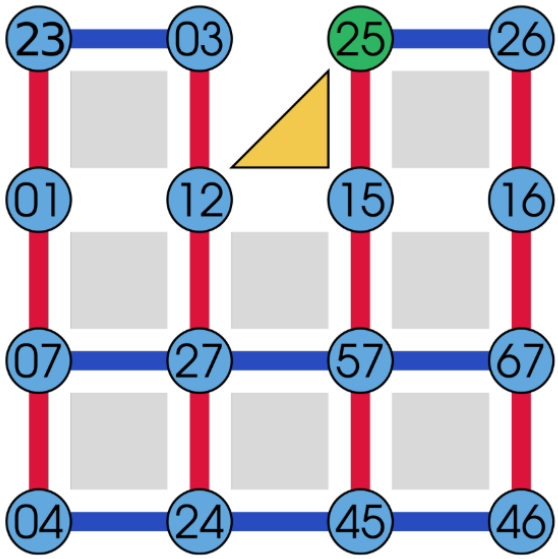


Figure 2.2: Planar lattice geometry illustration: the circular nodes represent the physical qubits, each of which can interact directly with its nearest neighbours. The lines represent terms of the driver Hamiltonian, from Parity-QAOA [16]. The numbers refer to edges in the original graph. Image reproduced from [16].

Furthermore, the Qubit-Efficient MaxCut Heuristic Algorithm (QEMC), introduced in [3], is a quantum-classical hybrid algorithm that aims to solve the MaxCut problem using fewer qubits than QAOA, therefore extending its applicability to larger graphs. It has shown cutting-edge performance in practice, surpassing the classical state-of-the-art, i.e. GW, for graphs up to 2048 nodes. The novelty of this algorithm is that it opens up the door for quantum devices to be used for large MaxCut instances, previously unfeaseble due to qubit number limitations, which rendered QAOA computations for such large graphs impossible. As was already mentioned, QEMC allows for an exponential compression in the number of needed qubits, enabling its utilization on today's quantum hardware for considerably larger graphs than what would be possible with traditional QAOA. However, this exponential compression also makes it efficiently simulable classically, which effectively defeats its purpose as a quantum algorithm. For this

[4]Qubit connectivity refers to the pattern or arrangement of connections between qubits in a quantum computing system, determining which qubits can interact directly with each other.

reason, QEMC is termed a quantum-inspired classical algorithm and, by definition, will not produce any quantum advantage. Either way, we include it in this subsection, since it can always be implemented as a VQA on current-day NISQ devices.

More recently, different algorithms have been proposed in an attempt to reduce the number of qubits required for larger graphs, without falling into the trap of efficient classical simulability. (After all, we are looking for quantum advantage.) Polynomial compression-based algorithms, such as that proposed in [18], have too shown promising results. This one in specific [18] is, to the best of our knowledge, boasting the highest quality attained experimentally on sizes up to 7000 graph nodes, competitive with state-of-the-art classical solvers. Not only that, but their specific qubit-efficient encoding brings in a super-polynomial mitigation of barren plateaus[5] as a built-in feature, which constitutes a significant advantage over other VQAs. Curiously enough, our initial idea for the iQAQE algorithm is quite similar to what is explored in [18]. However, we believe our algorithm to be somewhat more general, by not assuming *a priori* a polynomial compression in the number of qubits.

2.2 Quantum Computing Primer

In this section, we present a brief introduction to quantum computing, focusing on its fundamental ideas and principles. We begin with an overview of the general concepts, followed by a discussion of quantum gates and circuits, crucial for understanding quantum algorithms. This groundwork will prepare us to delve into the specifics of variational quantum algorithms and their application to the MaxCut problem.

General concepts and mathematical formalism

(This description is inspired by [19].)

Quantum computing makes use of the foundational principles of quantum mechanics in an attempt to extract some sort of quantum advantage from them. Instead of using classical binary digits, quantum computers use their quantum analog, qubits. In practice, qubits can be any two-state quantum system. Said states are, unequivocally, associated with the states $|0\rangle$ and $|1\rangle$, conventionally, very much like in classical computing when one uses bits. The primary distinction lies in the fact that a qubit, being a quantum system, can exist in a superposition of both states generally denoted as:

$$\mathcal{H}^2 \ni |\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.6}$$

where $\alpha, \beta \in \mathbb{C}$, with $|\alpha|^2 + |\beta|^2 = 1$. This ensures probability normalization to 1, as, according to the famous Born rule, the probability of measuring the qubit in state $|0\rangle$ is $|\langle 0|\psi\rangle|^2 = |\alpha|^2$, and in state $|1\rangle$ is $|\langle 1|\psi\rangle|^2 = |\beta|^2$.

In the same vein, one can compose (represented by the tensor product) n qubits to obtain an n-qubit

[5] In VQA analysis, barren plateaus are characterized by a vanishing expectation value of $\nabla\mathcal{L}$ over random parameter initializations and an exponential decay (in n) of its variance. Note that $\mathcal{L}$ refers to the loss function. These are, thus, flat regions of the parameter space, impeding optimization.

state:

$$|\psi\rangle = \bigotimes_{i=0}^{n-1}(\alpha_i |0\rangle \ + \beta_i |1\rangle). \tag{2.7}$$

More generally, we can express an arbitrary pure state $|\psi\rangle$ by a normalized linear combination of the computational basis states $|b_0 b_1 ... b_{n-1} b_n\rangle \coloneqq |b_0\rangle \otimes |b_1\rangle \otimes ... \otimes |b_{n-1}\rangle \otimes |b_n\rangle$:

$$|\psi\rangle = \sum_{\boldsymbol{b}\in\{0,1\}^{\otimes n}} \alpha_{\boldsymbol{b}} |b_0 b_1 ... b_{n-1} b_n\rangle , \tag{2.8}$$

where $\sum_{\boldsymbol{b}\in\{0,1\}^{\otimes n}} |\alpha_{\boldsymbol{b}}|^2 = 1$. This superposition phenomenon (mathematically characterized by the linear combination of basis states), utterly impossible in classical physics, brings unprecedented computational power under certain scenarios, as it introduces a kind of built-in parallelism in quantum computing, a highly desirable feature for any computational scheme.

In addition to this, quantum computing also exploits entanglement as a resource. Entanglement allows for intricate correlations between qubits, which have no classical counterpart and are believed to offer computational speed-ups, although it is yet unclear exactly how. An entangled state, by definition, is one that cannot be expressed as the tensor product of individual qubit states, i.e., it cannot be represented in the form of Eq. 2.7. The most famous example of entangled states are the Bell states, representing what are known as two-qubit (or bipartite) maximally entangled states. Such Bell states are reproduced below:

$$\begin{aligned} \left|\Phi^+\right\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad \left|\Phi^-\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \\ \left|\Psi^+\right\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), \quad \left|\Psi^-\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle). \end{aligned} \tag{2.9}$$

Clearly, these states cannot be expressed as a tensor product of two individual qubit states. From a practical standpoint, Bell states frequently serve as foundational elements in advanced techniques like quantum teleportation and quantum cryptography. Quantum Key Distribution (QKD) protocols, for example, frequently rely on entangled Bell pairs to distribute secure keys for encrypting and decrypting messages.

Another key topic in quantum information theory is the distinction between pure and mixed states. So far, we have only been treating pure states, which are represented by a single ket vector in some Hilbert space. Mixed states, on the other hand, are represented by density matrices, which are positive semidefinite matrices with unit trace. These matrices are used to describe a statistical ensemble of quantum states. The density matrix ρ of a quantum state $|\psi\rangle$ is defined as:

$$\rho = |\psi\rangle\langle\psi| \tag{2.10}$$

In the more general case of a statistical ensemble of states $|\psi_i\rangle$, each with probability p_i, the density matrix ρ is given by:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| . \tag{2.11}$$

It's crucial to avoid confusing this with superposition, as a mixed state comprises a statistical ensemble of pure states, while superposition pertains to a single state that is a linear combination of other states. For instance, the state $|\psi_S\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ represents a superposition of $|0\rangle$ and $|1\rangle$, whereas the state $\rho_M = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$ denotes a mixed state, signifying that half the states in the ensemble are in state $|0\rangle$ and the other half in state $|1\rangle$. Unlike the probabilistic mixture (ρ_M), the superposition (ψ_S) can exhibit quantum interference, hence why it is important to distinguish between the two. If desired, this distinction can also be observed in the off-diagonal terms of the density matrix, referred to as coherences, which are responsible for quantum interference effects: the off-diagonal terms of ρ_M are zero, whereas those of $\rho_S = |\psi_S\rangle\langle\psi_S| = \frac{1}{4}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| + |1\rangle\langle 1|)$ are not.

Currently, numerous quantum computing architectures are in competition and the ultimate victor remains uncertain. Nevertheless, the most prominent architectures have qubits composed of either photons [20, 21], neutral atoms (using Rydberg states) [22, 23], trapped ions [24], semiconductors (still in their infancy), or, most notably, superconducting circuits [25, 26] – employed by giants like IBM and Google in their quantum processors. Looping back to the definition of a qubit, considering a photonic quantum computer, it becomes rather straightforward and natural to map the photons' vertical and horizontal polarizations to the $|0\rangle$ and $|1\rangle$ states, respectively. Once again, this mapping is analogous to how classical computing maps the current/no current states to 1 and 0.

The next step is to understand how these so-called quantum circuits act on our qubits, and how they can be used to perform quantum computations. This is where quantum gates come into play.

Quantum gates and circuits

Presently, quantum computers are being designed predominantly with a circuit-based architecture. These circuits serve as the quantum equivalent of classical logic circuits, however, instead of the typical AND, OR, and NOT gates, quantum circuits feature quantum gates. These gates apply specific transformations to qubits, based on their definitions, taking an initial state $|\psi\rangle$ to an output state $|\phi\rangle = U|\psi\rangle$, for some gate U, **designed to be unitary**, i.e., $U^\dagger U = UU^\dagger = I$, where I is the identity matrix. Unitary transformations preserve the normalization of quantum states and the inner product between states, which are essential properties in quantum mechanics. Additionally, they ensure that quantum operations are reversible, meaning that information is not lost during computation. This [unitary gates] can also be seen as a natural consequence of the Schrödinger equation, governing the time evolution of quantum systems, requiring said evolution to be described by a unitary operator.

The most frequently used quantum gates consist of rotations around each of the three Cartesian axes, represented as complex exponentials of the Pauli x, y and z matrices (σ_x, σ_y and σ_z). The Pauli matrices are defined as: (Note the alternative notation, $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$, for the sake of clarity.)

$$\boldsymbol{X} \coloneqq \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{Y} \coloneqq \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \boldsymbol{Z} \coloneqq \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.12}$$

Furthermore, when we mention rotations, these are to be seen in the Bloch sphere, which is a geomet-

rical representation of the pure state space of a two-level quantum system, an illustration of which can be seen in Figure 2.3.

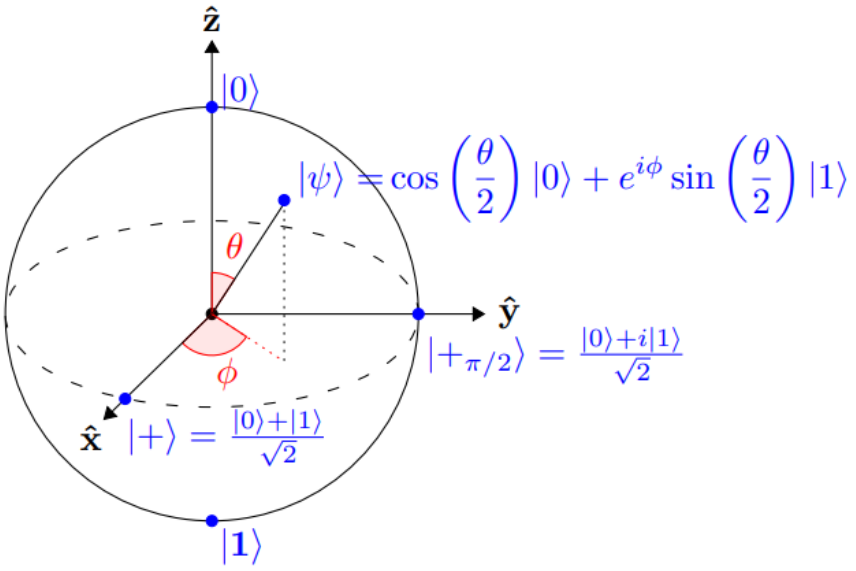


Figure 2.3: Bloch sphere representation. The sphere's north and south poles correspond to states $|0\rangle$ and $|1\rangle$, respectively. Reproduced from [27].

The aforementioned rotation gates can be written as: $R_x(\theta) = e^{-i\theta\sigma_x/2}$, $R_y(\theta) = e^{-i\theta\sigma_y/2}$, and $R_z(\theta) = e^{-i\theta\sigma_z/2}$. In matrix form, this reads:

$$R_x(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \quad R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \quad R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}, \tag{2.13}$$

These will frequently appear in quantum circuits, as they are the most basic quantum gates, in the sense that all other single-qubit gates can be re-constructed from them. I.e., for any unitary single-qubit gate U, one can always find a decomposition $U = e^{i\phi}R_Z(\gamma)R_X(\beta)R_Z(\alpha)$ (proven in [28]).

For illustrative purposes, to elucidate how single-qubit gates act on qubits, it is quite simple to verify that the σ_x matrix (alternative symbol, $\boldsymbol{X}$) behaves exactly like a NOT gate. This correspondence is established through a $180°$ rotation around the $x-$axis. Mathematically,

$$\boldsymbol{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{2.14}$$

such that:

$$\boldsymbol{X}|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle\,. \tag{2.15}$$

Another essential gate found in most quantum algorithms is the Hadamard gate. It can be represented as the following linear combination of $\boldsymbol{X}$ and $\boldsymbol{Z}$ matrices: $H = \frac{1}{\sqrt{2}}(\boldsymbol{X} + \boldsymbol{Z})$, and is particularly useful for

creating superpositions, as it maps the state $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \coloneqq |+\rangle$, and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \coloneqq |-\rangle$.

In addition to these simple one-qubit quantum gates, there are also gates designed for multiple qubits. For instance, consider the controlled-NOT gate (CNOT), which does exactly what its name suggests: if the control qubit is in state $|1\rangle$, it applies NOT to the test qubit, whereas if it is in state $|0\rangle$ instead, nothing happens. The CNOT gate is also quite prevalent in quantum circuits, playing a vital role in creating entanglement between qubits, which is a crucial resource in quantum computing. The circuit representation and equivalent matrix form of the CNOT gate are presented below, in Figure 2.6.

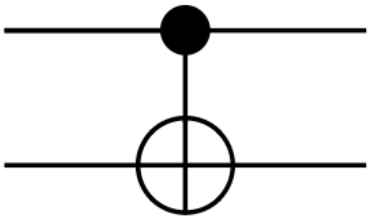


Figure 2.4: CNOT gate – Circuit representation. The black circle identifies the control qubit. The other qubit is the test qubit.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2.5: CNOT gate – Equivalent matrix form.

Figure 2.6: Two-qubit gate example – The controlled-NOT gate.

More generally, within a quantum circuit, an intricate network of interconnected qubits and gates collaborates to execute a specific algorithm. For example, the circuit depicted in Figure 2.7, below, is designed for the implementation of the renowned Grover's search algorithm [29].

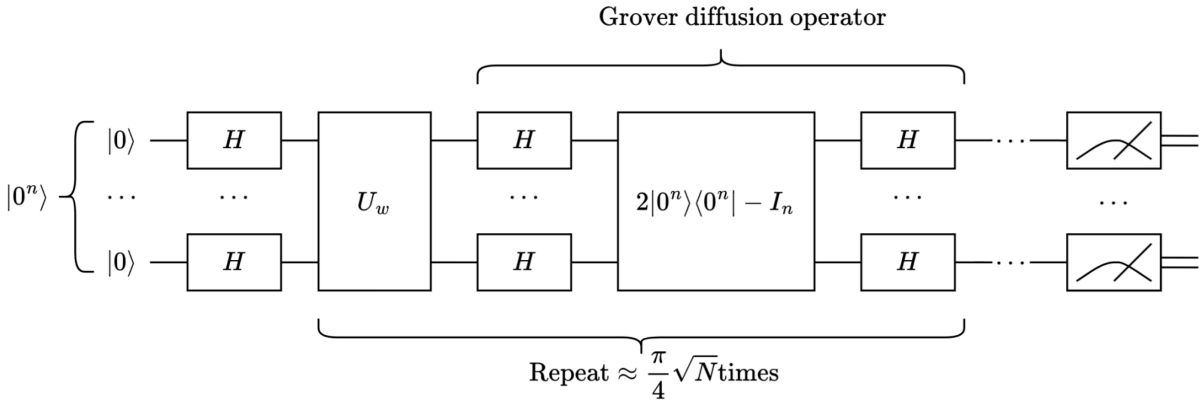


Figure 2.7: Quantum circuit utilized in the famous Grover's search algorithm [29]. Integraly reproduced from [30].

This is a purely quantum algorithm, different from what we intend to explore in this work. Nevertheless, it serves as a good example of how quantum circuits can be used to implement complex quantum algorithms.

An additional point of interest is the potential for classical simulation of quantum circuits. Quantum gates, being akin to mathematical operators, can always be expressed in matrix form. Consequently, analyzing the impact of a series of quantum gates, i.e., a quantum circuit, on a specific quantum state is achievable by sequentially applying the relevant operators, as they appear in the circuit. At the end of this

process, we arrive at a quantum state typically distinct from the initial state. This entire simulation can be executed classically. However, its scalability is severely limited, as the number of states, constituting our computational basis, representable with n qubits grows exponentially, at a rate of 2^n. As one might anticipate, this very quickly becomes unmanageable. Eventually, either memory constraints impede the storage of such extensive states, or the computations slow down to a point where the endeavor loses practicality. Effectively, simulating quantum computers becomes challenging beyond a few dozen qubits. Some tabletop calculations suggest that simulating 28 qubits would necessitate approximately 13 GB of memory, reaching the theoretical maximum capacity of a modern standard 16 GB personal computer. It's important to note that for each additional qubit, memory requirements double. For instance, simulating 50 qubits would demand about 54 Petabytes of memory, and for 51 qubits, approximately 108 PB, and so forth. These estimations are based on the assumption that a complex number is stored as two Python floats, each requiring 24 bytes of memory, thus a complex amplitude requires 48 bytes of memory to be stored. Then, one such amplitude is required for each of the 2^n basis states, for n qubits.

2.3 Hybrid Quantum-Classical Computing

Hybrid quantum-classical computing refers to a computational approach that combines elements of both classical and quantum computing paradigms to leverage the strengths of each. In this model, classical processors and quantum processors work in tandem to solve complex problems more efficiently than either could achieve alone. This collaborative strategy aims to harness quantum computing's unique capabilities while mitigating the challenges and limitations associated with quantum systems, such as error correction and decoherence. As of today, the synergy between classical and quantum elements holds promise for addressing complex real-world problems in areas like optimization, machine learning, and cryptography (cf. Figure 2.8).

2.3.1 Variational Quantum Algorithms

The hallmark of HQCC is what are called variational quantum algorithms (VQAs), which correspond to hybrid quantum-classical algorithms, typically realized through a parameterized quantum circuit (PQC). Additionally, as part of their implementation, the parameters are subjected to training in order to achieve the desired outcomes, by minimizing the value of a cost function. As such, VQAs can be thought of as the quantum analogue of highly successful machine-learning methods, such as neural networks. Moreover, since they outsource the circuit parameters' optimization to a classical optimizer, exterior to the quantum processor, VQAs leverage the full toolbox of classical optimization. As a whole, this approach has the added advantage of keeping the quantum circuit depth shallow, which, ultimately, helps in mitigating the overall noise level. This is crucial, since it allows for these algorithms to be implemented in the current NISQ (Noisy Intermediate Scale Quantum) era, not requiring complete fault-tolerance to produce reasonable results. A wide array of possible applications has been examined for VQAs, essentially covering all the use cases envisioned by researchers for quantum computers (See Figure 2.8).

Despite all this, it is important to note that VQAs are not without fault. There is still a lot of work to be done regarding their trainability, accuracy and efficiency. Either way, they are undoubtedly the most promising candidates for achieving useful quantum advantage in the near future, which is exactly why they have come under the spotlight, drawing the attention of numerous scientists over the past few years.

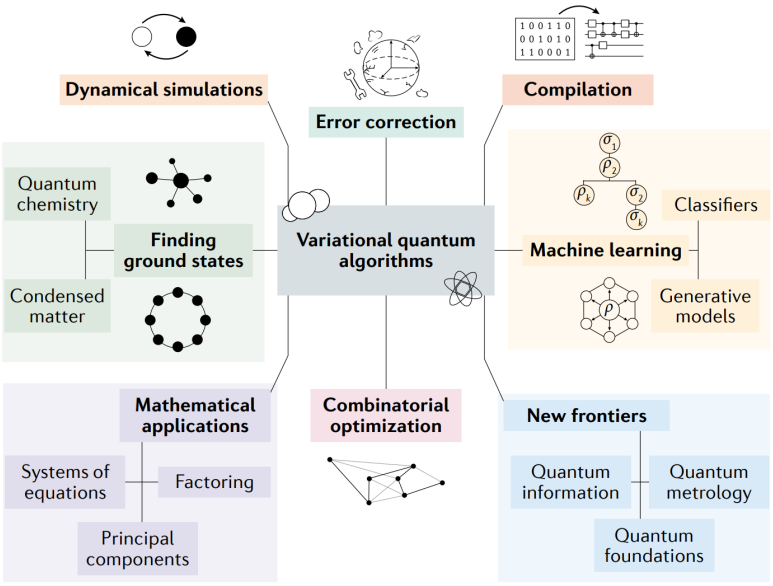


Figure 2.8: Applications of variational quantum algorithms. Sourced from [31] in its entirety.

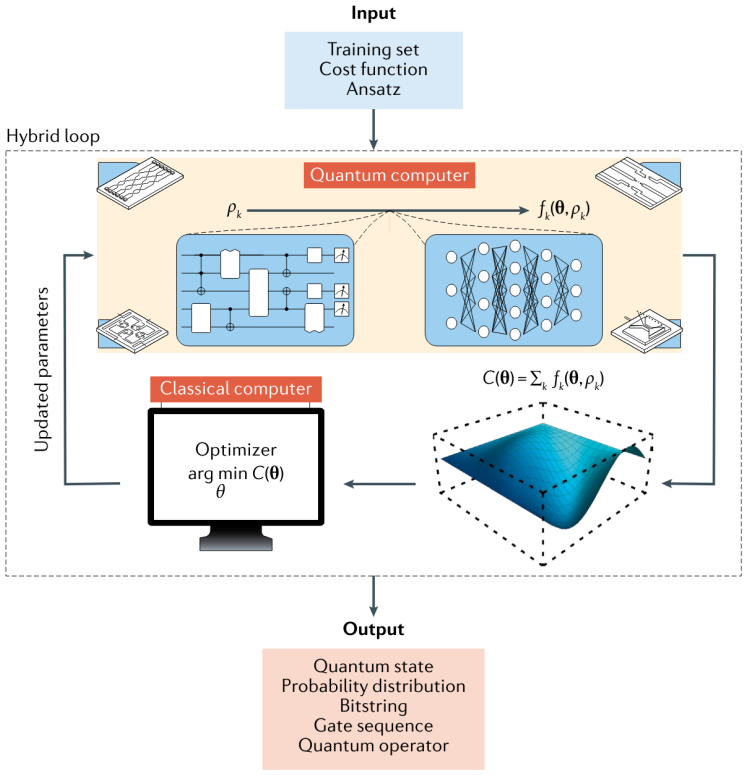


Figure 2.9: Schematic diagram of a variational quantum algorithm. Adapted from [31].

2.3.1.1 Basic structure of a VQA

(This description is heavily inspired by [31].)

This section provides an overview of the essential components of VQAs. Kicking off the development of a VQA involves defining a cost (or loss) function, C, that encodes the solution to the problem. Following this, an ansatz is suggested, that is a quantum operation that relies on a set of continuous or discrete parameters, $\boldsymbol{\theta}$, subject to optimization. In other words, we propose the functional form/structure of the parameterized quantum circuit. Afterwards, the suggested ansatz undergoes training, within a hybrid quantum-classical loop (cf. Figure 2.9), to address the following optimization task

$$\boldsymbol{\theta}^{\star} = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}), \tag{2.16}$$

corresponding to the minimization of the selected cost function. The distinctive feature of VQAs is their application of a quantum computer to estimate the cost function, $C(\boldsymbol{\theta})$, and its gradient, while relying on classical routines for the optimization of the parameters' values, $\boldsymbol{\theta}$. Next, we provide supplementary information for each step of the VQA framework.

Cost function

An essential aspect of VQA involves encoding the problem into an adequate cost function. Similar to classical machine learning, this function associates values of the trainable parameters, $\boldsymbol{\theta}$, with real numbers. At a more abstract level, the cost function defines a hypersurface, often termed the cost landscape (cf. Figure 2.9), in a way such that the optimizer's task is to then navigate across this landscape and discover its global minimum, as this would correspond to the sought-after solution. In many instances, it is beneficial and feasible to express the cost in the form:

$$C(\boldsymbol{\theta}) = \sum_{k} f_k \left(Tr \left[O_k U(\boldsymbol{\theta}) \rho_k U^{\dagger}(\boldsymbol{\theta}) \right] \right), \tag{2.17}$$

for some set of functions $\{f_k\}$, where $U(\boldsymbol{\theta})$ is a parameterized unitary, $\boldsymbol{\theta}$ is composed of discrete and continuous parameters, ρ_k are input states from a training set and O_k are a set of observables. Note how the argument of f_k, $Tr\left[O_k U(\boldsymbol{\theta}) \rho_k U^{\dagger}(\boldsymbol{\theta})\right]$, corresponds to the expectation value of observable O_k, in state $\rho_k' = U(\boldsymbol{\theta}) \rho_k U^{\dagger}(\boldsymbol{\theta})$. Oftentimes, it is convenient to design the cost function to be given by the expectation value of a Hamiltonian (e.g., the problem Hamiltonian in QAOA), which is why we mention that it is beneficial for it to have this specific form. Then, the minimization of the cost corresponds, unambiguously, to the determination of the ground-state energy. Additionally, one frequently uses Pauli operators, whose expectation values are rather easy to compute in practice.

Furthermore, when designing a cost function, we want it to be as faithful as possible, in the sense that its minimum should closely correspond to the desired solution. Likewise, it is imperative that we can efficiently estimate $C(\boldsymbol{\theta})$, as this is integral to the functionality of this method. So far, we have been taking this for granted. However, one must guarantee that this is indeed verified experimentally, otherwise we are not able to proceed.

Ansatzë

Another crucial element of a VQA is the ansatz. In broad terms, the ansatz's structure determines the nature of the parameters $\boldsymbol{\theta}$ and, consequently, how they can be optimized to minimize the cost. The design of an ansatz often relies on the specific requirements of the task, allowing for the creation of problem-inspired ansätze. However, certain ansatz architectures are generic and problem-agnostic, making them applicable even in scenarios where relevant information is lacking. As one might expect, problem-agnostic ansätze, being more general, will also, usually, require a greater number of parameters, which unsurprisingly hinders the optimization task. However, this also grants them more flexibility. On the other hand, problem-oriented ansätze are capable of assimilating problem-specific information into the structure of quantum circuits. This custom-fit approach can result in a reduction of the parameter space, making optimization more efficient, and lead to solutions that are more meaningful and interpretable. Furthermore, adopting such an approach might (at times, not always!) enable the use of shallower quantum circuits, thereby aiding in the reduction of overall noise levels. (Greater circuit depths require more quantum gates, which results in more noise.) When it all comes down to it, one should weigh the trade-off between accuracy and generality, and carefully assess resource utilization in the choice of ansatz. (Yet, for achieving optimal results tailored to a specific problem, it's always recommended to utilize problem-inspired ansätze. The challenge lies in their design. While it's tempting to opt for a problem-agnostic ansatz and consider the task done, problem-inspired designs consistently yield superior performance.) For illustration purposes, here are a few examples of commonly considered ansätze types: hardware-efficient ansätze (aimed at reducing circuit depth), unitary coupled clustered ansätze (for quantum chemistry simulations), quantum alternating operator ansätze (used in QAOA), *et cetera*.

Gradients

After specifying the cost function and ansatz, the next step involves training the parameters, $\boldsymbol{\theta}$, and tackling the optimization problem described in Eq. (2.16). It is known that, for numerous optimization tasks, leveraging information from the gradient or higher-order derivatives of the cost function can enhance the speed and ensure the convergence of the optimizer. A notable benefit of various VQAs is the ability to analytically compute the gradient of the cost function. This can be done using the parameter-shift rule. If the cost function has the form of Eq. (2.17), with $f_k(x) = x$, and θ_l is the l^{th} element in $\boldsymbol{\theta}$, which parameterizes a unitary $e^{i\theta_l \sigma_l}$, with σ_l a Pauli operator, then the parameter-shift rule states that the equality

$$\frac{\partial C}{\partial \theta_l} = \sum_k \frac{1}{2\sin\alpha} \left(Tr\left[O_k U(\boldsymbol{\theta}_+) \rho_k U^\dagger(\boldsymbol{\theta}_+) \right] - Tr\left[O_k U(\boldsymbol{\theta}_-) \rho_k U^\dagger(\boldsymbol{\theta}_-) \right] \right), \tag{2.18}$$

with $\boldsymbol{\theta}_\pm = \boldsymbol{\theta} \pm \alpha \boldsymbol{e_l}$, holds for any real number α. In practice, one uses $\alpha = \pi/4$, since this maximizes the accuracy of the result [31]. Note that, although the expression is analytically exact, we can only ever obtain approximate results (hence the discussion about accuracy), since we are required to compute

expectation values of O_k, necessitating the quantum circuit to be sampled. This section was simply meant to shed some light on how one can experimentally compute these gradients, required for the optimization procedure.

Workflow

The VQA workflow is incredibly similar to a traditional machine learning pipeline. We have some input quantum state entering our ansatz, which is then processed by the quantum circuit, parameterized by $\boldsymbol{\theta}$. Afterwards, the output of this circuit is measured and the results are used to compute the cost function, $C(\boldsymbol{\theta})$. Following this, said cost is fed into a classical optimizer, which updates the parameters, $\boldsymbol{\theta}$, in order to minimize the cost[6]. This process [run quantum circuit, estimate cost, update parameters] is repeated iteratively until the optimizer converges to a minimum, at which point the optimal parameters, $\boldsymbol{\theta}^\star$, are obtained. These can then be used to prepare the quantum state that solves the problem at hand, in the sense that the problem's solution can be extracted from it. This is a very high-level overview of the VQA workflow, but it should give a good idea of how the previously mentioned concepts come together to form a working algorithm.

Applications & Examples

In what follows, I will merely outline some of the most promising applications of VQAs for solving real world problems. These are (cf. Figure 2.8): (Specific algorithms for these tasks are indicated in parenthesis.)

1. **Finding ground states and excited states** (Variational Quantum Eigensolver [32], VQE, and variations thereof);
2. **Dynamical quantum simulations** - Based on iterative variational algorithms, they allow, e.g., for the simulation of open quantum systems [31];
3. **Optimization tasks** (Quantum Approximate Optimization Algorithm [2], QAOA, and Qubit Efficient MaxCut Heuristic Algorithm [3], QEMC, both of which we shall analyse in further detail later);
4. **Mathematical applications** [31] - Linear systems, matrix-vector multiplication, non-linear equations, factoring, *et cetera*;
5. **Error correction** (Variational Quantum Error Corrector [33], QVECTOR);
6. **Machine learning and data science** - Quantum Machine Learning (QML) [34], classifiers, generative models, *et cetera*.

We are, now, finally ready to tackle the two specific variational quantum algorithms, QAOA and QEMC, which constitute the main focus of this work. Let us start with QAOA.

[6] It's a bit more complex, as we are required to compute the gradients for the classical optimization. For this, we simply use the parameter-shift rule. This requires some more shots, with different parameter values, but is entirely feasible.

2.3.1.2 Quantum Approximate Optimization Algorithm (QAOA) Review

Renowned as the most prominent VQA in quantum-enhanced optimization, the QAOA [2] was originally devised to approximate solutions for combinatorial optimization problems, including constraint-satisfaction, SAT [35], and Max-Cut problems [36]. In this section, we intend to explain in greater detail the inner workings of QAOA, since it is closely related to the new hybrid algorithm proposed by the HQCC project collaboration[7]. In a similar vein, the next section will delve into an equivalent level of detail, this time concentrating on the Qubit Efficient MaxCut Heuristic Algorithm, QEMC.

Combinatorial optimization problems are formulated on binary strings, $s = (s_1, ..., s_N)$, with the goal of minimizing/maximizing a designated classical objective function, $L(s)$. In QAOA, however, the cost function is defined as the expectation value of a quantum Hamiltonian, H_C (or H_P), termed the cost (or problem) Hamiltonian, and doesn't explictly depend on such bit-string s. This [cost] is constructed by mapping each classical variable, $s_j \in \{1, -1\}$, to a Pauli spin-$1/2$ operator, $\boldsymbol{Z}_j$. Thus, the usual MaxCut objective $L(s) = \frac{1}{2}\sum_{\text{Edge (j,k)}}(1 - s_i s_j)$, denoting the cut of partition $s = (s_1, ..., s_N)$, is transformed into the cost Hamiltonian

$$H_C = \frac{1}{2}\sum_{\text{Edge (j,k)}}(1 - \boldsymbol{Z}_i\boldsymbol{Z}_j), \tag{2.19}$$

from which one can extract the QAOA objective function.

Next up is the QAOA ansatz. Drawing inspiration from the quantum adiabatic algorithm, QAOA substitutes adiabatic evolution with p cycles of alternating time evolution between the cost Hamiltonian, H_C, and a suitably chosen mixer (or bias) Hamiltonian, H_M (or H_B). The role of this mixer Hamiltonian can be understood as introducing quantum fluctuations, or transitions, between different states, helping the algorithm explore the solution space more effectively, ideally preventing it from getting trapped in sub-optimal local minima/maxima. The entirety of this process forms the previously mentioned quantum alternating operator ansatz.

In practice, defining $\boldsymbol{\theta} = \{\boldsymbol{\gamma}, \boldsymbol{\alpha}\}$, the cost function is $C(\boldsymbol{\gamma}, \boldsymbol{\alpha}) = \langle\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})| H_C |\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})\rangle$, with

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})\rangle = e^{-i\alpha_p H_M} e^{-i\gamma_p H_C} ... e^{-i\alpha_1 H_M} e^{-i\gamma_1 H_C} |\psi_0\rangle\,, \tag{2.20}$$

where $|\psi_0\rangle$ is the initial state entering the ansatz. This ansatz is illustrated in Figure 2.10, below, for $n \in \mathbb{N}$ QAOA layers.

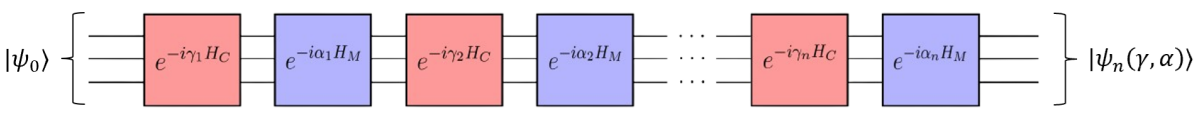


Figure 2.10: Quantum alternating operator ansatz. Adapted from [37].

Replacing adiabatic time evolution with a *Trotterized* evolution works due to the famous Trotter-Suzuki

[7] More information about this collaboration is present in the Acknowledgments section.

formulas [19], which state that (for general non-commuting H_j):

$$e^{-i\sum_{j=1}^{m} H_j t} = \prod_{j=1}^{m} e^{-iH_j t} + O(m^2 t^2). \tag{2.21}$$

If $t \ll 1$, then the error in this approximation becomes negligible. On the other hand, if t is large, Trotter-Suzuki formulas can still be used to simulate the dynamics accurately by breaking things up into a sequence of short time-steps. Let r be the number of steps taken in the time evolution, so each time-step runs for time t/r. Then, we have that

$$e^{-i\sum_{j=1}^{m} H_j t} = \left(\prod_{j=1}^{m} e^{-iH_j t/r}\right)^{r} + O(m^2 t^2/r), \tag{2.22}$$

which implies that if r scales as m^2t^2/ϵ, then the error can be made at most ϵ for any $\epsilon > 0$. In the QAOA case, we simply have $H = H_C + H_M$ and we treat each of the evolution time intervals as variational parameters that are optimized classically (can be seen as $t/r \equiv \gamma_l, \alpha_l$). This means we do not use this *Trotterization* approach directly, but it heavily inspires the QAOA ansatz nevertheless.

Let me now further detail the practical realization of the algorithm. Reiterating, the aim of MaxCut is to maximize the number of edges in a graph that are "cut" by a given partition of the vertices into two sets (cf. Figure 2.1). One way of implementing QAOA for MaxCut is to consider: H_C as in Eq. 2.19, where the sum is over all edges of the studied graph, connecting vertices (j, k); and $H_M = \sum_{j=1}^{N} \boldsymbol{X_j}$, for N qubits. In this scheme, the associated cost function, $\langle\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})| H_C |\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})\rangle$, merely rewards cases where we have edges between nodes of different sets. Furthermore, we can, now, also write

$$U_{H_{C_l}} = e^{-i\gamma_l H_C} = \prod\nolimits_{\text{Edge (j,k)}} e^{-i\gamma_l(1-\boldsymbol{Z}_j\boldsymbol{Z}_k)/2} \tag{2.23}$$

$$U_{H_{M_l}} = e^{-i\alpha_l H_M} = \prod\nolimits_{j=1}^{n} e^{-i\alpha_l \boldsymbol{X}_j} \tag{2.24}$$

These give us the form of the unitaries that are applied in each layer of the QAOA ansatz, representing the *Trotterized* time evolution of Figure 2.10 (red and blue boxes). They correspond to the complex exponentials of the cost and mixer Hamiltonians, respectively: $e^{-i\gamma_l H_C}$ and $e^{-i\alpha_l H_M}$. Notice how $U_{H_{M_l}}$ uses Pauli-x operators, instead of the usual Pauli-z in $U_{H_{C_l}}$. In practice, the mixer Hamiltonian terms $e^{-i\alpha_l H_M}$ correspond to $R_x(\alpha_l/2)$ and are easy to implement. The cost Hamiltonian terms, on the other hand, are a bit more tricky, requiring the use of 2 CNOT gates. Each of the terms $e^{-i\gamma_l(1-\boldsymbol{Z}_j\boldsymbol{Z}_k)/2}$ can be transpiled into a quantum circuit, as shown in Figure 2.11, below. For each edge in the graph, we'll have one of these terms, in each of the p layers of the QAOA ansatz.

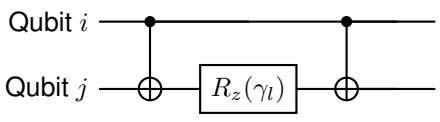


Figure 2.11: $e^{i\gamma_l \boldsymbol{Z}_i \boldsymbol{Z}_j/2}$ decomposition.

This allows us to re-write Eq. 2.20 as

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\alpha})\rangle = U_{H_{M_p}} U_{H_{C_p}} ... U_{H_{M_1}} U_{H_{C_1}} |\psi_0\rangle \tag{2.25}$$

In QAOA, it is customary to start with a uniform superposition over the n bit-string basis states, i.e., $|\psi_0\rangle = |+_n\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle$. This is achieved by applying a Hadamard gate to each of the qubits, at the start of the quantum circuit.

Subsequently, a classical routine is employed to optimize the values of the parameters by minimizing the negative of the cost (analogous to the cut), therefore enabling the extraction of the MaxCut partition. Experimentally, for each step of the classical optimization, it is necessary to compute expectation values of Pauli-z operators, which requires the quantum circuit to be sampled a certain number of times (shots). Additionally, in case it was not yet clear, note that in QAOA we use one qubit for each node of the graph, with the qubits' values indicating which set they correspond to: $|0\rangle$: "Set 0"; $|1\rangle$: "Set 1". As such, for the graph in Figure 2.1, we'd require 5 qubits. Exemplifying, if the most sampled basis state at the end of the quantum circuit is $|00011\rangle$, then nodes 1, 2 and 3 belong to "Set 0", while nodes 4 and 5 belong to "Set 1".

2.3.1.3 Variational Qubit-Efficient MaxCut Heuristic Algorithm (QEMC) Review

The Qubit Efficient MaxCut Heuristic Algorithm (QEMC) [3] is somewhat similar to QAOA. After all, it was heavily inspired by it. However, it has a number of crucial differences. First, it only requires $n = \log_2(N)$ qubits, instead of N, where N is the number of nodes in the graph. Additionally, the QEMC algorithm is based on a novel probability threshold encoding scheme, a suitable cost function, and a parameterized unconstrained quantum circuit. Going in order:

Probability threshold encoding scheme

With n qubits, each of the $N = 2^n$ basis states will represent one of the graph's nodes. Following the sampling of the quantum circuit, a probability distribution is generated. Nodes with probabilities exceeding a certain threshold, $p_{th} = \frac{1}{2B}$, belong to "Set 1", while probabilities below this value indicate inclusion in "Set 0". This strongly diverges from how we encode set inclusion in QAOA.

Suitable cost function

The objective function utilized in QEMC is the following:

$$L(\{p(i)\}) = \sum_{\substack{j<k: \\ \{j,k\} \in E}} \left[\left(d(j,k) - \frac{1}{B} \right)^2 + \left(s(j,k) - \frac{1}{B} \right)^2 \right], \tag{2.26}$$

where $d(j,k) = |p(j) - p(k)|$ and $s(j,k) = p(j) + p(k)$ are the absolute difference and sum of the corresponding states' probabilities. The idea is that as both $d(j,k)$ and $s(j,k)$ tend towards $1/B$, the probability of one node approaches zero (distinctive "Set 0"), while the probability of the other node

approaches $1/B$ (distinctive "Set 1"), without specifying which is which. Ultimately, just like for QAOA, connections between nodes of different sets are favoured. Note, however, that this probability threshold encoding scheme assumes, *a priori*, that one of the sets ("Set 1") has B nodes. Nevertheless, this is not an issue, as we can efficiently iterate through all potential values of $B = 1, ..., \lfloor \frac{N}{2} \rfloor$. Frequently, it is reasonable to set $B = N/2$, and we shall use this as our starting point.

Problem-agnostic quantum circuit

The QEMC circuit ansatz is agnostic to specific graph instances, a departure from QAOA where the graph structure is explicitly encoded in the quantum circuit. Instead, the graph is implicitly encoded through the cost function. As a result, the QEMC quantum circuit is not bound to any particular form and only needs to be expressive enough to approximate the optimal states in the Hilbert space. Such problem-independent ansatz approach provides considerable flexibility in ansatz selection. Frequently, the circuit ansatz known as "Strongly Entangling Layers" is employed, as depicted below.

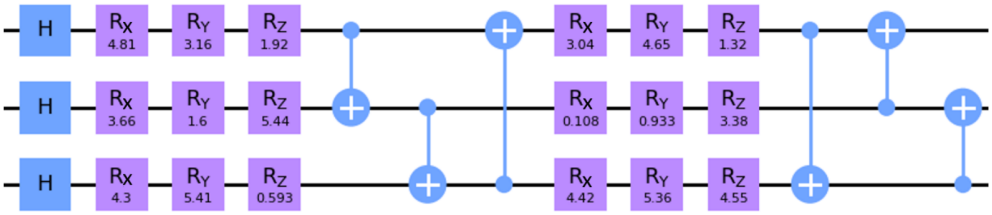


Figure 2.12: "Strongly Entangling Layers" circuit ansatz: case of $n = 3$ qubits and $p = 2$ layers. After a single layer of Hadamard gates, each subsequent layer consists of $3n$ single-qubit parameterized rotation gates and n CNOT gates (entangling gates). Reproduced from [3].

These constitute the main ingredients necessary to understand the QEMC algorithm. At this point, the same hybrid loop as before would be run, so as to optimize the "Strongly Entangling Layers" ansatz's parameters, to minimize the cost function. To read the MaxCut partition, generated by the QEMC algorithm, one would sample the circuit's output one more time, after the training, and build the basis states' probabilities distribution. The threshold p_{th} would then be applied to determine each nodes' set inclusion, from each of their associated basis states' probabilities.

One notable, and perhaps unfortunate, property of QEMC is that it is efficiently simulable classically. Due to the exponential compression of the number of qubits, the algorithm can be feasibly run on a classical computer, even for large graphs, hence defeating its purpose as a quantum algorithm. This is something the authors of the algorithm [3] realized in hindsight. For this reason, it is now termed a quantum-inspired classical algorithm.

Chapter 3

Base iQAQE Algorithm

To motivate the development of this algorithm, let us summarize the merits and drawbacks of both QAOA and QEMC. QEMC boasts several advantages over QAOA, including significantly lower qubit requirements ($n = \log_2(N)$ *vs.* N) and the ability to accommodate shallower circuit depths [3]. Additionally, the threshold probability encoding scheme allows QEMC to associate each graph partition with a volume of quantum states, potentially enhancing noise resilience. Furthermore, it is also believed that QEMC might be better trainable [3]. Conversely, QEMC demands the entire probability distribution at each optimization step (to compute the cost), resulting in an overall increase in the number of needed shots when compared to QAOA, which solely necessitates expectation values. However, as we've mentioned before, it has been shown [3] that QEMC can be efficiently simulated entirely classically, somewhat undermining its purpose as a quantum algorithm.

With a thorough grasp of these two algorithms, the *HQCC* project collaboration has put forward a novel, hybrid algorithm that integrates aspects from both, intending to capitalize on their individual strengths for a more substantial outcome. This is what we will be discussing next.

3.1 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE)

3.1.1 Theoretical Framework

The proposed algorithm aspires to act as an interpolation between QAOA and QEMC, aiming to amalgamate the strengths of both into a more robust approach, tentatively named iQAQE (Interpolated QAOA/QEMC). In iQAQE, we depart from the QEMC approach by associating each graph node with a list of basis states, in contrast to QEMC's consideration of a single basis state for each node. Each of these lists comprises somewhere between $\left[1, 2^N - 1\right]$ basis states, where N represents the number of nodes. This design allows for potential overlap among states from different lists/nodes. Additionally, it is important to note that the encoding of these states will utilize a qubit range expected to fall between the QEMC and QAOA requirements, specifically in $[\log_2 N, N]$. With all that said, the main goal of my thesis is to ascertain the optimal mapping from basis states to lists, essentially determining which basis states should be included in specific lists/nodes. Naturally, this undertaking also involves the identification of a

suitable cost function and ansatz to ensure the algorithm's effective operation.

3.1.2 Motivation & Outlook

At this point, it is important to mention that the development of this algorithm is entirely exploratory, although we do believe it should allow for a more practical algorithm implementation-wise. We can postulate that such an hybrid approach might have some potential advantages. Namely, it should allow for less shots than in QEMC, and fewer qubit requirements than in QAOA, in addition to, arguably, being better trainable. Another interesting characteristic of such an algorithm is its tunable "quantumness". As we interpolate between QAOA (hybrid quantum-classical) and QEMC (classical, quantum-inspired), the ability to selectively adjust the degree of both could prove to be advantageous. As part of a more long-term perspective (not in this thesis), I may explore the possibility of generalizing this algorithm to tackle additional combinatorial problems, such as Max-k-Cut. Throughout my thesis, I will rigorously test the algorithm using classical simulations of quantum machines, aiming to identify the most effective implementation strategies considering both cost and ansatz considerations.

3.1.3 Algorithm Description & Workflow

A rough sketch of the workflow, in its simplest form, is provided below:

1. Select the number of qubits: `n_qubits`$= n \in [\log_2(N), N]$, where N refers to the graph's number of nodes;
2. Select the cardinality of the lists, to associate with each graph node: `list_cardinality`$= c \in [1, 2^n - 1]$, where n represents the chosen number of qubits. Note that c is taken to be the same for all the nodes, although this could change in future implementations;
3. Assignment/mapping – from basis states to the lists: c basis states are assigned to each graph node;
4. To obtain each node's probability: sum the probabilities of the basis states in the list (and normalize the results, so they add to 1);
5. Keep the same cost function and ansatz as in the QEMC algorithm (just adapt the ansatz to the correct number of qubits).

I will now provide some more information on each of these steps.

Selecting the number of qubits

In selecting the number of qubits, we allow for any intermediate value between the QEMC [3] ($\log_2(N)$) and QAOA [2] (N) limits. This should give us more flexibility regarding the degree of "quantumness" that we want our algorithm to exhibit. This is to be understood in the context of QEMC being entirely classical, whereas QAOA is hybrid, quantum-classical. As such, by choosing a number of qubits

in between these two limits, we expect our algorithm to be hybrid, in terms of its quantum nature. In addition to this, the number of qubits, alongside the ansatz, will also influence the expressivity of the model. If one opts for a QEMC-type exponential compression, not only does that render the scheme classically simulable in an efficient manner, but it also hinders its expressivity [3]. One way to fight back against this would be to use more layers in the ansatz (if it allows for this). However, as this would require greater circuit depths, we might run into trouble trying to implement this in actual quantum hardware, in the current NISQ era. On the other hand, we also do not want to use all the N (QAOA) qubits, as this quickly becomes impractical for larger graph instances. (Think hundreds of nodes upwards.) With all that being said, we should look for the perfect balance between the two limits that provides us with the best results. This might vary depending on the specific graph instance that we are solving for.

Another thing to keep in mind, which is connected to the next point in our workflow, is that all of this depends on the type of encoding that we end up choosing. If we opt for rather specific means of encoding each node's color, or set (e.g., expectation values of Pauli strings, like in [18]), this will *a priori* restrict the number of qubits that we use. (In the case of [18], the Pauli string encoding automatically enforces a polynomial compression on the number of qubits.) So far, we've served ourselves of QEMC's probability threshold encoding scheme, which relies only on the nodes' probabilities. (The nodes' colours are, then, decided depending on whether they are above, or below, the threshold: $p_{th} = \frac{1}{2B}$, for a chosen B value.) This gives us more flexibility, in the sense that we are, then, free to choose how to obtain the nodes' probabilities from the basis states'[1].

Selecting the cardinality of the lists

In case it wasn't crystal clear by now, we associate a list of basis states to each graph node, from which we shall extract the node's corresponding probability value (used to, afterwards, define its colour, based on the aforementioned probability threshold encoding scheme). The cardinality, c, is chosen to be in the interval $\left[1, 2^{n-1}\right]$, where n represents the number of qubits. Once again, this is to be understood as an interpolation between the QEMC (1) and QAOA (2^{n-1}) limits. *A priori*, it is not possible to know which values of c would be the optimal, as the algorithm's performance depends on many, interconnected factors. Namely, the number of qubits, ansatz, cost function, colour encoding, etc. In any case, in an attempt to study this performance, as a function of the values of c, we perform grid-searches on this parameter.

Assignments/mapping

There is also quite some complexity in how we go about this mapping from basis states to lists. For now, we've settled for doing it in a random fashion, attributing c, random, basis states, generated from the n selected qubits, to each of the N graph nodes. In doing so, we only make sure that all the basis states are used at least once, so we're not discarding their information. We note, however, that there might be potential benefits in utilizing a more complex/elaborate mapping. This is a possibility that we

[1] This matter of going from the basis states' probabilities to the nodes' is another, whole can of worms. We'll discuss this, in-depth, in point 4.

definitely intend to explore further. As an example of one such, "special", mapping, take the QAOA mapping, for instance: for each graph node, we fix a single of the N qubits[2] to one, allowing for all the possible, remaining (2^{N-1}) permutations of the remaining qubits. Each of these lists (of 2^{N-1} elements) will be associated to one graph node, with each node having a different qubit fixed to one.

Obtaining the nodes' probabilities from the basis states'

Above, it is mentioned that we simply add the probabilities of each node's associated basis states to obtain a final node probability, which is, naturally, normalized afterwards. This is the most basic way we could go about doing this. Alternatively, there might be hidden merits in finding more elaborate ways of computing these nodes' probabilities. As a matter of example, one could make use of the arithmetic (or geometric) average of the basis states' probabilities to obtain the nodes' (again, followed by normalization). This is something that is worth spending some time on, which we intend to pursue further. However, one must be very careful when designing these schemes, as in certain circumstances a mismatch between the ansatz and this encoding (which will influence the objective function's value) can lead to constant probabilities for the nodes. Well, independent probabilities (implying a constant cost) mean we cannot optimize the variational circuit, so tread carefully. Additionally, keep in mind that the normalization process itself might also kill the probabilities' dependence on the variational parameters (under certain scenarios): one more thing to be wary of[3].

Cost function and ansatz

For now, we make use of the QEMC cost function, based on the previously mentioned probability threshold encoding scheme:

$$L(\{p(i)\}) = \sum_{\substack{j<k: \\ \{j,k\}\in E}} \left[\left(d(j,k) - \frac{1}{B} \right)^2 + \left(s(j,k) - \frac{1}{B} \right)^2 \right], \tag{3.1}$$

where $d(j,k) = |p(j) - p(k)|$ and $s(j,k) = p(j) + p(k)$ are the absolute difference and sum of the corresponding nodes' probabilities. The idea is that as both $d(j,k)$ and $s(j,k)$ tend towards $1/B$, the probability of one node approaches zero (distinctive "Set 0"), while the probability of the other node approaches $1/B$ (distinctive "Set 1"), without specifying which is which. Ultimately, just like for QAOA, connections between nodes of different sets are favoured. Note, however, that this probability threshold encoding scheme assumes, *a priori*, that one of the sets ("Set 1") has B nodes[4], which may not be true for the MaxCut partition. In the unluckiest of scenarios, we might be required to iterate through all potential values of $B = 1, ..., \lfloor \frac{N}{2} \rfloor$ (Although, in practice, this should never happen, if B is chosen carefully.) Frequently, though, it is perfectly reasonable to set $B = N/2$, and we shall use this as our starting point.

[2] Remember that in QAOA, $n = N$.

[3] This actually happened once, while we were testing using the QAOA mapping and ansatz, together with QEMC's cost function and probability threshold encoding scheme. This had been a test to try to recover the QAOA limit, from the iQAQE algorithm, but it failed magnificently (in this specific formulation).

[4] This is a value that we are free to choose.

Provided our current technique for mapping the basis states to the nodes' lists, it made sense for us to use the QEMC cost function, instead of QAOA's, as the latter relies on a problem Hamiltonian, defined on N qubits, which we might not have, depending on our choice of n. (More often than not, this will be the case: $n < N$. After all, we're also looking to reduce the number of qubits, so it wouldn't make sense to keep $n = N$.)

For similar reasons, we also make use of the "Strongly Entangling Layers" ansatz, featured in the QEMC scheme. This problem-agnostic ansatz allows us to be flexible in the number of qubits that we choose, being more adequate for this algorithm's implementation, when compared to QAOA's problem-inspired ansatz. Nevertheless, there are plans to attempt to develop more dynamic ansätze, which would be correlated to each individual mapping that is performed. Ideally, this would increase the algorithm's performance, by taking into account formulation-specific information. One should also note that this might require an equally adapted objective function, developed with these aspects in mind.

Final notes

The iQAQE algorithm, as it is enunciated here, has already proven to yield reasonable results. Nevertheless, these were not systematically better than either QAOA or QEMC. More testing is required to fine tune the values of n and c, as well as the number of layers and Adam's learning rate, for each specific graph instance. This is something that will require a more in-depth analysis of the algorithm, so we can find an efficient and systematic way to do so. (Can explore grid-searches, e.g.) Further analysis on how the number of shots impacts the algorithm's performance are also due, as this is something we intend to improve relative to QEMC, which is especially reliant on a large number of shots per iteration, as it requires estimates of the circuit's output's probability distribution.

Chapter 4

Implementation details

In this chapter, we outline the numerical implementation of the models discussed in Chapters 2 and 3, focusing on pseudo-code representations. Furthermore, we highlight the Python libraries employed and provide specific code details, accompanied by a brief discussion on the benchmarks developed to evaluate the algorithms' performance.

4.1 Individual Algorithms

For numerical implementation of the previously described algorithms, we employ PennyLane [38], a versatile Python library designed for differentiable programming of quantum computers. PennyLane facilitates the execution of variational quantum circuits and their simultaneous training, akin to training a classical neural network, within the same Python environment, which is highly convenient. It offers numerous features, including automatic differentiation, crucial for optimizing variational quantum circuits.

In our implementation with PennyLane, we utilize one of two devices from its extensive selection. Firstly, we employ the `default.qubit` device, a straightforward state-vector qubit simulator implemented in Python, compatible with Autograd, JAX, TensorFlow, and Torch backends. This device is well-suited for optimizations involving a small to moderate number of qubits and parameters, offering exact expectation values. Secondly, we utilize the `lightning.qubit` device, a fast state-vector qubit simulator with a C++ backend. Recommended for scenarios involving moderate numbers of qubits and parameters or when utilizing stochastic expectation values. Our simulations utilize both devices; the former excels in analytical simulations of small quantum circuits, which predominates our work, while the latter is preferable for numerical simulations of larger quantum circuits, especially when multiple shots are required. Although analytical simulations are favored due to their lower computational resource requirements, they may lack the output sampling aspect of a true quantum computer simulation. Nonetheless, they suffice for our goal of optimizing variational quantum circuit parameters.

Moreover, we heavily rely on the NetworkX [39] Python library, a valuable tool for creating, manipulating, and analyzing complex networks. NetworkX aids in generating the graphs utilized in the MaxCut problem, offering useful visualization tools for graph representation and partitioning visualization, essen-

tial for interpreting algorithm results.

Additionally, we utilize the CVXPY [40] Python library, specifically designed for convex optimization tasks. CVXPY facilitates solving the semidefinite programming (SDP) relaxation of the MaxCut problem within the GW algorithm framework, which inherently constitutes a convex optimization problem.

Regrettably, we lack access to a genuine quantum computer, which would have been invaluable for assessing the performance of these algorithms on real hardware. Furthermore, the absence of access to any High-Performance Computing (HPC) cluster significantly restricted our ability to conduct large-scale simulations and grid-searches, such as those required for hyperparameter tuning. These limitations occasionally posed a bottleneck in our research, particularly concerning our inability to evaluate the algorithms on larger graphs, which would have been crucial for comprehensive testing. All simulations presented herein were conducted using my personal computer – a system equipped with an 8-core AMD Ryzen 7 5800H CPU, integrated Radeon graphics, and 16GB of RAM.

Lastly, all simulations were performed using the Adam classical optimizer [41], with default parameters except for the learning rate, which was treated as a hyperparameter and fine-tuned accordingly. The Adam optimizer is a popular optimization algorithm for variational quantum algorithms, utilizing stochastic gradient descent with adaptive learning rates and momentum.

4.1.1 Quantum Approximate Optimization Algorithm (QAOA)

Below, we present the pseudo-code representation of the Quantum Approximate Optimization Algorithm (QAOA) implementation, as described in section 2.3.1.2.

Algorithm 1 Quantum Approximate Optimization Algorithm

Require: `(n_layers not None) and (init_parameters not None)`

Ensure: `n_qubits = self.n_nodes`

1. Define U_C and U_M as for usual QAOA
2. Create the variational quantum circuit, using a `@qml.qnode`, based on the QAOA ansatz (Layering U_C and U_M p times, for $p =$ `n_layers` layers). Note that we start with `qml.Hadamard` on all wires
3. The objective function is defined as the expectation value of the problem Hamiltonian, obtained from the previously defined `@qml.qnode`, through `qml.expval(H_C)`
4. Initialize the Adam optimizer, using its default parameters
5. Start timer & training:

while Stopping criteria `not True` **do**

   5.1 Adapt the value of `parameters`, using stochastic gradient descent (Adam)

   5.2 Store the cut and cost function values, alongside the approximation ratio, for each iteration

   5.3 Break if `max_iter` or `abs_tol` or `rel_tol` is reached, depending on which are specified explicitly

end while

6. Stop timer: training time estimate
7. Sample the circuit, so as to obtain the most frequently sampled bitstring
8. Use that to obtain the computed partition & respective cut value

All of the developed code has been compiled in a GitHub repository, available at https://github.com/kaiuki2000/HQCC_Code_Implementations/tree/main. For access to the code, please contact the author.

4.1.2 Qubit-Efficient MaxCut Heuristic (QEMC)

Here, we provide the pseudo-code representation of the implementation of the Qubit-Efficient MaxCut Heuristic (QEMC), as detailed in section 2.3.1.3.

Algorithm 2 Qubit Efficient MaxCut Heuristic Algorithm

Require: `(n_layers not None) and (parameters not None) and (B is not None)` ▷ `B` is a parameter of the cost function

Ensure: `n_qubits = self.n_nodes`

1. Define the QEMC ansatz layer (Strongly Entangling Layers)
2. Create the variational quantum circuit, using a `@qml.qnode`, based on the QEMC ansatz. Note that we start with `qml.Hadamard` on all wires
3. The objective function is defined according to the probability threshold encoding scheme (more information below)
4. Initialize the Adam optimizer, using its default parameters
5. Start timer & training:

while Stopping criteria `not True` **do**

5.1 Optimize the value of `parameters`, using stochastic gradient descent (Adam)

5.2 Store the cut and cost function values, alongside the approximation ratio, for each iteration

5.3 Break if `max_iter` or `abs_tol` or `rel_tol` is reached, depending on which are specified explicitly

end while

6. Stop timer: training time estimate
7. Sample the circuit, so as to obtain the output's probability distribution
8. Use that to obtain the computed partition (using the probability threshold encoding scheme) & respective cut value

4.1.3 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE)

This section introduces the pseudo-code for the iQAQE algorithm.

4.1.4 Goemans-Williamson Algorithm

Here, we present the numerical implementation of the Goemans-Williamson model, in the form of pseudo-code.

4.2 Benchmarking and Testing Methods

How do we benchmark our models? This should be described here: mention the Avg. BSF metric, and how it was ”wrong”, initially, and how it was ”fixed”. Also mention any other possible metrics that could be used to compare the models: Grid-searches, etc.

Basic test cases to compare the implemented model against other numerical tools (verification) and experimental data (validation).

I should also introduce the utilized score metrics: Best-so-far average and median, etc. Maybe, mention the difference between the before and after of the ”BSF Correction”.

Chapter 5

iQAQE Schemes and Results

Insert your chapter material here. - In this chapter, we should present all the many schemes that we've come up with, and the results of the numerical simulations of these schemes. We should also compare the results of the different schemes, and discuss the potential of each one. We should also mention the importance of the results and the potential applications of the schemes. This will, surely, be the largest chapter of the thesis.

Here, I should present each realization of iQAQE, theoretically, as well as the results of the numerical simulations of each realization. I should also mention that this work is, essentially, a collection/compilation of different schemes/ideas, all inspired by the base-iQAQE algorithm, described above. As such, some schemes will appear disconnected from others, but they all serve the same purpose, ultimately. (This is how I should justify the existence of so many schemes, and the lack of a clear connection between them. Also, this is how one should look at this.)

I might re-order these a little.

5.1 Base iQAQE (Random)

Base iQAQE schemes and their results.

5.2 Polynomial Compression-type Encodings

Polynomial Compression-type Encodings and their results.

5.2.1 Basic Polynomial Compression-type iQAQE

Basic Polynomial Compression-type iQAQE schemes and their results.

5.2.2 Parity-like QAOA

Parity-like QAOA schemes and their results.

5.2.3 Correlation-based iQAQE

Correlation-based iQAQE schemes and their results.

5.2.4 Fixed-Parity iQAQE

Fixed-Parity iQAQE schemes and their results.

5.3 Parity-like QAOA

Parity-like QAOA schemes and their results.

5.4 Other Exploratory Ideas

Other exploratory ideas and their results.

5.4.1 Tranche-based Oracle colouring

Tranche-based Oracle colouring schemes and their results.

5.5 Extended-QEMC

Extended-QEMC scheme and variations thereof, and their results.

5.5.1 Vanilla Extended-QEMC

Vanilla Extended-QEMC scheme and variations thereof, and their results.

5.5.2 Cardinality $= k$ Extended-QEMC

Cardinality $= k$ Extended-QEMC scheme and variations thereof, and their results.

5.6 Goemans-Williamson and Bigger Graphs

Goemans-Williamson scheme and its results on bigger graphs.

5.7 Alternative ansatzë

Alternative ansatzë and their results: More specifically, Non-deterministic CNOTs. Could also feature some discussion on problem-inspired *vs.* problem-agnostic ansatzë. Mention how we've been

using problem-agnostic ansatzë, in iQAQE, and how this hinders the results. Explain how/why problem-inspired is better, and how we could use it in the future. (Could refer to the QML review paper, here.)

5.8 Average Best-so-Far correction

Average Best-so-Far correction and its results. Mention how it was ”wrong”, initially, and how it was ”fixed”. I’m not sure where to include this, though.

5.9 Randomized iQAQE benchmarking

Talk about the randomized benchmarking that we’ve been doing, and how it’s been helping us to understand the performance of the different schemes. Type 1 and 2 variables, etc. This could have many subsections.

Chapter 6

Conclusions

Insert your chapter material here.

6.1 Achievements

The major achievements of the present work.

6.2 Future Work

A few ideas for future work.

Bibliography

[1] J. Preskill. Quantum computing 40 years later, 2023.

[2] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm, 2014.

[3] Y. Tene-Cohen, T. Kelman, O. Lev, and A. Makmal. A variational qubit-efficient maxcut heuristic algorithm, 2023.

[4] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1, 2001. doi: 10.1109/ICCV.2001.937505.

[5] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988. doi: 10.1287/opre.36.3.493.

[6] J. Poland and T. Zeugmann. Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts. In L. Todorovski, N. Lavrač, and K. P. Jantke, editors, *Discovery Science*, pages 197–208, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46493-8.

[7] R. M. Karp. *Reducibility Among Combinatorial Problems*, pages 219–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-540-68279-0. doi: 10.1007/978-3-540-68279-0_8. URL https://doi.org/10.1007/978-3-540-68279-0_8.

[8] A. Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005. URL https://www.frontiersin.org/articles/10.3389/fphy.2014.00005.

[9] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL https://doi.org/10.1145/227683.227684.

[10] R. Mirka and D. P. Williamson. An experimental evaluation of semidefinite programming and spectral algorithms for max cut. *ACM J. Exp. Algorithmics*, 28, aug 2023. ISSN 1084-6654. doi: 10.1145/3609426. URL https://doi.org/10.1145/3609426.

[11] J. A. Soto. Improved analysis of a max-cut algorithm based on spectral partitioning. *SIAM Journal on Discrete Mathematics*, 29(1):259–268, 2015. doi: 10.1137/14099098X. URL https://doi.org/10.1137/14099098X.

[12] S. Sen. Simulated annealing approach to the max cut problem. In U. M. Fayyad and R. Uthurusamy, editors, *Applications of Artificial Intelligence 1993: Knowledge-Based Systems in Aerospace and Industry*, volume 1963 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 61–66, Mar. 1993. doi: 10.1117/12.141755.

[13] Y.-H. Kim, Y. Yoon, and Z. W. Geem. A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm and evolutionary computation*, 44:130–135, 2019.

[14] G. G. Guerreschi and A. Y. Matsuura. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific Reports*, 9(1):6903, May 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-43176-9. URL https://doi.org/10.1038/s41598-019-43176-9.

[15] Z. Zhou, Y. Du, X. Tian, and D. Tao. Qaoa-in-qaoa: solving large-scale maxcut problems on small quantum machines, 2022.

[16] K. Ender, A. Messinger, M. Fellner, C. Dlaska, and W. Lechner. Modular parity quantum approximate optimization. *PRX Quantum*, 3(3):030304, 2022.

[17] K. Ender, R. ter Hoeven, B. E. Niehoff, M. Drieb-Schön, and W. Lechner. Parity Quantum Optimization: Compiler. *Quantum*, 7:950, Mar. 2023. ISSN 2521-327X. doi: 10.22331/q-2023-03-17-950. URL https://doi.org/10.22331/q-2023-03-17-950.

[18] M. Sciorilli, L. Borges, T. L. Patti, D. García-Martín, G. Camilo, A. Anandkumar, and L. Aolita. Towards large-scale quantum optimization solvers with few qubits, 2024.

[19] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 9781139495486. URL https://books.google.pt/books?id=-s4DEy7o-a0C.

[20] S. Slussarenko and G. J. Pryde. Photonic quantum information processing: A concise review. *Applied Physics Reviews*, 6(4), 2019.

[21] L. Madsen, F. Laudenbach, M. Askarani, F. Rortais, T. Vincent, J. Bulmer, F. Miatto, L. Neuhaus, L. Helt, M. Collins, A. Lita, T. Gerrits, S. Nam, V. Vaidya, M. Menotti, I. Dhand, Z. Vernon, N. Quesada, and J. Lavoie. Quantum computational advantage with a programmable photonic processor. *Nature*, 606:75–81, 06 2022. doi: 10.1038/s41586-022-04725-x.

[22] L. Henriet, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Reymond, and C. Jurczak. Quantum computing with neutral atoms. *Quantum*, 4:327, Sept. 2020. ISSN 2521-327X. doi: 10.22331/q-2020-09-21-327. URL https://doi.org/10.22331/q-2020-09-21-327.

[23] X. Wu, X. Liang, Y. Tian, F. Yang, C. Chen, Y.-C. Liu, M. K. Tey, and L. You. A concise review of rydberg atom based quantum computation and quantum simulation*. *Chinese Physics B*, 30(2): 020305, Feb. 2021. ISSN 1674-1056. doi: 10.1088/1674-1056/abd76f. URL http://dx.doi.org/10.1088/1674-1056/abd76f.

[24] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2), 2019.

[25] H.-L. Huang, D. Wu, D. Fan, and X. Zhu. Superconducting quantum computing: a review. *Science China Information Sciences*, 63(8), July 2020. ISSN 1869-1919. doi: 10.1007/s11432-020-2881-9. URL http://dx.doi.org/10.1007/s11432-020-2881-9.

[26] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020. ISSN 1947-5462. doi: https://doi.org/10.1146/annurev-conmatphys-031119-050605. URL https://www.annualreviews.org/content/journals/10.1146/annurev-conmatphys-031119-050605.

[27] A. S. Cacciapuoti, M. Caleffi, R. Van Meter, and L. Hanzo. When entanglement meets classical communications: Quantum teleportation for the quantum internet. *IEEE Transactions on Communications*, 68(6):3808–3833, June 2020. ISSN 1558-0857. doi: 10.1109/tcomm.2020.2978071. URL http://dx.doi.org/10.1109/TCOMM.2020.2978071.

[28] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5): 3457–3467, Nov. 1995. ISSN 1094-1622. doi: 10.1103/physreva.52.3457. URL http://dx.doi.org/10.1103/PhysRevA.52.3457.

[29] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[30] H. Wu, X. Feng, and J. Zhang. Quantum implementation of the sand algorithm and its quantum resource estimation for brute-force attack. *Entropy*, 26(3), 2024. ISSN 1099-4300. doi: 10.3390/e26030216. URL https://www.mdpi.com/1099-4300/26/3/216.

[31] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9): 625–644, Aug. 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00348-9. URL http://dx.doi.org/10.1038/s42254-021-00348-9.

[32] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Na-*

ture Communications, 5(1), July 2014. ISSN 2041-1723. doi: 10.1038/ncomms5213. URL http://dx.doi.org/10.1038/ncomms5213.

[33] P. D. Johnson, J. Romero, J. Olson, Y. Cao, and A. Aspuru-Guzik. Qvector: an algorithm for device-tailored quantum error correction, 2017.

[34] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, Sep 2022. ISSN 2662-8457. doi: 10.1038/s43588-022-00311-3. URL https://doi.org/10.1038/s43588-022-00311-3.

[35] C. Y.-Y. Lin and Y. Zhu. Performance of qaoa on typical instances of constraint satisfaction problems with bounded degree, 2016.

[36] Z. Wang, S. Hadfield, Z. Jiang, and E. G. Rieffel. Quantum approximate optimization algorithm for maxcut: A fermionic view. *Phys. Rev. A*, 97:022304, Feb 2018. doi: 10.1103/PhysRevA.97.022304. URL https://link.aps.org/doi/10.1103/PhysRevA.97.022304.

[37] Intro to QAOA. https://pennylane.ai/qml/demos/tutorial_qaoa_intro/, 2024. Accessed: 2023-12-18.

[38] Pennylane Website. https://pennylane.ai/, 2024. Accessed: 2024-05-13.

[39] NetworkX Website. https://networkx.org/, 2024. Accessed: 2024-05-13.

[40] CVXPY Website. https://www.cvxpy.org/, 2024. Accessed: 2024-05-13.

[41] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.