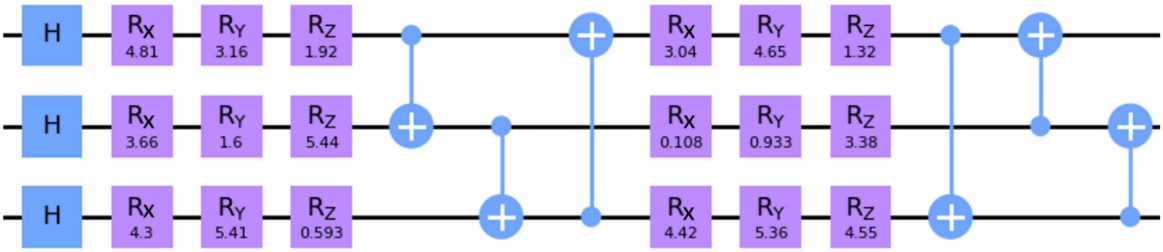
TÉCNICO LISBOA



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. João 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

Acknowledgments . . . vii
Resumo . . . ix
Abstract . . . xi
List of Tables . . . xv
List of Figures . . . xvii
Nomenclature . . . xix
List of acronyms . . . xxi

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Topic Overview . . . 2
1.3 Objectives . . . 3
1.4 Thesis Outline . . . 3

2 Theoretical background **5**
2.1 The Maximum Cut Problem & Computational Complexity . . . 5
2.1.1 Classical State-of-the-Art Algorithms for MaxCut . . . 7
2.1.1.1 Goemans-Williamson Algorithm . . . 8
2.1.2 Hybrid Quantum-Classical State-of-the-Art Algorithms for MaxCut . . . 10
2.2 Quantum Computing Primer . . . 12
2.3 Hybrid Quantum-Classical Computing . . . 17
2.3.1 Variational Quantum Algorithms . . . 17
2.3.1.1 Basic structure of a VQA . . . 19
2.3.1.2 Quantum Approximate Optimization Algorithm (QAOA) . . . 22
2.3.1.3 Variational Qubit-Efficient MaxCut Heuristic Algorithm (QEMC) . . . 24

3 iQAQE Algorithm Framework **27**
3.1 Comparison of QAOA and QEMC . . . 27
3.2 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE) . . . 27
3.2.1 Theoretical Model . . . 27
3.2.2 Motivation & Outlook . . . 28
3.2.3 Algorithm Description & Workflow . . . 29

4 Implementation **33**
4.1 Individual Algorithms . . . 33
4.1.1 Quantum Approximate Optimization Algorithm (QAOA) . . . 34
4.1.2 Qubit-Efficient MaxCut Heuristic (QEMC) . . . 35
4.1.3 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE) . . . 35
4.1.4 Goemans-Williamson Algorithm . . . 36
4.2 Benchmarking and Testing Methods . . . 37

5 iQAQE Schemes and Results **39**
5.1 Random iQAQE . . . 40
5.2 Polynomial Compression-type Encodings . . . 41
5.2.1 Basic Polynomial Compression-type iQAQE . . . 41
5.2.2 Correlation-based iQAQE . . . 46
5.2.3 Fixed-Parity iQAQE . . . 48
5.3 Extended-QEMC . . . 50
5.3.1 Unmodified Extended-QEMC . . . 50
5.3.2 Cardinality $= 1$ Extended-QEMC . . . 52
5.4 Alternative Ansätze . . . 52
5.5 Goemans-Williamson and Bigger Graphs . . . 54
5.6 Average Best-so-Far correction . . . 56
5.7 Randomized iQAQE benchmarking . . . 56

6 Exploratory Ideas **63**
6.1 Parity-like QAOA . . . 63
6.2 Tranche-based Oracle Colouring . . . 63

7 Conclusions **65**
7.1 Main Findings . . . 65
7.2 Future Work . . . 65

Bibliography **67**

List of Tables

3.1 Steps for the Base-iQAQE algorithm. 29

5.1 Cross-validation loss (average mean square error) and R^2 values for the two fitted models, Multiple Linear Regression and PCR, in the context of the usual 8-node graph. 60

List of Figures

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. 6

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]. 11

2.3 Bloch sphere representation. The sphere's north and south poles correspond to states $|0\rangle$ and $|1\rangle$, respectively. Reproduced from [27]. 15

2.4 Two-qubit gate example – The controlled-NOT gate. 16

2.5 Quantum circuit utilized in the famous Grover's search algorithm [29]. Integraly reproduced from [30]. 16

2.6 Applications of variational quantum algorithms. Sourced from [31] in its entirety. 18

2.7 Schematic diagram of a variational quantum algorithm. Adapted from [31]. 18

2.8 Quantum alternating operator ansatz. Adapted from [37]. 22

2.9 $e^{i\gamma_l \boldsymbol{Z}_i \boldsymbol{Z}_j/2}$ decomposition. 23

2.10 "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]. 25

3.1 Sample schematic representation of the iQAQE encoding scheme. Comparison between the QEMC and iQAQE encoding schemes: example for a trivial 4-node graph. In QEMC, each node is associated with a single basis state, while in iQAQE, each node is associated with a list of basis states. 28

5.1 Considered 8-node graph instance. The optimal cut (10) was found through brute-force (exhaustive search). 39

5.2 Comparison between the 3 VQAs' performances (QAOA, QEMC and iQAQE), using an infinite and finite number of shots. The best-so-far average is plotted. 40

5.3 BSF Average Approximation Ratio *vs.* iteration number for the tested VQAs: QAOA, QEMC, and iQAQE, using the aforementioned polynomial compression scheme. The number of layers considered were 4 for $k = 2$ and 5 for $k = 3$ 42

5.4 Average BSF Approximation Ratio *vs.* iteration number for the tested VQAs: QAOA, QEMC, and iQAQE, using the aforementioned polynomial compression scheme. The number of layers considered were 4 for $k = 2$ and 5 for $k = 3$ 42
5.5 Basic Polynomial Compression-type iQAQE – Grid search, using $k = 2$. All the simulations were done using an infinite number of shots (`shots = None`), and each value consists of the average of 10 iQAQE instances, using different random initial parameters. 44
5.6 Basic Polynomial Compression-type iQAQE – Grid search, using $k = 3$. All the simulations were done using an infinite number of shots (`shots = None`), and each value consists of the average of 10 iQAQE instances, using different random initial parameters. 44
5.7 Average approximation ratio ($\pm\sigma$) vs iteration number, for the three tested VQAs: `QAOA`, `iQAQE_QAOA` (as described above) and `iQAQE_BtIn`. The latter stands for iQAQE, with better initialization. This corresponds to the same mapping as the one in Figure 5.2. . . . 45
5.8 Correlation-based polynomial-type compression scheme ($k = 2, 3$ and 4): numerical simulations for the usual 8-node graph. 46
5.9 Correlation-based polynomial-type compression scheme ($k = 2, 3, 4, 5, 6$ and 7): numerical simulations for the usual 8-node graph. 47
5.10 Fixed-parity polynomial-type compression scheme: numerical simulations for the usual 8-node graph. 49
5.11 Fixed-parity polynomial-type compression scheme: numerical simulations for a 16-node graph. 50
5.12 Implementation of the Unmodified Extended-QEMC scheme for the usual 8-node graph: comparison with QAOA and regular QEMC for various values of m. Note that m is not extended beyond 5, as this would exceed the number of qubits used in QAOA 51
5.13 Implementation of the Cardinality $= 1$ Extended-QEMC scheme for the standard 8-node graph: comparison with QAOA and regular QEMC for various values of m. 52
5.14 Comparison of QEMC and Extended-QEMC implementations utilizing both D-CNOTs and ND-CNOTs for the usual 8-node graph, alongside QAOA. 53
5.15 Comparison of performance (average cut values) among various VQA schemes for the specified 32-node graph. `n_layers = 3` was employed for both QEMCs, with `step_size = 0.95` for D-CNOT-QEMC and `step_size = 0.5` for ND-CNOT-QEMC. For all Extended-QEMC schemes, a `step_size = 0.95` was utilized, with the number of layers specified in the legend. 54
5.16 Comparison of performance (average cut values) among various VQA schemes for the specified 100-node graph. This time, all considered VQAs utilized `n_layers = 3` and `step_size = 0.95`, except for Extended-QEMC with $m = 3$, which used `n_layers = 4`. . 55
5.17 Basis states' distributions over the 8 nodes of the usual graph, for many different `np.random.seed` seeds. Only those for which the average best-so-far approximation ratio [after 100 iQAQE iterations] was found to be ≥ 0.975 were selected. (These were the best performing seeds out of the 50 tested, for `seed` $\in [0, 49] \cap \mathbb{Z}$.) 57

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
BSF	Best-so-far
D-CNOT	Deterministic CNOT
DV	Dependent Variable
GW	Goemans-Williamson Algorithm
HPC	High Performance Computing
HQCC	Hybrid Quantum-Classical Computing
iQAQE	Interpolated QAOA/QEMC Hybrid Algorithm
IV	Independent Variable
MaxCut	Maximum Cut Problem
ML	Machine Learning
ND-CNOT	Non-deterministic CNOT
NISQ	Noisy Intermediate-Scale Quantum
PC	Principal Component
PCA	Principal Component Analysis
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
VIF	Variance Inflation Factor
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

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 7 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].

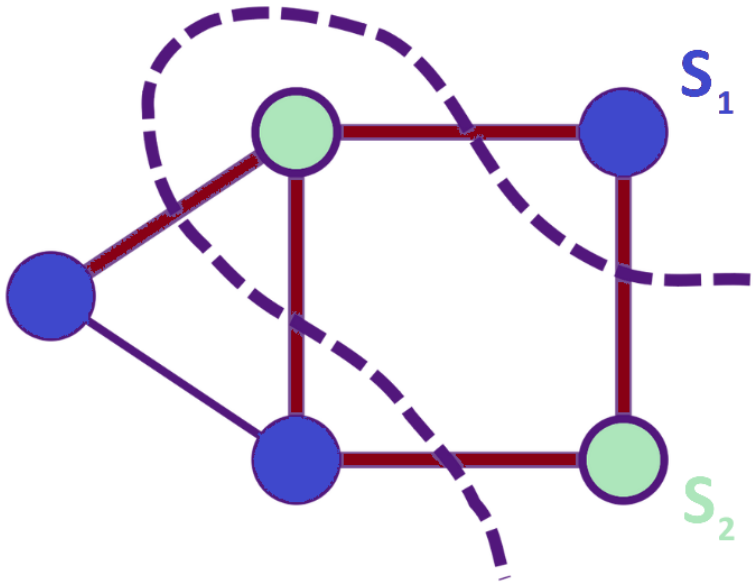


Figure 2.1: An example of a graph with a partition that maximizes the number of cut edges (red). Note that the MaxCut partition might not be unique.

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.

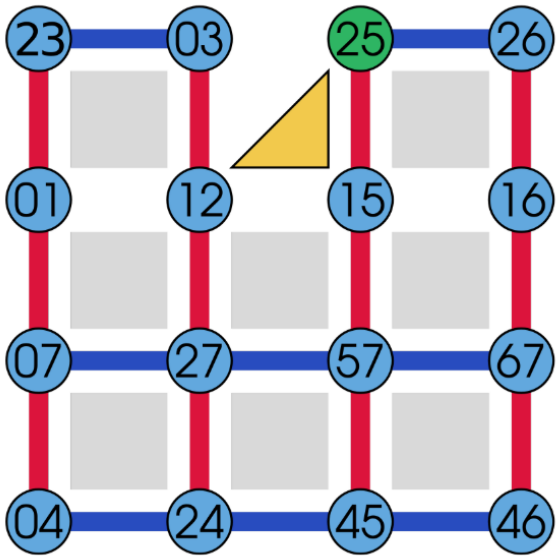


Figure 2.2: Planar lattice geometry illustration: the circular nodes represent the physical qubits, each of which can interact directly with its nearest neighbours. The lines represent terms of the driver Hamiltonian, from Parity-QAOA [16]. The numbers refer to edges in the original graph. Image reproduced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

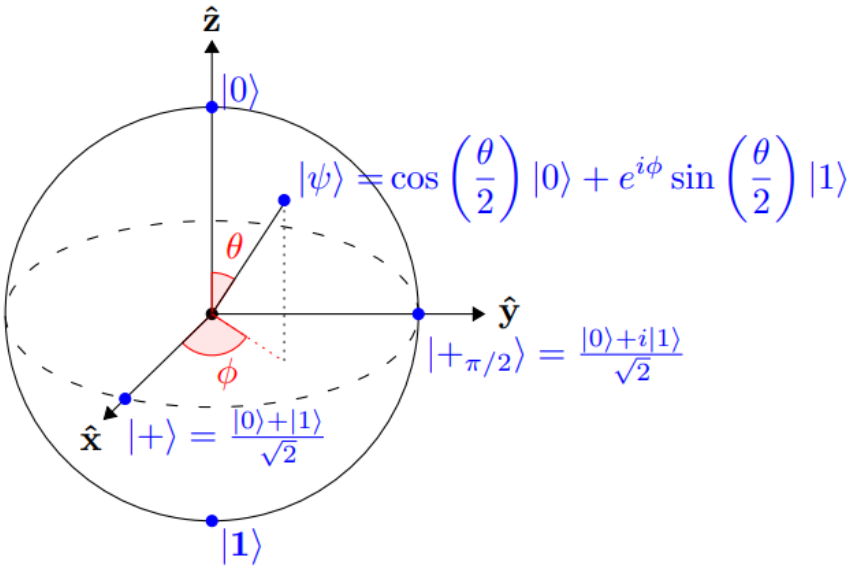


Figure 2.3: Bloch sphere representation. The sphere's north and south poles correspond to states $|0\rangle$ and $|1\rangle$, respectively. Reproduced from [27].

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,

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

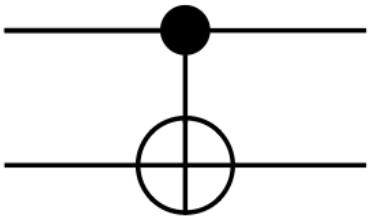
such that:

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

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 2.4.



(a) CNOT gate – Circuit representation. The black circle identifies the control qubit. The other qubit is the test qubit.

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

(b) CNOT gate – Equivalent matrix form.

Figure 2.4: 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 2.5, below, is designed for the implementation of the renowned Grover's search algorithm [29].

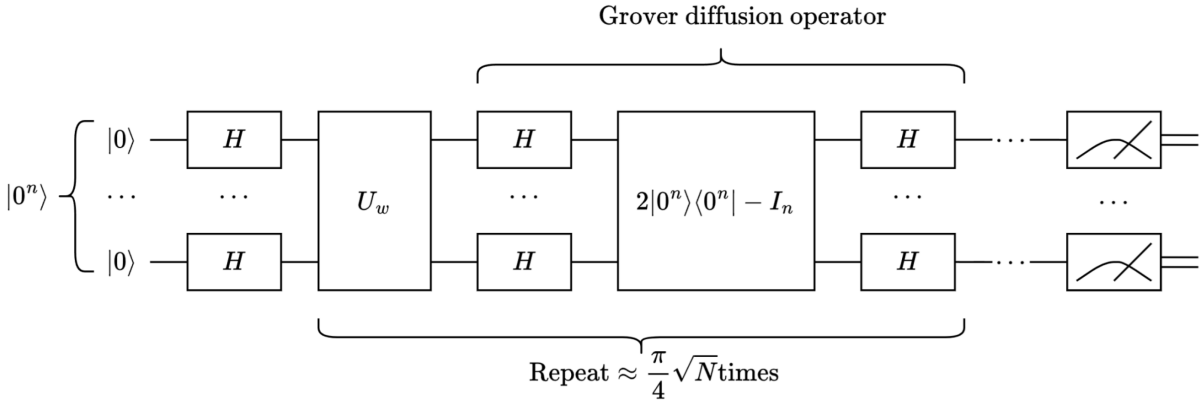


Figure 2.5: Quantum circuit utilized in the famous Grover's search algorithm [29]. Integraly reproduced from [30].

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 2.6).

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 2.6).

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.

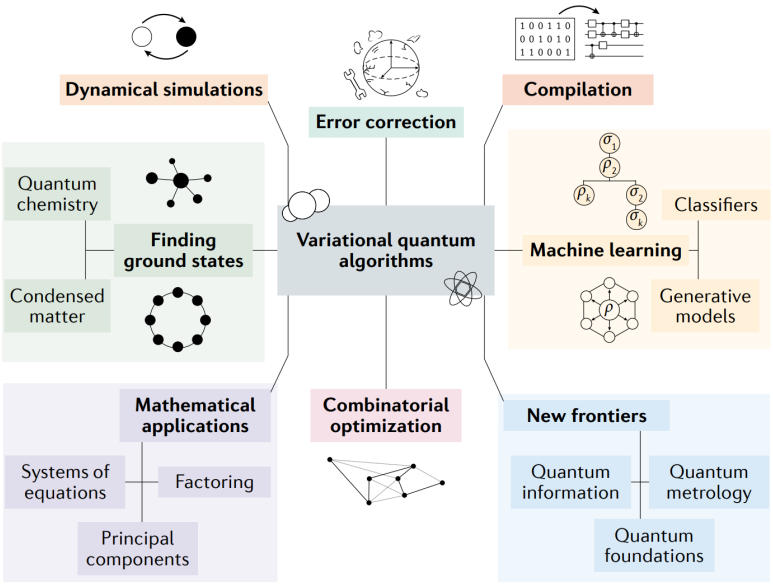


Figure 2.6: Applications of variational quantum algorithms. Sourced from [31] in its entirety.

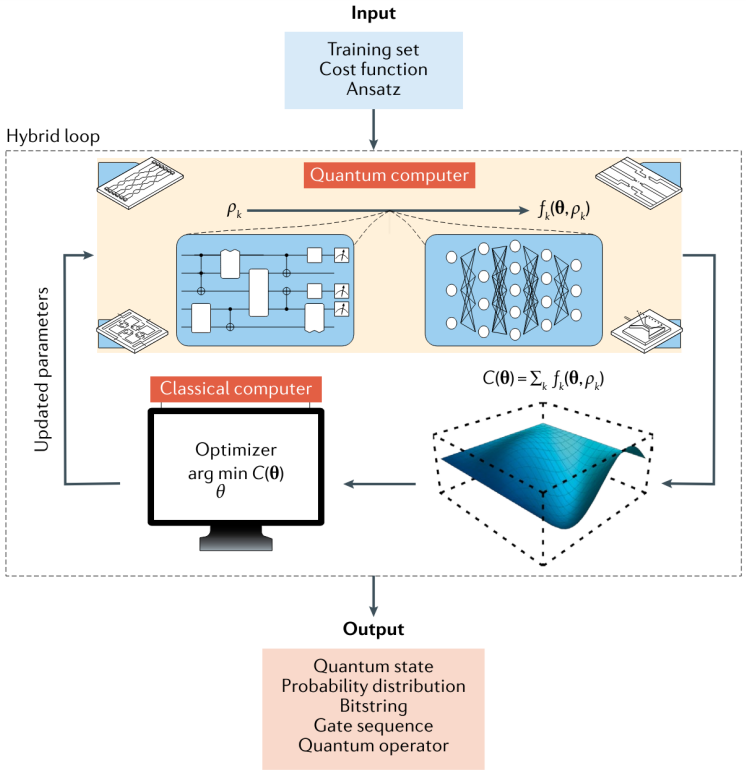


Figure 2.7: Schematic diagram of a variational quantum algorithm. Adapted from [31].

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 2.7), 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 2.7), 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.

Ansätze

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 ansatz 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 (Hybrid Loop)

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. All the VQAs described in this work will use this hybrid loop (cf. Figure 2.7). Therefore, specifying the cost function and ansatz will be sufficient to characterize them.

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 2.6): (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)

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 2.8, below, for $n \in \mathbb{N}$ QAOA layers.

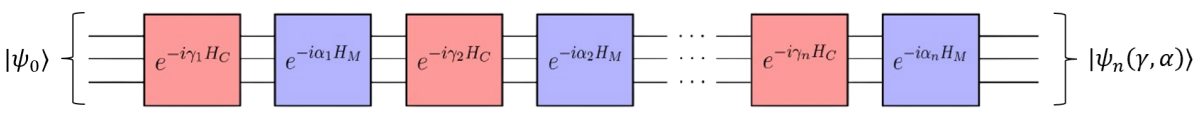


Figure 2.8: Quantum alternating operator ansatz. Adapted from [37].

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

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

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

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 2.8 (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 2.9, below. For each edge in the graph, we'll have one of these terms, in each of the p layers of the QAOA ansatz.

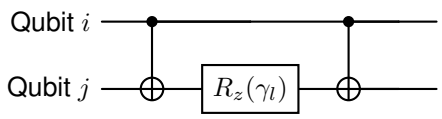


Figure 2.9: $e^{i\gamma_l \boldsymbol{Z}_i \boldsymbol{Z}_j/2}$ decomposition.

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)

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 2.10). As can be seen, this ansatz applies a series of parameterized single-qubit rotations interspersed with controlled entangling gates to generate a highly entangled quantum state. We use Pennylane's `qml.StronglyEntanglingLayers` implementation for this purpose.

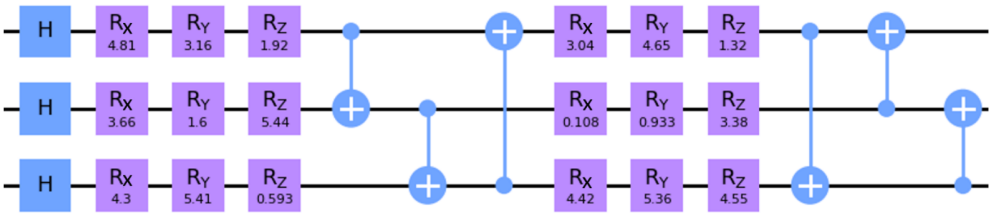


Figure 2.10: "Strongly Entangling Layers" circuit ansatz: case of $n = 3$ qubits and $p = 2$ layers. After a single layer of Hadamard gates, each subsequent layer consists of $3n$ single-qubit parameterized rotation gates and n CNOT gates (entangling gates). Reproduced from [3].

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

iQAQE Algorithm Framework

In this chapter, we present the proposed algorithm, iQAQE, which interpolates between QAOA and QEMC. We begin with a comparison of the two algorithms, followed by a detailed description of iQAQE, its theoretical framework, motivation, and workflow.

3.1 Comparison of QAOA and QEMC

To motivate the development of our new algorithm, iQAQE, we summarize the merits and drawbacks of both QAOA and QEMC. QEMC has several advantages over QAOA, including significantly lower qubit requirements ($n = \log_2(N)$ vs. N) and the ability to support shallower circuit depths [3]. The threshold probability encoding scheme allows QEMC to associate each graph partition with a volume of quantum states, potentially enhancing noise resilience. Moreover, QEMC might be better trainable [3]. However, QEMC requires the entire probability distribution at each optimization step to compute the cost, increasing the number of needed shots compared to QAOA, which only requires expectation values. Additionally, it has been shown that QEMC can be efficiently simulated classically, somewhat undermining its purpose as a quantum algorithm [3].

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.

3.2 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE)

3.2.1 Theoretical Model

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 (cf. Figure

3.1). Each of these lists comprises somewhere between $\left[1, 2^{\texttt{n\_qubits}-1}\right]$ basis states, where $\texttt{n\_qubits}$ represents the number of qubits. 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 $[\log_2 N, N]$, for an N-node graph. 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. For most of our work, we consider iQAQE to use the same ansatz and cost function as QEMC, adjusted for the appropriate number of qubits and with some modifications that we'll discuss below.

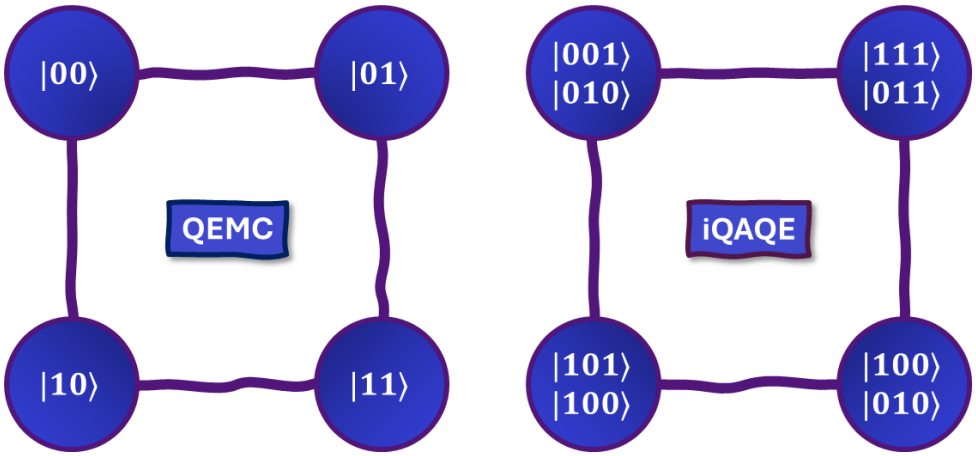


Figure 3.1: Sample schematic representation of the iQAQE encoding scheme. Comparison between the QEMC and iQAQE encoding schemes: example for a trivial 4-node graph. In QEMC, each node is associated with a single basis state, while in iQAQE, each node is associated with a list of basis states.

3.2.2 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 "quantum-ness". 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. The aim is to identify the most effective implementation strategies, taking into account both cost and ansatz considerations, as well as the distribution of basis states.

3.2.3 Algorithm Description & Workflow

A sketch of the workflow is provided below (Table 3.1). Following this, I will elaborate on each step. Regarding the hybrid loop, I will only detail the changes relative to QEMC.

Step	Description
1. **Select number of qubits**	`n_qubits`$= n \in [\log_2(N), N]$, where N refers to the graph's number of nodes.
2. **Select list cardinality**	`list_cardinality`$= c \in \left[1, 2^{n-1}\right]$, 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 work.
3. **Assignment/mapping**	Map c basis states to each graph node.
4. **Calculate nodes' probabilities (within the hybrid loop)**	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. **Cost function and ansatz (within the hybrid loop)**	Keep the same cost function and ansatz as in the QEMC algorithm, just adapt the ansatz to the correct number of qubits.

Table 3.1: Steps for the Base-iQAQE algorithm.

1. Select the number of qubits

In selecting the number of qubits, we allow for any intermediate value between the QEMC ($\log_2(N)$) and QAOA (N) limits. This should give us more flexibility regarding the degree of "quantum-ness" 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.

Another key consideration, closely tied to our next workflow step, is the chosen encoding method. This refers to how we determine each node's color and map basis states to respective lists. Different encoding strategies, such as using expectation values of Pauli strings (e.g., as seen in [18]), inherently influence the number of required qubits. For instance, the Pauli string encoding in [18] automatically enforces a polynomial compression on the number of qubits. Currently, we've employed QEMC's probability threshold encoding scheme, offering flexibility in deriving node probabilities from basis states. Further discussion on this will be provided in point 4. below.

2. Select list cardinality

If it hasn't been clear already, we assign a list of basis states to each graph node. The cardinality of each list, denoted as c, falls within the interval $\left[1, 2^{n-1}\right]$, where n represents the number of qubits used. This choice represents a balance between the QEMC (1) and QAOA (2^{n-1}) extremes. Determining the optimal value for c is non-trivial, as it depends on various interconnected factors including the number of qubits, ansatz, cost function, and color encoding. To explore its impact on algorithm performance, we conduct grid searches on this parameter.

3. **Assignments/mapping**

The process of mapping basis states to lists adds significant complexity. *A priori*, there's no clear-cut method for this mapping. The simplest approach involves randomly assigning c basis states to each node, ensuring all states are utilized at least once to avoid discarding information. However, exploring more intricate mappings may yield potential benefits. For instance, in the QAOA mapping, a single qubit among N is fixed to 1 for each node, allowing for 2^{N-1} permutations of the remaining qubits. Reiterating, each list, comprising 2^{N-1} elements, corresponds to a distinct graph node, with each node having a different qubit fixed to 1. This will result in a myriad of distinct mappings, which we'll thoroughly examine in Chapter 5, each presenting its unique advantages and disadvantages. The challenge lies in identifying the most effective mapping for a given problem instance. We aim to address this in our work.

4. **Calculate nodes' probabilities**

Above (Table 3.1), we mentioned a straightforward method of computing node probabilities by summing the probabilities of associated basis states and normalizing the result. This approach serves as our foundational strategy throughout this work. However, there may be hidden advantages in devising more intricate methods. For instance, one could explore calculating node probabilities using the arithmetic or geometric averages of basis states' probabilities, followed by normalization. Yet, designing such schemes requires caution, as mismatches between the ansatz and encoding could result in constant probabilities for nodes, impeding variational circuit optimization[1]. Furthermore, the normalization process itself may eliminate the probabilities' dependence on variational parameters under certain circumstances. While we explore alternative methods for calculating probabilities, our primary emphasis in this work remains on the basic approach described earlier, given its apparent robustness.

5. **Cost function and ansatz**

We employ the QEMC cost function, utilizing the probability threshold encoding scheme mentioned earlier. It's important to note that this scheme assumes, by default, that one of the sets ("Set 1") comprises B nodes[2], which may not align with the MaxCut partition. In some cases, we may need to iterate through potential values of B, ranging from 1 to $\lfloor \frac{N}{2} \rfloor$ – although, in practice, careful selection of B should prevent this scenario. Often, setting $B = N/2$ proves to be a reasonable starting point, and we adopt this as our default.

Given our approach to mapping basis states to node lists, it's logical for us to employ the QEMC cost function rather than QAOA's. The latter relies on a problem Hamiltonian defined on N qubits, which may not align with our choice of n. Typically, $n < N$ as we aim to reduce the number of qubits, making $n = N$ impractical.

Similarly, we opt for the "Strongly Entangling Layers" ansatz from the QEMC scheme for its versatility regarding qubit count. This ansatz, agnostic to specific problems, offers flexibility in our algorithm's im-

[1] This occurred once during testing when we used the QAOA mapping and ansatz alongside the QEMC cost function and probability threshold encoding scheme.

[2] This value is user-defined.

plementation compared to QAOA's problem-inspired ansatz. However, there are plans to explore more tailored ansatz designs correlated with individual mappings, aiming to enhance algorithm performance by leveraging formulation-specific information. Note that this may necessitate an adjusted objective function developed with these considerations in mind. While we experiment with problem-inspired Ansätze in this work, our primary focus remains on the Strongly Entangling Layers ansatz.

Chapter 4

Implementation

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' performances.

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 lack of access to a High-Performance Computing (HPC) cluster significantly limited our capacity to perform large-scale simulations and grid-searches necessary 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[1], 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 subsubsection 2.3.1.2.

Algorithm 1 Quantum Approximate Optimization Algorithm (QAOA)

Require: `graph`, `n_layers`, `init_parameters`

Ensure: `n_qubits = graph.n_nodes`

1. Define U_C and U_M
2. Create variational circuit using `@qml.qnode` with p layers of U_C and U_M

   Note: Start with `qml.Hadamard` on all qubits
3. Define objective function: $\langle H_C \rangle$, obtained through `qml.expval(H_C)`
4. Initialize Adam optimizer with default parameters
5. Start timer & begin training:

while True **do**

   5.1 Update `parameters` using Adam

   5.2 Store cut value, cost function, and approximation ratio, for each iteration

   if `max_iter` **or** `abs_tol` **or** `rel_tol` reached **then**

      break

   end if

end while

6. Stop timer: record training time
7. Sample circuit to get most frequent bitstring
8. Compute partition & corresponding cut value

[1] No GPU-assisted computations were conducted.

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 subsubsection 2.3.1.3.

Algorithm 2 Qubit Efficient MaxCut Heuristic Algorithm (QEMC)

Require: `graph`, `n_layers`, `init_parameters`, `B` ▷ `B` is a parameter of the cost function

Ensure: `n_qubits = graph.n_nodes`

1. Define QEMC ansatz layer (Strongly Entangling Layers)
2. Create variational circuit using `@qml.qnode` with QEMC ansatz
   Note: Start with `qml.Hadamard` on all qubits
3. Define objective function using probability threshold encoding scheme
4. Initialize Adam optimizer with default parameters
5. Start timer & begin training:

while True **do**

5.1 Update `parameters` using Adam

5.2 Store cut value, cost function, and approximation ratio, for each iteration

if `max_iter` **or** `abs_tol` **or** `rel_tol` reached **then**

break

end if

end while

6. Stop timer: record training time
7. Sample circuit to get output probability distribution
8. Compute partition using probability threshold encoding scheme & corresponding cut value

4.1.3 Interpolated QAOA/QEMC Hybrid Algorithm (iQAQE)

This subsection presents the pseudo-code for the iQAQE algorithm (Algorithm 3), as discussed in Chapter 3. It is important to note that point 3. of the algorithm involves significant complexity. The method for mapping the states is flexible and can be chosen in various ways. We will explore several possible mappings in Chapter 5.

Additionally, note the similarities between iQAQE and QEMC in terms of pseudo-code, as iQAQE is fundamentally built on the same framework.

Algorithm 3 Interpolated QAOA/QEMC Algorithm (iQAQE)

Require: `graph`, `n_layers`, `init_parameters`, `B` ▷ `B` is a parameter of the cost function

1. Select number of qubits: `n_qubits` = $n \in [\log_2(N), N]$
2. Select list cardinality: `list_cardinality` = $c \in [1, 2^{n-1}]$
3. Assignment/mapping: Map c basis states to each graph node
4. Define iQAQE ansatz layer (adapted Strongly Entangling Layers)
5. Create variational circuit using `@qml.qnode` with iQAQE ansatz

   Note: Start with `qml.Hadamard` on all qubits
6. Define objective function using probability threshold encoding scheme
7. Initialize Adam optimizer with default parameters
8. Start timer & begin training:

while True **do**

8.1 Update `parameters` using Adam

Note: Calculate nodes' probabilities by summing the probabilities of the basis states in each list and normalizing

8.2 Store cut value, cost function, and approximation ratio, for each iteration

if `max_iter` **or** `abs_tol` **or** `rel_tol` reached **then**

break

end if

end while

9. Stop timer: record training time
10. Sample circuit to get output probability distribution
11. Compute partition using probability threshold encoding scheme & corresponding cut value

4.1.4 Goemans-Williamson Algorithm

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

Algorithm 4 Goemans-Williamson Algorithm for MaxCut

Input: `graph`, Optional `MaxCut` value

Output: `Partition`, `Cut Value`, `Approximation Ratio` (if `MaxCut` is given), `Running Time`

1. Initialize: `n` (number of nodes), `edges` (`graph` edges)
2. Define SDP variable `X` and constraints `X >> 0` and `X[i, i] == 1 for i in range(n)`
3. Set up the objective function:

   `objective = sum(0.5 * (1 - X[i, j]) for (i, j) in edges)`
4. Start the timer and solve the SDP problem with the defined objective function and constraints, using CVXPY
5. Record the running time
6. Retrieve `X` and perform random-hyperplane "selection" to obtain the partition
7. Compute the cut value and, if applicable, the approximation ratio
8. Return the partition, cut value, approximation ratio (if applicable), and running time

4.2 Benchmarking and Testing Methods

Now that we have introduced all the algorithms and their numerical implementations, we must discuss the benchmarking and testing methods used to evaluate their performance. The primary metrics for evaluation are the cost plot (cost *vs.* iterations) and the approximation ratio plot (approximation ratio *vs.* iterations), both produced during training. These metrics help compare the algorithms' convergence rates and final cut values.

Given that we initialize our algorithms with random parameters, we devised statistically meaningful metrics to compare their performance. A single run isn't sufficient to draw conclusions about an algorithm's performance, as initial parameters can significantly impact the results. Thus, the primary statistical metric used is the best-so-far (BSF) average cut value. This metric averages the best-so-far cut values derived at each iteration during optimization. For a given training curve, with cut values *vs.* iterations, the BSF cut value is obtained by storing the highest cut value achieved so far. The code for this transformation is shown below:

```
def BSF_Transform(vec):
    """
    Transforms a vector into a best-so-far vector.
    """
    bsf_vec = vec.copy()
    for i in range(1, len(bsf_vec)):
        if bsf_vec[i] < bsf_vec[i-1]:
            bsf_vec[i] = bsf_vec[i-1]
    return bsf_vec
```

Listing 4.1: Auxiliary function: Best-so-far transformation

This transformation ensures that the BSF cut value curve is monotone increasing, as it always takes the best value so far during training. Additionally, we consider the median best-so-far value, which helps eliminate outliers often present in our simulations. By using the median, we can mitigate the impact of particularly lucky or unlucky initializations and better reflect the true performance of the algorithm. Both the average BSF and the median BSF curves are usually computed using either 3, 5 or 10 training curves[2], to ensure statistical robustness. Each curve is generated with different random initial parameters, obtained through `2 * numpy.pi * numpy.random.rand`, producing values in the range $[0, 2\pi]$.

Initially, we used the best-so-far of the average, but we found it less representative of the algorithms' best performances. We now prefer to first apply the BSF transform to obtain the best possible performance from each curve and then compute the average or median. This approach ensures that we account for the best performance of each curve, rather than just the best average performance. However, we initially used the best-so-far of the average for most of our work and only later switched to the new metrics. While this doesn't alter the interpretation of the results, it does affect the metric values, providing a more accurate quantification of the algorithms' performances.

Additionally, we conduct grid searches on the models' hyperparameters (such as Adam's learning rate, `n_layers`, `n_qubits`, etc.) to optimize their performance and understand how they behave under

[2]Unless explicitly stated otherwise, we always employ 10 runs for computing averages/medians

different conditions. Nevertheless, due to the unavailable access to an HPC cluster, these grid searches are often limited in scope.

To compare the algorithms' performances, we also benchmark them against the Goemans-Williamson algorithm. This comparison helps us evaluate how well our algorithms perform relative to state-of-the-art solutions. Most of our plots will include comparisons with QAOA[3], QEMC, and the GW algorithm for a comprehensive analysis of the algorithms' performances.

[3] If not specified otherwise, we default to using 3 QAOA layers.

Chapter 5

iQAQE Schemes and Results

In this chapter, we present the various iQAQE schemes we have developed and the results of their numerical simulations. These schemes explore different combinations of qubit numbers, list cardinalities, basis state mappings, and circuit layers. Each scheme features a unique mapping, resulting in distinct outcomes and properties. We compare these results and discuss the potential of each scheme. This chapter is essentially a collection of various ideas, all built upon the iQAQE Framework described in Chapter 3. While some schemes may seem disconnected, they all contribute to the overall goal, justifying their inclusion.

At times, our work diverged from iQAQE, leading us to explore other ideas for MaxCut algorithms. Although these ideas originated from our work on iQAQE, they are distinct enough to warrant their own chapter. These will be presented in the next chapter (Chapter 6).

From here onwards, unless explicitly stated otherwise, all VQAs are being tested on the following 8-node graph, whose optimal cut is 10 (Figure 5.1). This graph is one of the benchmarks used in [3], which is why we employ it here. Additionally, its small size allows for reasonably quick simulations on my personal computer, which is a significant advantage.

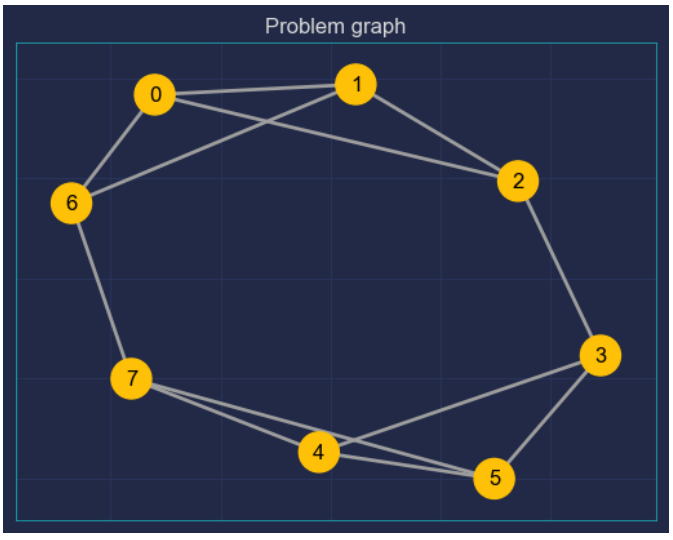


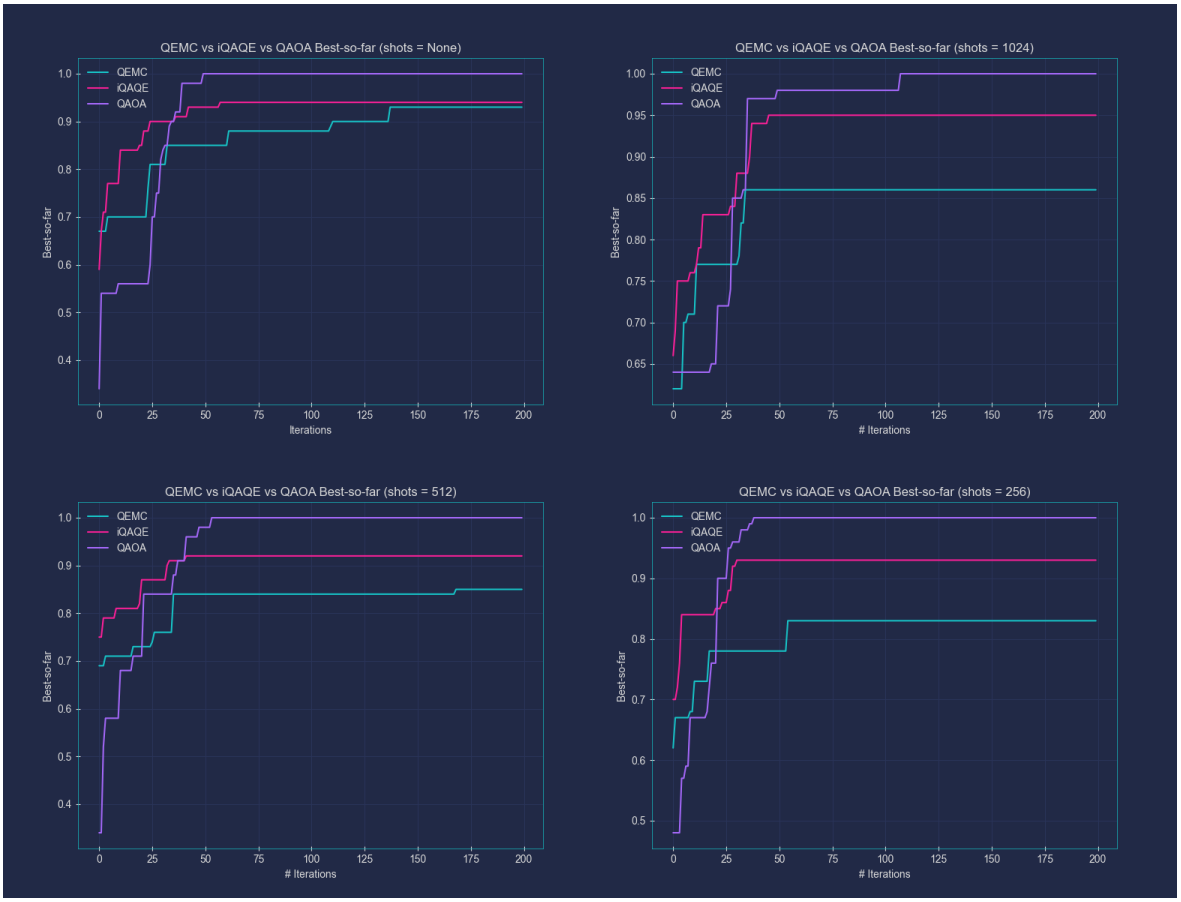
Figure 5.1: Considered 8-node graph instance. The optimal cut (10) was found through brute-force (exhaustive search).

5.1 Random iQAQE

First and foremost, we aim to understand how iQAQE compares with both QAOA and QEMC. In terms of iQAQE's implementation for this comparison, the selection of the number of qubits (n) and the list cardinalities (c) was not carefully considered. They were randomly chosen using `np.random.randint`, resulting in $n = 4$ and $c = 4$. Currently, the allocation of basis states to lists is also done randomly, without ensuring that all basis states are utilized. The goal of this initial simulation is to take a preliminary look at iQAQE's performance and understand how much the results vary between finite shot number simulations and analytical ones. (This also serves as a comparison test between the two previously mentioned Pennylane devices.) The plots below display the best-so-far average approximation ratio, derived from 10 iQAQE runs, each with different random initial parameters. The results for all three algorithms are generated using a configuration of 4 ansatz layers and an Adam learning rate of 0.99.

(a) Shots = None (Infinite)

(b) Shots = 1024



(c) Shots = 512

(d) Shots = 256

Figure 5.2: Comparison between the 3 VQAs' performances (QAOA, QEMC and iQAQE), using an infinite and finite number of shots. The best-so-far average is plotted.

These plots generally align with our theoretical expectations. As QEMC relies heavily on a large number of shots, transitioning to a finite shot number naturally impacts the results. Surprisingly, however, iQAQE seems to exhibit slightly greater resilience to this effect, despite sharing QEMC's cost function

and ansatz. This resilience might be attributed to the presence of multiple basis states associated with each graph node, potentially reducing the need for exhaustive sampling. If some basis states are harder to sample, the probabilities of the remaining states might adapt to compensate, as governed by the QEMC objective function. Nonetheless, QAOA consistently outperforms both approaches, highlighting its superiority. For the same number of layers, QAOA's problem-inspired ansatz and cost function consistently yield better results. To determine if iQAQE can surpass QAOA's performance, extensive testing across various combinations of parameters such as n, c, number of layers, and Adam's learning rate is necessary. This will motivate some of the grid searches we conduct later in this study. However, given the vast number of potential combinations, testing all of them is impractical. Therefore, some of the more targeted mappings/encodings explored in this work aim to constrain one or more of these variables, thereby reducing the total number of combinations for testing. Following this approach, conducting grid searches over the parameter values becomes more manageable and practical to determine the optimal settings. The subsequent sections of this chapter will detail these mapping strategies.

5.2 Polynomial Compression-type Encodings

At this point, driven by our earlier discussion on the nature of random-based algorithms, we begin exploring more meaningful mappings. Here, we opt to fix one or more qubits to 1 (or 0) while allowing the remainder to vary. The number of qubits we fix (k) depends on the desired order of compression. This approach not only enables the utilization of fewer qubits (if $k > 1$) but also establishes the total number of qubits to be used, simplifying subsequent grid searches over parameters. By fixing k qubits, we achieve polynomial compression of order-k (in the number of qubits). The total number of nodes that can be encoded in this manner is determined by $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, where n denotes the number of qubits. Essentially, this represents the number of ways we can select k qubits from the available n and set them to 1 (or 0), each combination encoding one graph node. Consequently, we need to ensure that $\binom{n}{k} \geq N$, where N represents the number of graph nodes. This criterion guides our selection of n and k. In practice, to determine n for any k, we select the smallest n that satisfies $\binom{n}{k} \geq N$. This yields $N = \mathcal{O}(n^k)$. Subsequently, we decide on the number of basis states to associate with each graph node, chosen from a total of 2^{n-k}. While we could potentially utilize all 2^{n-k} basis states for each node (with $n-k$ free qubits), we allow for the flexibility to choose only $c \leq 2^{n-k}$. The value of c requires optimization. With the mapping established, the algorithm proceeds as before, employing QEMC's cost function and ansatz. Notably, we cannot use QAOA's ansatz since $n \neq N$. Due to polynomial compression, $n < N$ always, unless $k = 1$ (in which case there is no compression).

5.2.1 Basic Polynomial Compression-type iQAQE

This is the simplest type of polynomial compression-based iQAQE scheme. It involves fixing k qubits to 1 instead of 0. The number of fixed qubits (k) is determined by the desired order of compression, as previously described. The results for $k = 2$ and $k = 3$ are shown below (Figures 5.3 and 5.4). In the

figures, the list cardinality is set to 4 for both the $k = 2$ curve (`iQAQE_k2`) and the $k = 3$ curve (`iQAQE_k3`). Additionally, the number of layers is set to 4 for $k = 2$ and 5 for $k = 3$, with an Adam learning rate of 0.98. The final curves are derived from 10 runs of the algorithms, each with different random initial parameters. The results are compared with those of QAOA and QEMC.

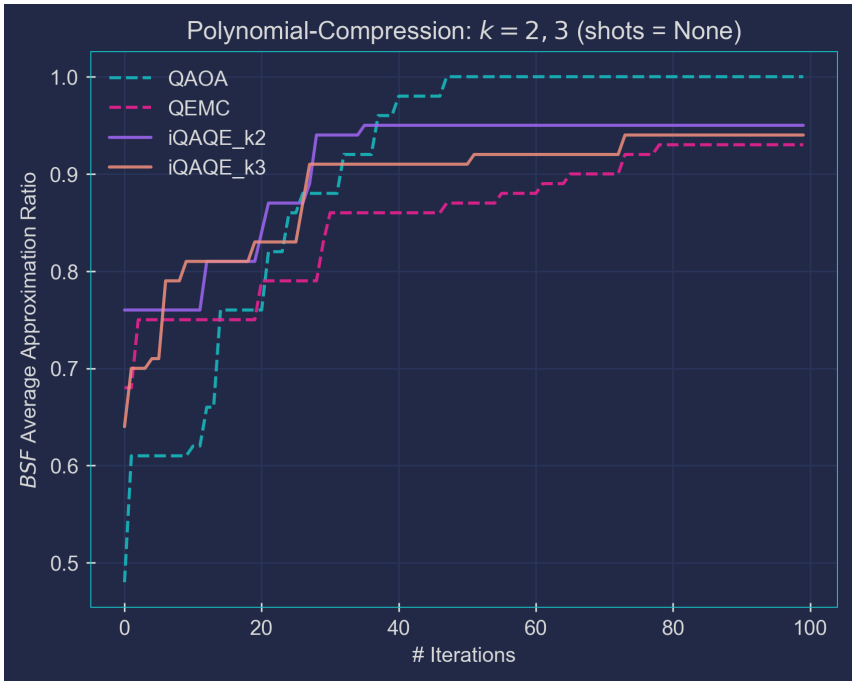


Figure 5.3: BSF Average Approximation Ratio *vs.* iteration number for the tested VQAs: QAOA, QEMC, and iQAQE, using the aforementioned polynomial compression scheme. The number of layers considered were 4 for $k = 2$ and 5 for $k = 3$

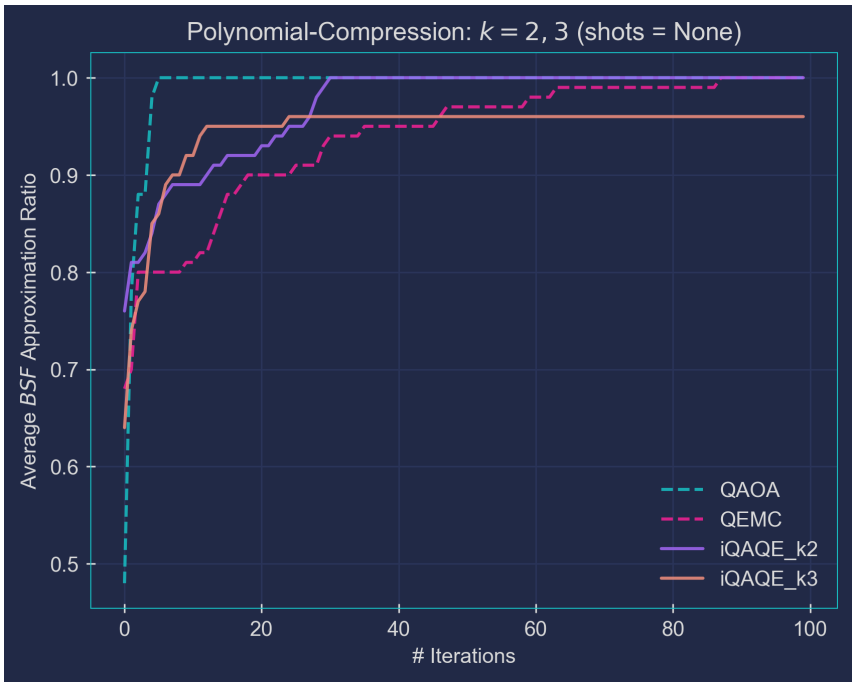


Figure 5.4: Average BSF Approximation Ratio *vs.* iteration number for the tested VQAs: QAOA, QEMC, and iQAQE, using the aforementioned polynomial compression scheme. The number of layers considered were 4 for $k = 2$ and 5 for $k = 3$

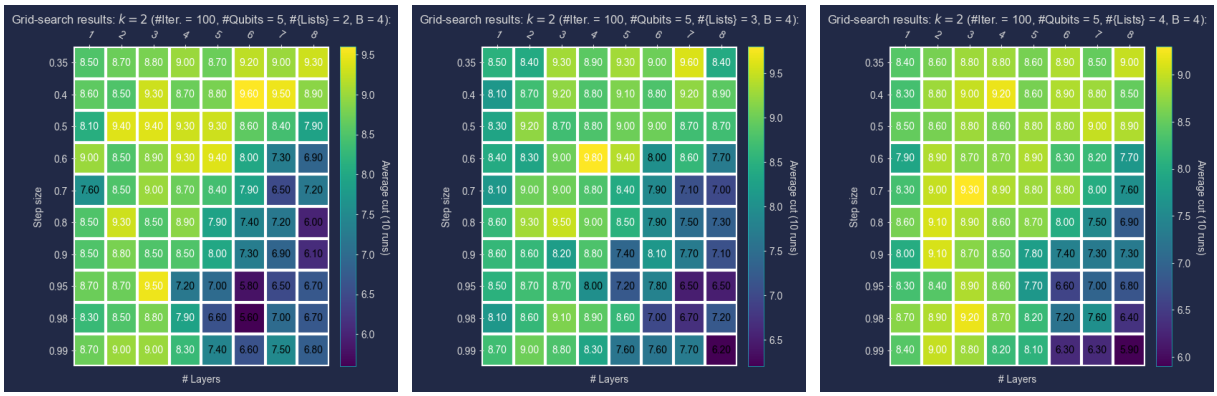
Notice the difference between the two figures. In the first figure, we average the curves first and then apply the BSF transformation. In the second figure, we reverse the process: first applying the BSF transformation to each curve and then averaging the transformed curves. The results are quite different, as illustrated. This indicates that both QEMC and iQAQE are generally more susceptible to outliers due to particularly unlucky initial parameterizations. This is an important consideration when interpreting the forthcoming results. Often, we will choose to discard the BSF average in favor of the average BSF. Nonetheless, both approaches are valuable for evaluating the algorithms' properties. Additionally, in some instances, we will simply use the average. Each case will be clearly specified.

Anyways, the primary aim of this study was to assess whether different types of mappings, like the one explored here, offer advantages compared to the random case. The results in Figures 5.3 and 5.4 indicate that this may be the case. At the very least, they suggest that there is potential in exploring these ideas further. For example, `iQAQE_k2` achieves a perfect approximation ratio, although it requires more iterations than QAOA. Nevertheless, our method surpasses QEMC for $k = 2$. The results for $k = 3$ are less encouraging. However, both mappings use fewer qubits ($n = 5$) than QAOA ($n = 8$), which is beneficial for implementation on practical (NISQ) quantum computers. The next step is to conduct grid searches over the parameters for this specific graph instance to determine if we can consistently achieve better performance than QAOA. This might provide further insights into the underlying principles of the algorithm's performance.

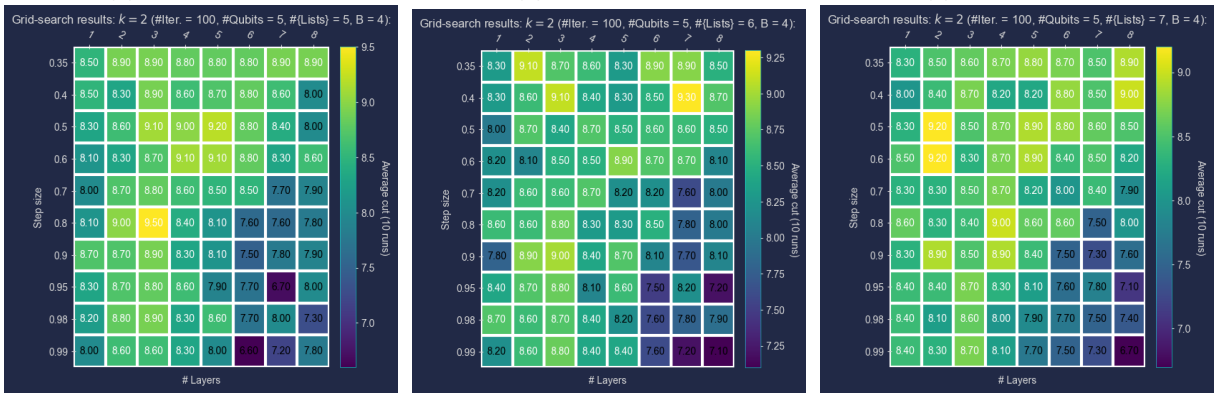
The results of the grid searches conducted for $k = 2$ and $k = 3$ are present in Figures 5.5 and 5.6. Various values were explored for the parameters: `step_size_list = [0.35, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.98, 0.99]`, `n_layers` ranging from 1 to 8, and `c` varying from 2 to $(2^{n-k} - 1)$, for $k = 2, 3$, where `n_layers` and `c` are integers. Notably, all simulations for the grid searches were executed with an infinite number of shots (`shots = None`). Additionally, this time, we utilized the average curve (from 10 instances) instead of the average best-so-far curve. This choice was made to evaluate the overall performance of the algorithm rather than focusing solely on the best-performing instances. Each run continued until convergence (`abs_tol <= 1e-5`) or `max_iter = 1000` was reached, followed by recording the final partition's cut. The objective was to identify any discernible patterns in the grid search to enhance the scheme's performance. However, no apparent pattern emerged, highlighting a recurring challenge in this work: the abundance of hyperparameters makes it challenging to optimize results through straightforward means. Later, we will explore employing a more intricate machine learning scheme to address this challenge. For now, we proceed with a different approach.

Recovering the QAOA limit ($k = 1$)

Next, we tested whether we could recover the QAOA limit using iQAQE's algorithm with what we call the QAOA mapping for this scheme. This mapping involves using the same number of qubits as there are nodes (as in standard QAOA) and assigning 2^{n-1} basis states to each graph node. For each graph node, a different qubit is fixed to 1, while the remaining $n - 1$ qubits are free to vary, resulting in 2^{n-1} basis states per node. The rest of the algorithm remains unchanged, employing QEMC's ansatz and cost function. The nodes' probabilities are derived from the normalized sum of the probabilities of

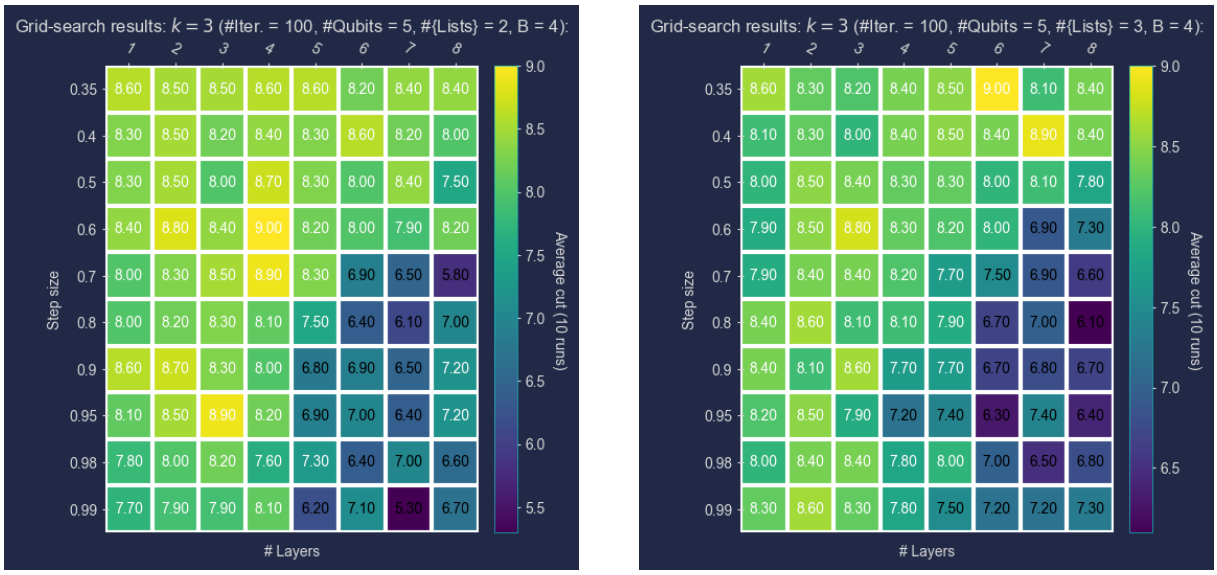


(a) `k = 2; c = 2` (b) `k = 2; c = 3` (c) `k = 2; c = 4`



(g) `k = 2; c = 5` (h) `k = 2; c = 6` (i) `k = 2; c = 7`

Figure 5.5: Basic Polynomial Compression-type iQAQE – Grid search, using $k = 2$. All the simulations were done using an infinite number of shots (`shots = None`), and each value consists of the average of 10 iQAQE instances, using different random initial parameters.



(a) `k = 3; c = 2` (b) `k = 3; c = 3`

Figure 5.6: Basic Polynomial Compression-type iQAQE – Grid search, using $k = 3$. All the simulations were done using an infinite number of shots (`shots = None`), and each value consists of the average of 10 iQAQE instances, using different random initial parameters.

their associated basis states. This approach was motivated by our desire to better understand iQAQE's properties and behavior. Previously, we observed (Figure 5.2) that iQAQE's performance seemed to fall between that of QAOA and QEMC, which is intuitive, as it is designed as an interpolation between the two. To further test this behavior, we used QAOA's mapping to see how closely we could approach QAOA's performance levels, which have been the best so far. The following results were obtained, focusing only on analytical simulations (i.e., infinite shots) – We employ the average curve obtained from 10 runs, as we did in the previous grid search:

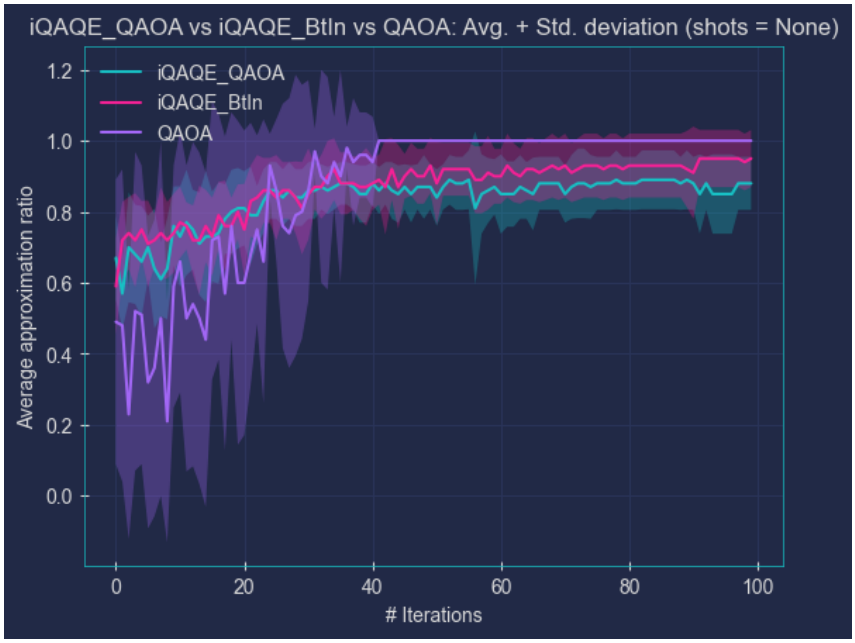


Figure 5.7: Average approximation ratio ($\pm\sigma$) vs iteration number, for the three tested VQAs: `QAOA`, `iQAQE_QAOA` (as described above) and `iQAQE_BtIn`. The latter stands for iQAQE, with better initialization. This corresponds to the same mapping as the one in Figure 5.2.

This realization led us to understand that we cannot expect to replicate QAOA's outstanding performance on this graph simply by using a QAOA-type encoding. After all, we are still relying on QEMC's cost function and ansatz, so this outcome is not entirely surprising. Moreover, to add to the challenge, we found that `iQAQE_QAOA`'s performance can even be surpassed by another random, non-specific mapping, highlighting how far we are from achieving QAOA-like results.

In an effort to bridge this gap, we attempted to integrate QAOA's ansatz into the scheme, alongside the QAOA-type encoding. However, after extensive testing, we realized this approach is not feasible. Here's why: using both elements together with the node probability encoding (as a normalized sum of their associated basis states' probabilities) causes the QEMC cost function to become independent of the ansatz's parameters[1]. This means we cannot proceed with training, rendering the scheme ineffective. Thus, implementing this approach in its current form is not possible. For it to be successful, other aspects would need to change, such as the cost function or the method of deriving node probabilities from basis states' probabilities, which is the main issue here.

[1] To verify this, we performed analytical calculations for a simple toy model: a graph with two nodes connected by a single edge. The results generalize to larger graphs.

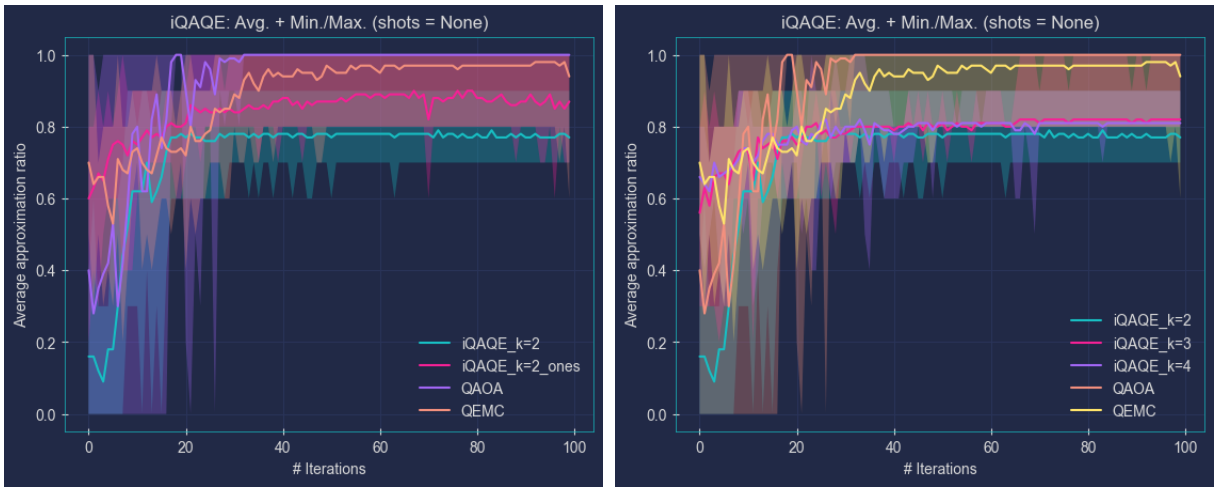
5.2.2 Correlation-based iQAQE

This scheme closely resembles what was previously discussed in subsection 5.2.1. The key distinction lies in the inclusion of the possibility for both 0's and 1's to be fixed. Consequently, the scheme aims to identify correlations among the different qubits, wherein "correlations" denote identical colors. In other words, the constructed lists regard the k fixed qubits as positively correlated, meaning they are either all set to 1 or all set to 0. This approach was inspired by the notion that by identifying correlations between nodes, it becomes feasible to color the entire graph as long as one node is initially colored. Although seeking correlations between qubits, as done here, differs from seeking correlations between nodes, we anticipated that it might yield promising results to some extent. In this scenario, the color of each node is determined by the probability that its k associated qubits are all positively correlated.

To delve deeper into the properties and characteristics of this scheme, we conducted a thorough formal analysis, exploring its limits by considering scenarios where k ranges from 1 to n qubits, covering the entire spectrum. Our primary objective was to ascertain if we could formulate this correlation-based iQAQE scheme in a manner that truly interpolates between QAOA and QEMC. We aimed to determine if we could retrieve both algorithms by adjusting the parameters accordingly. However, with this correlation-based scheme, we not only cannot recover QAOA, but we also fail to retrieve QEMC. Thus, this does not represent an interpolation between the two algorithms.

Likewise, earlier, we loosely referred to the iQAQE Framework as an interpolation. However, upon closer scrutiny, this characterization seems inaccurate. While it is possible to recover the QEMC limit[2], the same cannot be said for QAOA, as we have observed before.

Additionally, we provide the results of numerical simulations applied to the standard 8-node graph.



(a) Comparison of Correlation-based iQAQE for $k = 2$ with results from QAOA, QEMC, and the previous Basic Polynomial Compression-type iQAQE[3] (`iQAQE_k=2_ones`).

(b) Comparison of Correlation-based iQAQE results for $k = 2, 3$, and 4 with outcomes from QAOA and QEMC.

Figure 5.8: Correlation-based polynomial-type compression scheme ($k = 2, 3$ and 4): numerical simulations for the usual 8-node graph.

[2] Simply utilize the appropriate number of qubits and select one unique basis state for each node.
[3] This refers to the scenario where only 1's are fixed, not 0's.

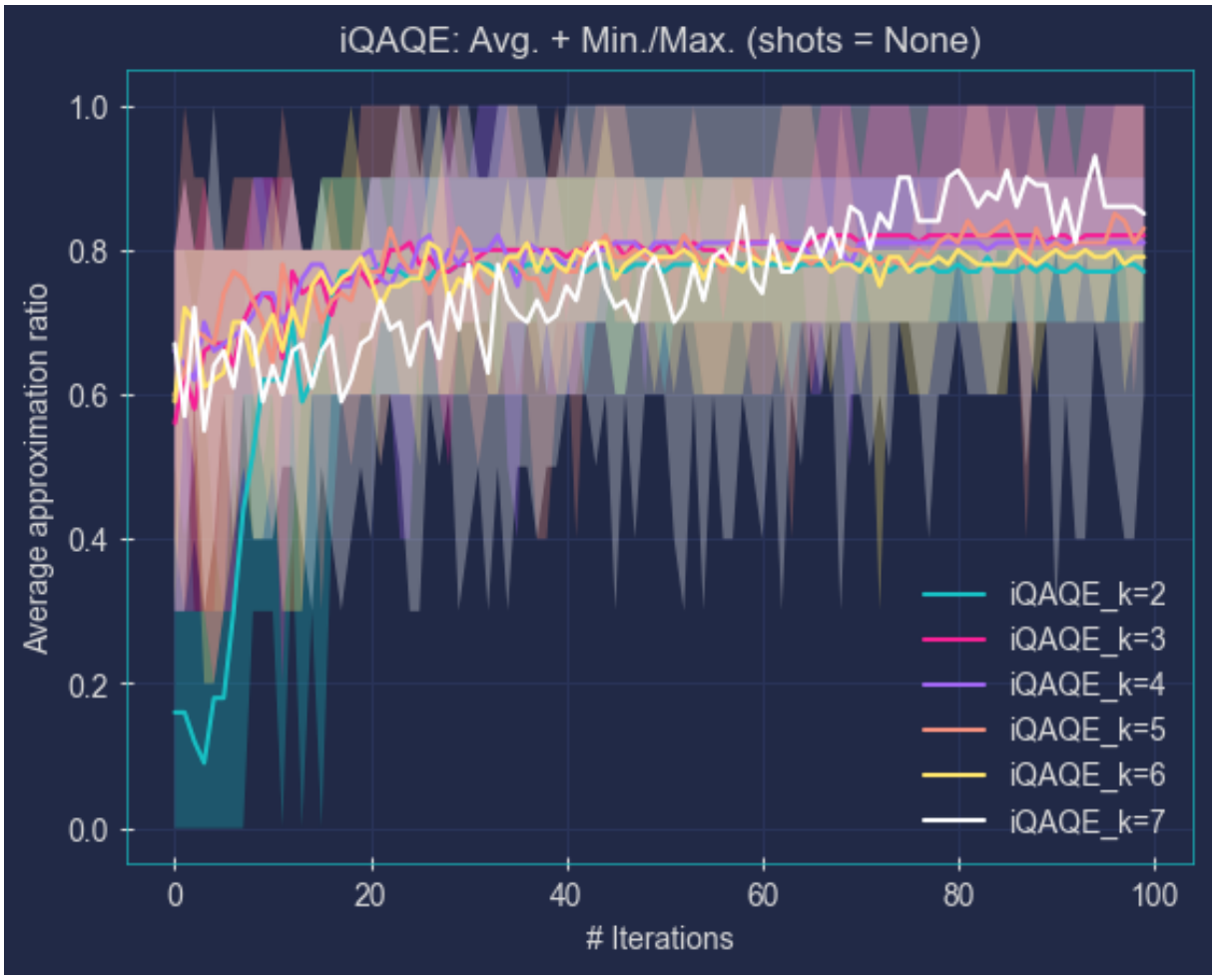


Figure 5.9: Correlation-based polynomial-type compression scheme ($k = 2, 3, 4, 5, 6$ and 7): numerical simulations for the usual 8-node graph.

Note that we once again use the average approximation ratio from 10 runs, each with different random initial parameters. Additionally, we plot the average, with the shaded region representing the maximum and minimum obtained approximation ratios. Performance-wise, this scheme does not fare well. It is significantly outperformed by QAOA, QEMC, and the basic polynomial compression-type iQAQE. We also compared the results for all possible values of k (Figure 5.9). It turns out that the specific value of k is not very meaningful, as performance is quite similar across all values.

The underperformance is likely due to our doubling the number of basis states in each sublist. This encoding results in each pair of sublists having a 50% overlap, similar to the Basic Polynomial Compression-type iQAQE. However, with twice[4] the number of basis states this time, the number of overlapping basis states is significantly higher. This increase means that more states are shared by multiple nodes. When the overlap is too large, it becomes more challenging to adjust the color of one node without affecting the others. We believe this to be the main reason for the underperformance. We will be mindful of this in the design of future schemes.

[4]Either 0's or 1's can be fixed.

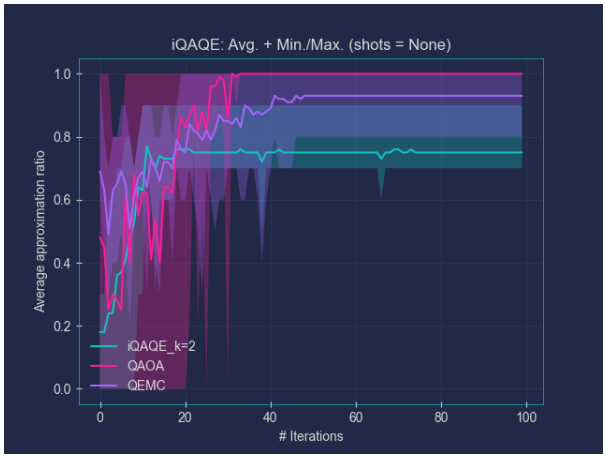
5.2.3 Fixed-Parity iQAQE

In this section, we present another heuristic method for mapping the basis states to the nodes. Although it is similar to the previous one, this time we **fixed the parity of the selected k qubits to be even**. Parity is determined by the number of 1's: if the count is even, the parity is even; otherwise, it is odd. For instance, for $k = 3$ (with `n_qubits = 5`), the lists would take the form: (Keep in mind that we are using an 8-node graph.)

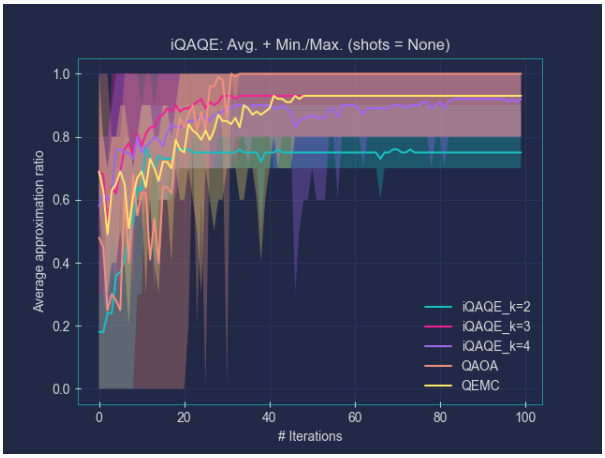
1. $\{|000\text{xx}\rangle, |011\text{xx}\rangle, |101\text{xx}\rangle, |110\text{xx}\rangle\}$;
2. $\{|00\text{xx}0\rangle, |01\text{xx}1\rangle, |10\text{xx}1\rangle, |11\text{xx}0\rangle\}$;
3. $\{|0\text{xx}00\rangle, |0\text{xx}11\rangle, |1\text{xx}01\rangle, |1\text{xx}10\rangle\}$;
4. $\{|\text{xx}000\rangle, |\text{xx}011\rangle, |\text{xx}101\rangle, |\text{xx}110\rangle\}$;
5. $\{|\text{x}0\text{x}00\rangle, |\text{x}0\text{x}11\rangle, |\text{x}1\text{x}01\rangle, |\text{x}1\text{x}10\rangle\}$;
6. $\{|\text{x}00\text{x}0\rangle, |\text{x}01\text{x}1\rangle, |\text{x}10\text{x}1\rangle, |\text{x}11\text{x}0\rangle\}$;
7. $\{|\text{x}000\text{x}\rangle, |\text{x}011\text{x}\rangle, |\text{x}101\text{x}\rangle, |\text{x}110\text{x}\rangle\}$;
8. $\{|0\text{x}0\text{x}0\rangle, |0\text{x}1\text{x}1\rangle, |1\text{x}0\text{x}1\rangle, |1\text{x}1\text{x}0\rangle\}$.

Once again, numerical simulations were performed, and the obtained results are presented below (Figure 5.10). Although the performance is still not quite on par with QAOA, it is significantly better than in the previously considered scenario. Note that the case where $k = 2$ corresponds to the previous correlation-based scheme. For $k = 2$, requiring the pair to be even is equivalent to requiring them to be the same (either both 0 or both 1), which explains the similarity in results. This similarity is apparent in the less favorable outcomes shown for $k = 2$ in Figure 5.10, which qualitatively match those in Figures 5.8 and 5.9. However, when we allow for $k > 2$, the performance improves significantly, likely due to the reduced overlap between the basis states associated with different nodes.

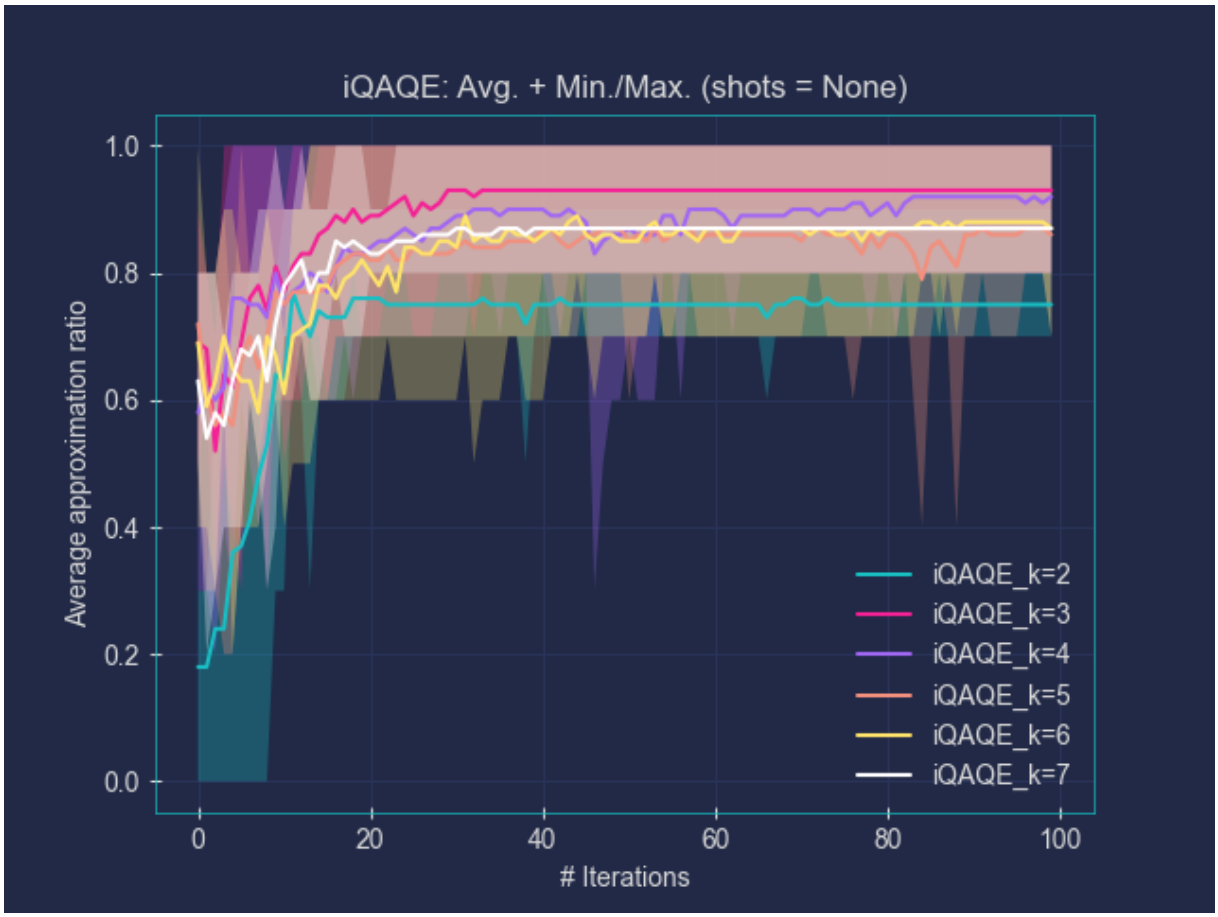
A keen observer might notice that this scheme's results also allow for generating the MaxCut partition, as shown by the shaded pink region in Figure 5.10b for $k = 3$. These shaded regions depict the maximum and minimum cuts achieved out of the 10 runs. This reveals that, despite the greater variability and lower average performance than QAOA, the algorithm can achieve the MaxCut partition in some runs. In practice, this is crucial. When running the algorithm 10 times, our main interest is in the best outcome. This highlights the potential of this heuristic scheme.



(a) Fixed-parity iQAQE for $k = 2$ compared with the results from QAOA and QEMC (8-node graph).



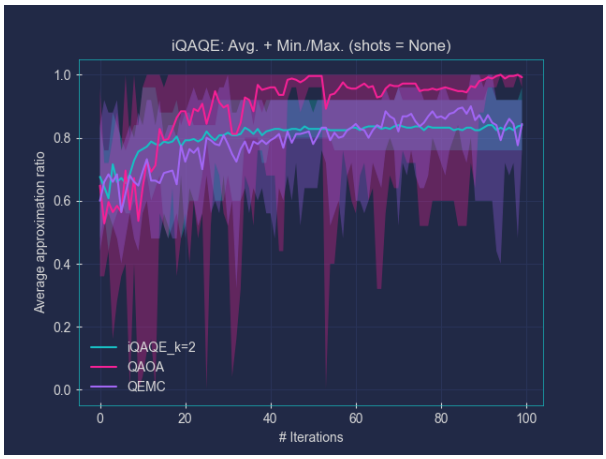
(b) Fixed-parity iQAQE for $k = 2, 3$, and 4 compared with the results from QAOA and QEMC (8-node graph).



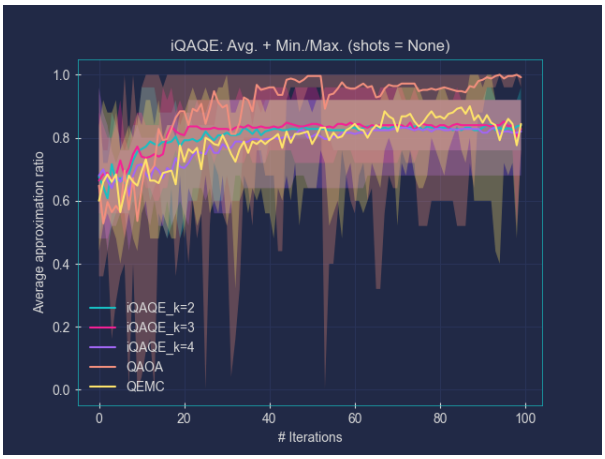
(c) Fixed-parity iQAQE for all values of $k = 2, 3, 4, 5, 6$, and 7 (8-node graph).

Figure 5.10: Fixed-parity polynomial-type compression scheme: numerical simulations for the usual 8-node graph.

Furthermore, numerical simulations were carried out on a random 16-node graph[5]. Considering the satisfactory performance observed with the previous 8-node graph, we were intrigued to assess the algorithm's performance on a larger scale. The results are now presented (Figure 5.11).



(a) Fixed-parity iQAQE for $k = 2$ compared with the results from QAOA and QEMC (16-node graph).



(b) Fixed-parity iQAQE for $k = 2, 3$, and 4 compared with the results from QAOA and QEMC (16-node graph).

[5]This graph was generated using NetworkX's `nx.gnm_random_graph(n, m)` with `n = 16` nodes and `m = 32` edges, using `np.random.seed` set to 44.

(c) Fixed-parity iQAQE for all values of $k = 2, 3, 4, 5, 6$, and 7 (16-node graph).

Figure 5.11: Fixed-parity polynomial-type compression scheme: numerical simulations for a 16-node graph.

The previously identified trends are also apparent here. However, the performance now seems insufficient to achieve the MaxCut partition (determined by brute-force), as illustrated by the shaded areas in Figure 5.11c: the average AR never quite reaches 1. Moreover, for this graph, varying the value of k does not significantly impact performance. Hence, it is advisable to choose the value of k that requires the fewest qubits. In this instance, $k = 3$ is optimal, needing only `n_qubits = 6`, since $\binom{6}{3} = 20 \geq 16$.

5.3 Extended-QEMC

Amid the various schemes we've attempted, I've devised an extension to the standard QEMC scheme. This new approach enhances QEMC by incorporating m additional qubits beyond the usual $\lceil \log_2(n) \rceil$ qubits required for n graph nodes.

5.3.1 Unmodified Extended-QEMC

Rather than associating a single basis state with each graph node, it assigns 2^m basis states per node. Importantly, similar to QEMC, the sets of basis states associated with different nodes do not over-

lap. This non-overlapping property is crucial, as we suspect that significant overlap between nodes' lists might contribute to the subpar performance observed in the correlation-based and fixed-parity iQAQEs.

This extended scheme can be implemented within the iQAQE formalism by using appropriate sub-lists of basis states. The assignment of states to each list is currently arbitrary; for simplicity, we use a straightforward partition: the first 2^m basis states go to the first list, the next 2^m to the second list, and so on (referred to as Unmodified Extended-QEMC). However, there may be a more optimal method for distributing these states that we have not explored here.

Additionally, it is important to note that Extended-QEMC requires more qubits than regular QEMC. This increase in qubits prevents the scheme from being easily classically simulable and allows for more basis states to be associated with each graph node. Consequently, this should provide more variables to manipulate, potentially improving our chances of finding a good solution.

Now, I present the results of applying this scheme to the usual 8-node graph (Figure 5.12).

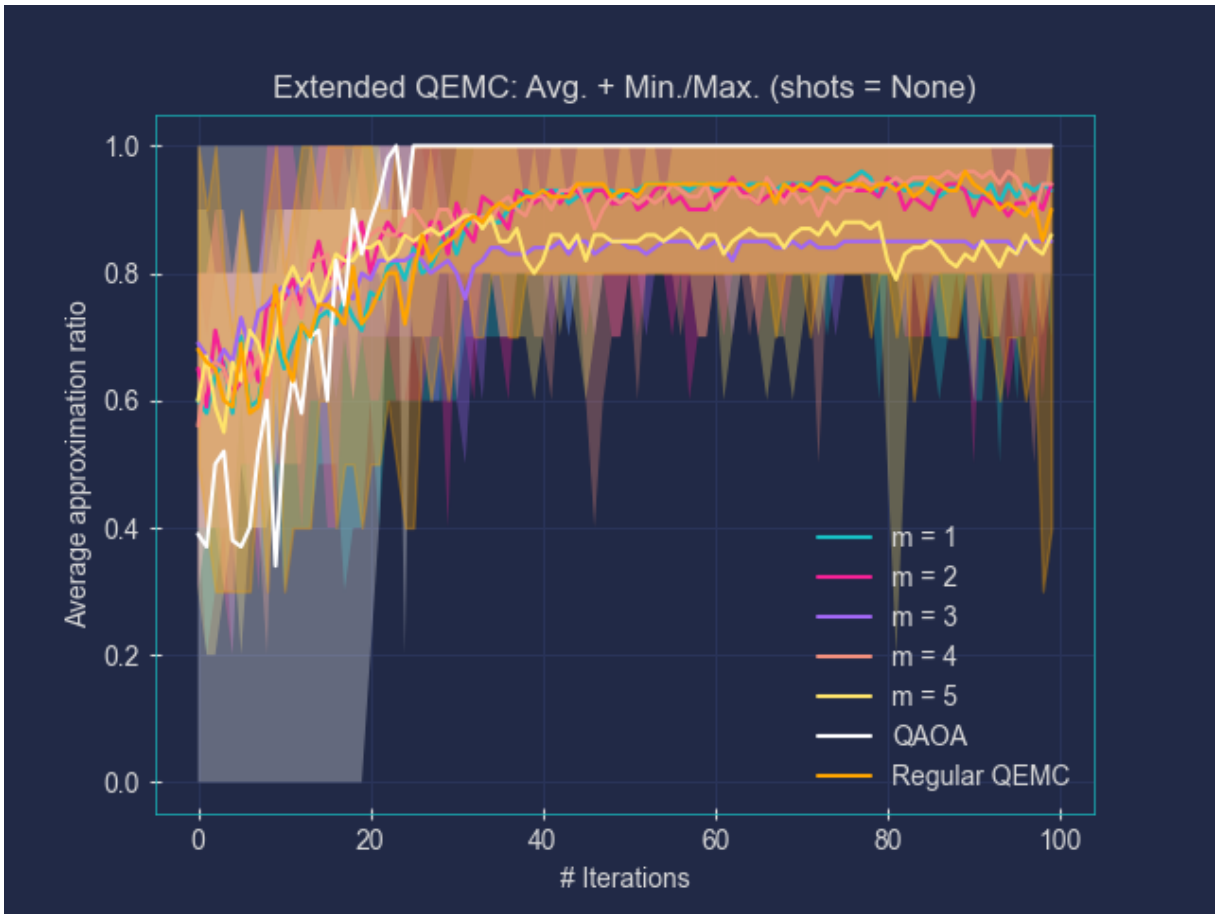


Figure 5.12: Implementation of the Unmodified Extended-QEMC scheme for the usual 8-node graph: comparison with QAOA and regular QEMC for various values of m. Note that m is not extended beyond 5, as this would exceed the number of qubits used in QAOA

As shown in the previous figure, the performance of the unmodified Extended-QEMC scheme is qualitatively very similar to that of regular QEMC. For some values of m, we achieve slightly better results, but these improvements are not particularly significant. After conducting several tests, I found no clear pattern to predict which values of m would yield the best results, supporting the earlier conclusion.

Therefore, this presents another heuristic for mapping the basis states, with performance comparable to the standard QEMC scheme. While this is a satisfactory outcome, it is not particularly remarkable.

5.3.2 Cardinality $= 1$ Extended-QEMC

Motivated by the results in Figure 5.17a[6], I decided to make a slight change to the Unmodified Extended-QEMC scheme: instead of assigning 2^m basis states to each node, we assign only one basis state to each node (Cardinality $= 1$ Extended-QEMC). Although this adjustment occasionally yielded marginally better results than regular QEMC, we do not consider these improvements significant (see Figure 5.13 for reference).

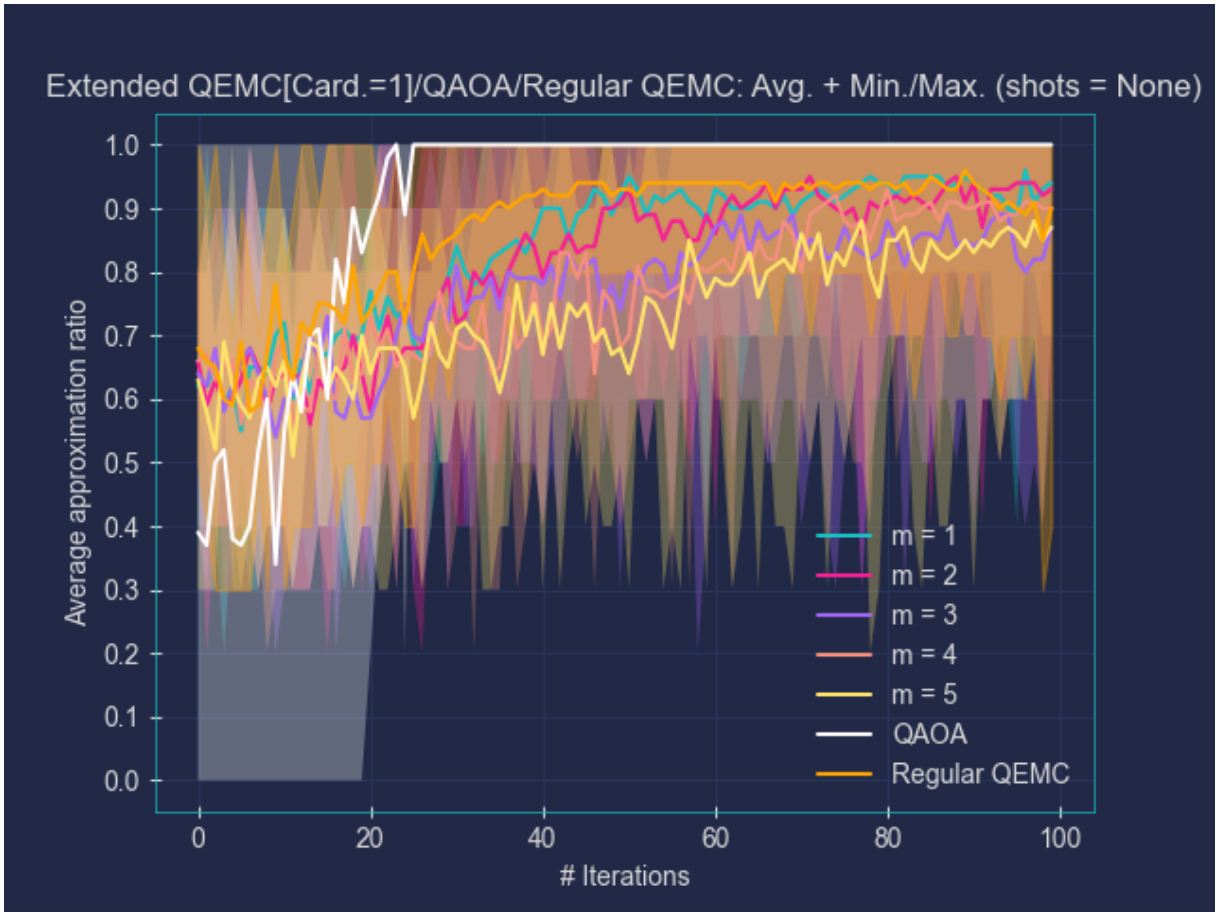


Figure 5.13: Implementation of the Cardinality $= 1$ Extended-QEMC scheme for the standard 8-node graph: comparison with QAOA and regular QEMC for various values of m.

5.4 Alternative Ansätze

After some reflection, I began to question whether the performance of QEMC and iQAQE could be hindered by the use of a problem-agnostic ansatz. It's generally advantageous to incorporate problem-specific information into the ansatz, as exemplified by problem-inspired ansätze. Additionally, I pondered

[6]The trajectory of this study was far from linear. The results presented here are not in the chronological order of our research, hence this reference to later work, which was actually conducted before devising the Extended-QEMC scheme.

whether excessive entanglement or correlations among the system's basis states might be affecting the ansatz's performance. Such phenomena could make it challenging for the model to adjust the amplitude of a specific basis state without significantly impacting others. To address this potential issue, the concept of implementing non-deterministic CNOT gates (ND-CNOTs) occurred to me. This approach would allow the model to dynamically adjust the degree of entanglement. The most rudimentary way to implement this is by employing parameterized R_x gates before each CNOT's control qubit, which is precisely what we did. The R_x gate allows the control qubit to rotate in and out of the $|1\rangle$ state, thus adjusting the degree of entanglement introduced by the CNOT gate. The results for the standard 8-node graph are presented below (Figure 5.14), for both QEMC and Cardinality $= 1$ Extended-QEMC.

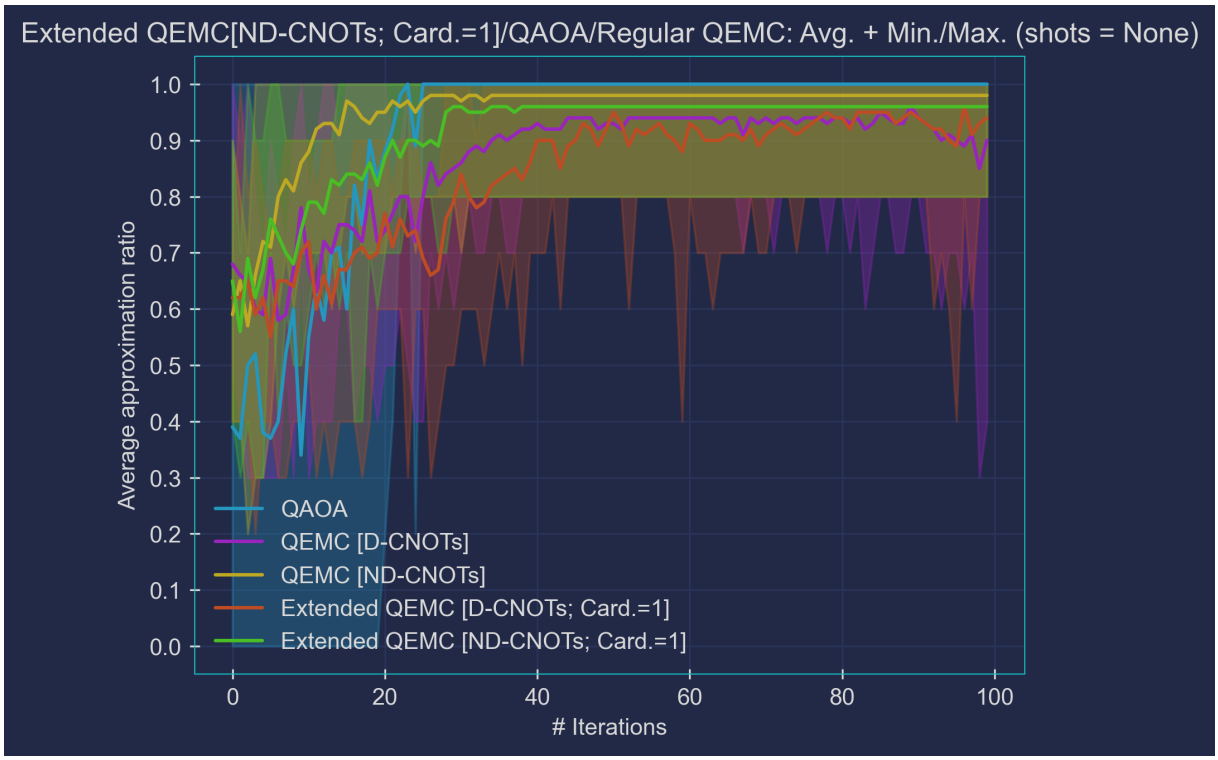


Figure 5.14: Comparison of QEMC and Extended-QEMC implementations utilizing both D-CNOTs and ND-CNOTs for the usual 8-node graph, alongside QAOA.

Several noteworthy observations can be made here. Firstly, it's apparent that ND-CNOT-based schemes exhibit significantly more stable performance with fewer fluctuations compared to D-CNOT-based schemes. This discrepancy arises from differences in their implementations: D-CNOT circuits employ `n_layers = 3` and a `step_size = 0.95` (Adam's learning rate), whereas ND-CNOT circuits use `n_layers = 2` and `step_size = 0.5`. The latter, `step_size = 0.5`, accounts for the smoother curves observed in Figure 5.14, as expected, and may also contribute to faster convergence. Secondly, the performance appears noticeably enhanced when utilizing ND-CNOTs, particularly evident in the case of regular QEMC: a distinct improvement is evident when comparing the yellow and purple lines. Despite introducing added complexity to the optimization landscape, this approach appears to yield significant performance gains, at least for this small 8-node graph. Thus, we've proposed a novel scheme that builds upon QEMC, showing great potential to outperform it.

Moreover, this analysis serves to validate the notion that deviating from purely problem-agnostic Ansätze is generally beneficial. This insight will inform our approach in developing future schemes in subsequent research endeavors. Encouraged by the promising outcomes from this study, we opted to expand the testing of this scheme to larger graphs to assess its scalability. Additionally, this paves the way for comparisons with the Goemans-Williamson scheme.

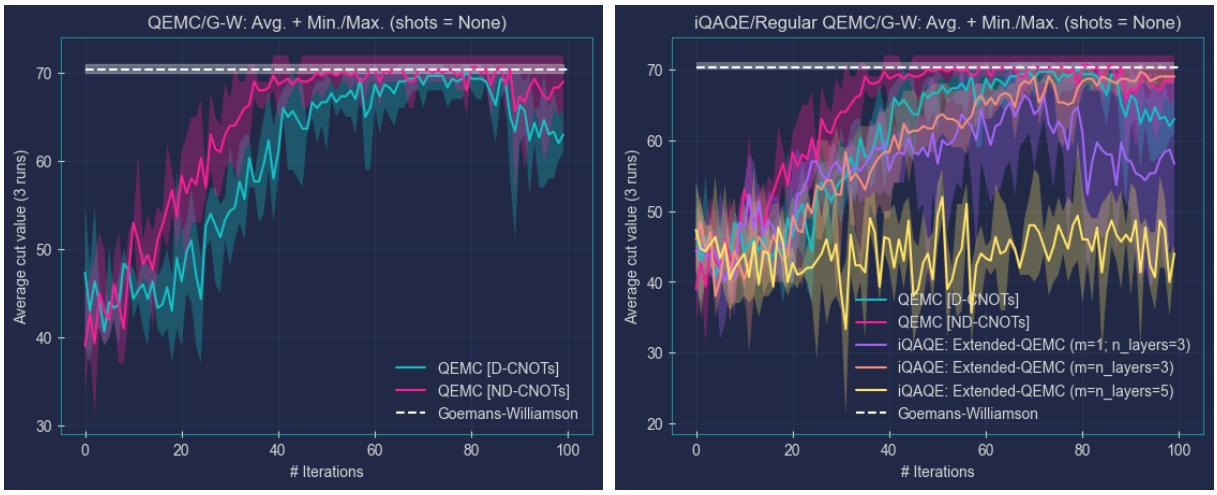
5.5 Goemans-Williamson and Bigger Graphs

At one point, I began to question whether it was appropriate to directly compare our results to QAOA, which stands out as the best-performing VQA among those tested. After thorough deliberation, we concluded that, indeed, it makes sense to do so for smaller graphs. However, as graphs increase in size, the availability of quantum machines with a necessary number of qubits to support such QAOAs becomes limited. This circumstance has led to the emergence of schemes like QEMC [3] and others [18], which aim to address the limitation imposed by the number of qubits.

In light of this, I proceeded to evaluate the performance of our proposed schemes on significantly larger graphs. Two such graphs were considered:

1. 32-node Erdős–Rényi graph, constructed using NetworkX's `nx.erdos_renyi_graph(n=32,p=0.2, seed=0,directed=False)`, where each edge is included in the graph with a probability of $p = 0.2$;
2. 100-node Erdős–Rényi graph, defined similarly to the previous graph, utilizing `nx.erdos_renyi_graph(n=100,p=0.2,seed=0,directed=False)`.

The results obtained from these evaluations are presented below[7] (Figures 5.15 and 5.16).

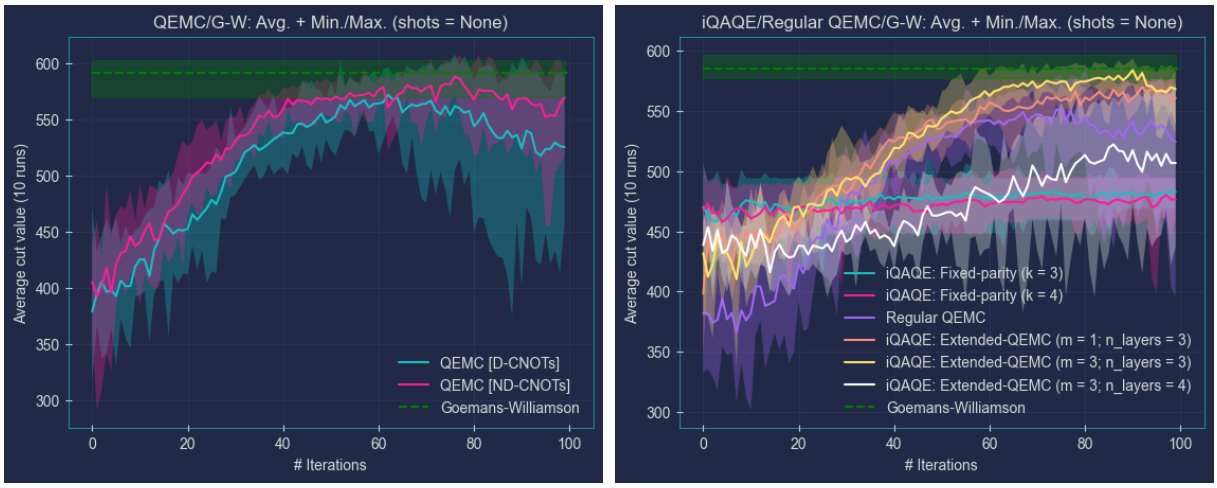


(a) 32-node graph – QEMC and GW exclusively.

(b) 32-node graph – other schemes.

Figure 5.15: Comparison of performance (average cut values) among various VQA schemes for the specified 32-node graph. `n_layers = 3` was employed for both QEMCs, with `step_size = 0.95` for D-CNOT-QEMC and `step_size = 0.5` for ND-CNOT-QEMC. For all Extended-QEMC schemes, a `step_size = 0.95` was utilized, with the number of layers specified in the legend.

[7] Please note that we have included only the most promising schemes.



(a) 100-node graph – QEMC and GW exclusively.

(b) 100-node graph – other schemes.

Figure 5.16: Comparison of performance (average cut values) among various VQA schemes for the specified 100-node graph. This time, all considered VQAs utilized `n_layers = 3` and `step_size = 0.95`, except for Extended-QEMC with $m = 3$, which used `n_layers = 4`.

In Figure 5.16, note that QEMC, fixed-parity iQAQE, and GW employ 10 runs for averaging, while all other schemes (including Cardinality $= 1$ Extended-QEMC) use only 5 runs. This decision was made to expedite the process while ensuring reasonable averaging of statistical fluctuations in the results. Likewise, in Figure 5.15, only 3 runs were utilized for all schemes. Additionally, in Figure 5.16, I chose to include the fixed-parity iQAQE scheme to evaluate its performance on a larger graph.

Now, concerning the results, several interesting observations emerge. For the 32-node graph, Extended-QEMC's performance varies considerably with different values of m and `n_layers`, as previously observed, making it inconsistent and challenging to optimize for other graphs. Once again, we observe the benefits of using ND-CNOTs in QEMC compared to the usual D-CNOTs. Although the average performances have not yet reached the level of the Goemans-Williamson algorithm, indicated by the white dashed line in Figure 5.15b, the maximum performances achieved by the ND-CNOT-based scheme surpass the best performance attained by GW, as evident from the comparison of the shaded pink and white regions. This marks the first instance in this study where one of our heuristics surpasses classical state-of-the-art algorithms, representing a significant milestone and underscoring the potential for further exploration of the proposed iQAQE Framework.

Turning to the 100-node graph, similar trends are observed across the board. Additionally, we observed that the previously mentioned fixed-parity iQAQE performs exceptionally poorly, indicating its inability to generalize for larger graphs. Nevertheless, as with the 32-node graph, reaching the performance level of the Goemans-Williamson algorithm on average remains elusive, indicating ample room for improvement. However, once again, promising results are seen in the maximum cuts (shaded regions).

5.6 Average Best-so-Far correction

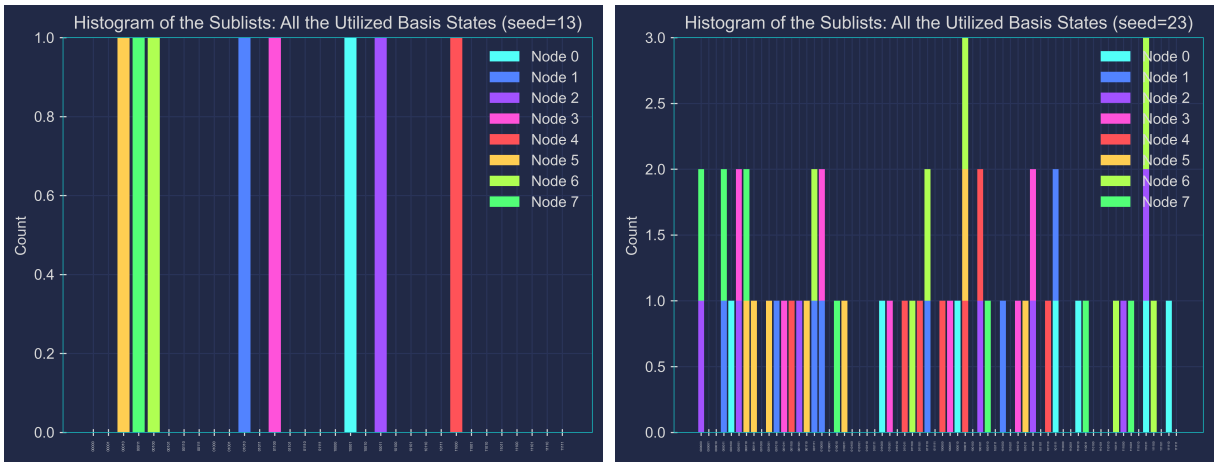
As previously noted, there exists a distinction between implementing the BSF transformation before and after the averaging process. We assert that the former (prior to averaging) provides the most accurate representation of the algorithm's optimal performance. Therefore, in alignment with the average AR plots previously showcased, we opted to rerun several earlier schemes with the BSF transformation applied before averaging. Given the substantial number of resulting graphs, they will be included in Appendix **??**.

5.7 Randomized iQAQE benchmarking

In this section, our aim is to adopt a more systematic approach to identify ways to enhance the mapping of basis states. While heuristics have served us thus far, we sought to determine whether there are additional patterns that we may have overlooked by relying solely on these heuristics.

Initial proposition

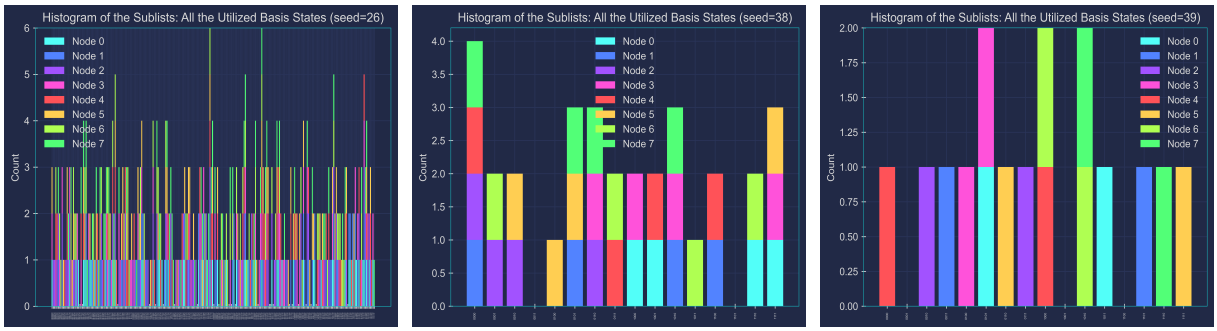
Continuing from the earlier discussion, we ran numerous iQAQE instances, each with random qubit number, list cardinalities and assignments[8]. The underlying idea was to identify optimal combinations of qubit number, list cardinalities and mapping by running multiple instances of iQAQE. Subsequently, we aimed to discern meaningful patterns that could inform the development of a strategy for more refined basis state selection. However, our analysis did not yield a clear indication of how the partitioning should proceed. As demonstrated below, there existed a plethora of vastly different combinations of partitioning and list cardinality that performed similarly well. Consequently, it proved humanly impossible to extract any coherent insights from this dataset. The results supporting this claim are presented in Figure 5.17 for the usual 8-node graph.



(a) Seed = 13.

(b) Seed = 23.

[8]We always employed `n_layers = 3` and `step_size = 0.99`. This time, the final selected cut is derived from the BSF average curve (not the average BSF), constructed from 10 repetitions. Although not optimal, this does not alter our interpretation of the results.



(c) Seed = 26. (d) Seed = 38. (e) Seed = 39.

Figure 5.17: Basis states' distributions over the 8 nodes of the usual graph, for many different `np.random.seed` seeds. Only those for which the average best-so-far approximation ratio [after 100 iQAQE iterations] was found to be ≥ 0.975 were selected. (These were the best performing seeds out of the 50 tested, for $\texttt{seed} \in [0, 49] \cap \mathbb{Z}$.)

As can be seen, not much can be directly inferred from this. When confronted with seemingly unstructured data, a natural progression is to employ machine learning schemes to discern patterns. Indeed, this aligns with the forte of such models. Various options were considered, spanning from neural networks to regression and clustering models. Initially, we opted to implement a basic regression model for its simplicity. Establishing such a model necessitates identifying a set of statistical properties of the input data (lists of basis states) that can serve as inputs for the regression model. This discussion follows in the subsequent section.

Machine learning-based approach

Following the prior discussion regarding the iQAQE formalism (refer to Figure 5.17), our objective was to construct a small machine learning (ML) model aimed at comprehending the patterns in basis states' partitioning that contribute to the algorithm's performance variations. Naturally, such a model necessitates defining input (independent variables - IV) and output (dependent variable - DV). As previously mentioned, our IVs will encompass statistically significant properties extracted from each specific sub-list encoding. Alternatively, including the basis states' partitioning directly as the model's input presents challenges concerning memory-efficient encoding. Traditional one-hot encoding methods are inadequate due to the vast number of potential basis states available (which scales exponentially with the number of qubits). Thus, we opted for the following statistical properties:

- **Type 1 variables** - these are input variables that are defined for each of the sub-lists of basis states. They are:
  - **Number of basis states** in each sub-list, the so-called cardinality of the sub-list;
  - **Average Hamming weight** in each sub-list;
  - **Standard deviation/variance of the Hamming weights** in each sub-list;
  - **Average pair-wise Hamming distance** between the basis states in each sub-list;
  - **Standard deviation/variance of the pair-wise Hamming distances** between the basis states in each sub-list;

  - **Parity score** of each sub-list: this is obtained by: #(Even basis states) $-$ #(Odd basis states), for each sub-list.

- **Type 2 variables** - these are input variables that are defined for the entire set of basis states. They are:
  - **Average Hamming weight** of the entire set of basis states;
  - **Standard deviation/variance of the Hamming weights** of the entire set of basis states;
  - **Average pair-wise Hamming distance** between the basis states in the entire set;
  - **Standard deviation/variance of the pair-wise Hamming distances** between the basis states in the entire set;
  - **Parity score** of the entire set of basis states. (Given by the sum of the parity scores of the sub-lists);
  - **Global basis-states' entropy** [of the entire set of basis states]: this is given by the Shannon entropy of the basis states' distribution in the entire set. It is calculated as: $-\frac{1}{\log_2(N)} \sum_{i=1}^{N} p_i \log_2(p_i)$, where p_i is the probability of the i^{th} basis state in the entire set. Note that it is normalized by the maximum entropy value, which is $\log_2(N)$, where N is the total number of basis states in the entire set. (This entropy's maximum is achieved when we have a uniform distribution of the basis states.) Also, this is a measure of the randomness of the basis states' distribution in the entire set.

The dataset was constructed in a similar manner to that of Figure 5.17, but expanded to encompass 1000 seeds. For each seed, we computed the **median** BSF vector (DV) and the associated independent variables (IVs) related to the partitioning of basis states. (It's worth noting that we employed `n_layers = 3`, `step_size = 0.99`, `max_iter = 50`, and `repeat = 5`.)

With the IVs and DV finally chosen, and the data set fully constructed, we now need to define our model. We decided to start simple, with a basic multiple linear regression model. Here, the DV is computed as a linear combination of the IVs, as below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon, \tag{5.1}$$

where x_i are the input variables (IVs), y is the dependent variable (DV), β_i are the coefficients, and ϵ is the error term. The coefficients will be estimated using the Ordinary Least Squares (OLS) method. The model's goodness-of-fit will be evaluated using the R^2 value. Additionally, it's worth noting that the independent variables were initially scaled using the `StandardScaler` from the `sklearn.preprocessing` module, a common practice in machine learning. This process standardizes the features by removing their mean and scaling them to unit variance, ensuring they have equal weight in the model. This standardization provides a stable foundation for both multiple linear regression and Principal Component Analysis (PCA).

After implementing the model, we obtained an R^2 of 0.1023, which is far from the ideal value of 1, and a 5-fold cross-validation loss of 0.0061 (average mean square error) with a standard deviation of 0.0004.

To improve the model, I analyzed the independent variables (IVs) by examining their correlation matrix and Variance Inflation Factor (VIF). The correlation matrix revealed that some IVs were highly correlated, which was expected in hindsight. High correlation among IVs is problematic because it makes it difficult to isolate the effects of each IV on the dependent variable (DV), as altering one IV affects others.

To quantify this issue, we calculated the VIF for each IV. The VIF measures the increase in the variance of parameter estimates if an additional variable is added to the linear regression, indicating the level of multicollinearity among the IVs. If the VIF is greater than 5 or 10 (depending on the user's tolerance), it suggests high multicollinearity, and the variable should probably be removed from the model. Our analysis showed that many input variables were perfectly multicollinear, meaning some were essentially redundant. I experimented with removing these redundant variables one by one to improve model performance. While this approach did lead to some improvement, it was never significant. I tried various combinations, such as removing only Type 2 variables, only Type 1 variables, or a mix of both, but none resulted in substantial performance enhancements. The predictive power of the model was also unsatisfactory. Although it could estimate the order of magnitude correctly, it was often off by more than 0.1 in approximation ratio (AR), which is unacceptable. If we could develop a model that accurately predicts the (AR) based on certain IVs, we could reverse-engineer the sub-lists that lead to the best-performing median BSF approximation ratios. This would be a significant step towards our ultimate goal of identifying effective partitioning schemes for the basis states.

In any case, this poor performance could indicate two things: either the model is inadequate (i.e., the underlying relationship is not actually linear), or the chosen IVs are not the most suitable. We suspect the latter might be true, although we're unsure which other variables to use. This circles back to the initial challenge of encoding the sub-lists' information in a memory-efficient and representative manner. Regarding the potential non-linearity of the model, we've considered using non-linear models, such as Support Vector Machine Regression (SVR), which exploits the kernel trick to fit non-linear data. However, we have set this idea aside for now[9], as our current goal is to identify which IVs most influence the DV's value.

One alternative approach that came to mind is to perform Principal Component Analysis (PCA) followed by regression (PCR). While PCA assumes a linear relationship, it could help us achieve a better model. By computing the Principal Components (PCs), we can indirectly understand which IVs have the greatest influence in explaining the variance of the input data. These influential IVs might be the ones we should focus on when choosing our basis states' partitioning. By examining the loadings of the PCs, we can determine which IVs constitute each PC. Another advantage of PCA is that the obtained PCs are orthogonal (uncorrelated), thus eliminating the multicollinearity problem among the input variables. We can then attempt linear fitting on the PCs against the DV (PCR). The main drawback is that we lose some interpretability of the input variables, as accurately interpreting the PCs can be challenging due to their nature as linear combinations of the original IVs.

Anyways, PCR using the 5 most significant principal components also yielded subpar results. An R^2 of 0.0287 was reported, with a 5-fold cross-validation loss of 0.0059 and a standard deviation of 0.0004.

[9] This would be an interesting direction for future work.

As one could already predict from the R^2 value, this model's predictive powers were also unsatisfactory. Nevertheless, we can still try to extract some information from the principal components to understand which independent variables are most influential in determining the dependent variable. Notably, the first 5 principal components (out of 47) account for 65.3% of the variance in the data:

```
Explained variance ratio:
[0.46558461 0.07637042 0.04557191 0.03879498 0.02634421]
```

Listing 5.1: Explained variance ratio, for the first 5 PCs (8-node graph).

Now, if we examine the loadings of the first and most significant principal component, we notice they are quite evenly distributed across the independent variables. This makes interpretation challenging, as we cannot easily identify one IV as being more important than the others. For instance, here are the loadings of the first principal component for the 8-node graph:

```
PCA_1 = [-1.75323157e-01 -9.26451207e-02 -8.98377538e-02 -8.77605361e-02 -9.18316965e-02
         -8.58000439e-02 -1.03221966e-01 -9.24948614e-02 -1.11659776e-01 -1.75791034e-01
         -1.73978061e-01 -1.70204027e-01 -1.70707746e-01 -1.73770114e-01 -1.72400120e-01
         -1.73386482e-01 -1.71136657e-01 -1.88767128e-01 -1.82462202e-01 -1.81233077e-01
         -1.81277591e-01 -1.83509589e-01 -1.80911686e-01 -1.83449672e-01 -1.85536765e-01
         -1.87937749e-01 -1.85009049e-01 -1.87993729e-01 -1.86774034e-01 -1.87581036e-01
         -1.85531361e-01 -1.86124394e-01 -1.86528612e-01 -2.09742228e-03 -3.66652955e-03
          9.18121507e-04 -9.31248048e-03 -4.63173882e-03 -1.18673071e-03  1.43514268e-02
         -1.14790537e-02 -1.48687943e-01 -1.33654722e-01 -1.54547996e-01 -1.55191025e-01
         -5.69361380e-03 -1.50033651e-01]
```

Listing 5.2: Loadings of the first PC, for the 8-node graph.

As can be seen, there are no IVs whose impact can be neglected, making the interpretation of this principal component nearly impossible, which doesn't help our case much. Ideally, we would discover that a small number of independent variables have a significantly larger impact than the others. This would enable us to concentrate on those variables when selecting our basis states' partitioning. Unfortunately, this is not the situation here. For the sake of completeness and organization, I present the obtained results in the following table (Table 5.1).

Model (8-node graph)	5-fold cross-validation loss		R^2
	Average ($\overline{x}$)	Std. dev. (σ)	
Multiple Linear Regression	0.0061	0.0004	0.1023
PCR (5 Principal Components)	0.0059	0.0004	0.0287

Table 5.1: Cross-validation loss (average mean square error) and R^2 values for the two fitted models, Multiple Linear Regression and PCR, in the context of the usual 8-node graph.

Furthermore, after revisiting the regression analyses' coefficients, we noticed something curious: when not performing PCA before fitting, we observed significantly higher coefficients for four specific input variables (on the order of magnitude 10^9 or 10^{10}, compared to $[10^{-4}, 10^{-2}]$ for all other variables). These IVs are (for the 8-node graph dataset):

1. Avg. Hamming weight (Type 1);

2. Parity Score (Type 1);

3. Avg. Hamming weight (Type 2);

4. Parity Score (Type 2).

Initially, it appeared that these variables contributed the most to the dependent variable's value (median best-so-far approximation ratio). However, we no longer believe that to be the case. Instead, we attribute this observation to high multicollinearity among the independent variables (IVs), which renders the model unstable.

We also tested Ridge regression ($L2$ regularization) as an alternative to PCR. Ridge regression is useful when the input data has obvious multicollinearity, as it adds a penalty to the regression coefficients, reducing their variance and stabilizing the model. Using `alpha = 1.0` in `sklearn.linear_model.Ridge`, we obtained an R^2 value of 0.1023 and an average cross-validation loss of 0.0061, with a standard deviation of 0.0004. These results were identical to those from multiple linear regression (for the 8-node graph). However, with Ridge regression, no IVs had disproportionately large coefficients; they all ranged between 10^{-4} and 10^{-2}. This regularization added stability to the model by preventing coefficient divergence. Moreover, its predictive power appeared to be slightly better than that of regular linear regression for our current dataset, often resulting in errors smaller than 0.1 in approximation ratio (AR).

This analysis reinforced our belief that the previous 10^{10}-valued coefficients were due to model instabilities from high multicollinearity. Thus, Ridge regression offers a better model with more meaningful and interpretable coefficients. However, the uniform spread of coefficients makes it difficult to pinpoint a few IVs as the most impactful, which we had hoped to identify.

We have explored several techniques to find patterns or strategies for better mapping. However, the task is more complex than initially anticipated. Capturing all intricate patterns might require a more complex model, such as a neural network or another non-linear approach. We have decided to leave this direction for future work. Another concern is that iQAQE's performance may depend heavily on the specific graph instance, complicating the generalization of a model for predicting optimal mapping strategies. Despite this, we believe it is still worth pursuing.

Chapter 6

Exploratory Ideas

Insert your chapter material here.

6.1 Parity-like QAOA

Introduce the concept of parity-like QAOA.

6.2 Tranche-based Oracle Colouring

Introduce the concept of tranche-based oracle colouring.

Chapter 7

Conclusions

Insert your chapter material here.

7.1 Main Findings

The major achievements of the present work.

7.2 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.