Flexible integration of classical and quantum techniques in the evolutionary path to quantum utility

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

Doctor of Philosophy

in the Department of Computer Science

by

Prashanti Priya Angara

B. Tech in Computer Science and Engineering, GITAM University, India, 2012
M.Sc. in Computer Science, University of Victoria, Canada 2018

We acknowledge and respect the lək^{w}əŋən (Songhees and X^{w}sepsəm /Esquimalt) Peoples on whose territory the university stands, and the lək^{w}əŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Flexible integration of classical and quantum techniques in the evolutionary path to quantum utility

by

Prashanti Priya Angara

B.Tech. in Computer Science and Engineering, GITAM University, India, 2012
M.Sc. in Computer Science, University of Victoria, Canada, 2018

Supervisory Committee:

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

Dr. Ulrike Stege, Supervisor
Department of Computer Science, University of Victoria

Dr. Irina Paci, Outside Member
Department of Chemistry, University of Victoria

Abstract

This dissertation addresses fundamental challenges in solving constrained combinatorial optimization problems using hybrid quantum-classical computing approaches. Quantum computing shows promise for tackling computationally intractable problems, current approaches face significant limitations. The path forward in computing involves combining quantum and classical computing approaches. Quantum processing units (QPUs) will play a key role in accelerating algorithms in optimization, simulation, and machine learning applications beyond what classical computers can achieve on their own, similar to how graphical processing units (GPUs) have become integral to general-purpose computing. The current generation of quantum devices is limited by noise and circuit depth constraints, making it challenging to gain *practical utility*. Hybrid quantum-classical approaches, which combine quantum computing resources with classical computing resources, are essential for addressing these limitations on near-term devices. In the area of combinatorial optimization, current practical state-of-the-art hybrid approaches are variational in nature, relying on parameterized quantum circuits to encode solutions and classical optimization techniques to find optimal parameters. While the ultimate goal of solving computationally intractable problems is to find provably optimal solutions, practical constraints of real-world scenarios often necessitate focusing on efficiently obtaining *high-quality, near-optimal solutions*.

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical approach for tackling these challenging problems that are encoded using quadratic and higher-order unconstrained binary optimization problems (QUBO and HUBO). QAOA alternates between two operators: a problem-specific cost Hamiltonian that encodes the optimization objective, and a mixer Hamiltonian that explores the solution space. The algorithm's parameters are iteratively optimized using classical methods to maximize the probability of sampling high-quality solutions. However, these methods often struggle with solution feasibility, especially for problems where the solutions must satisfy specific constraints. Penalty-based encodings require careful parameter tuning and risk producing infeasible solutions, while feasibility-preserving methods demand deeper quantum circuits that are challenging to implement on near-term quantum devices.

In this dissertation, we present novel strategies that reduce or omit dependency on penalty parameters while maintaining solution quality, effectively capturing both optimal and near-optimal solutions, and balancing the trade-offs between circuit depth and solution feasibility. We present SCOOP, a novel QAOA-based framework for solving *constrained* optimization problems. SCOOP transforms a constrained problem into an unconstrained counterpart, forming *SCOOP problem twins*. QAOA operates on the unconstrained twin to

identify potential optimal and near-optimal solutions. Effective classical post-processing reduces the solution set to the constrained problem space. Our SCOOP approach is solution-enhanced, objective-function-compatible, and scalable.

We demonstrate the effectiveness of the SCOOP framework on selected NP-hard problems, some of which can be encoded quadratically (such as MINIMUM VERTEX COVER and MAXIMUM INDEPENDENT SET), and some that can be encoded using higher-order terms (such as MINIMUM DOMINATING SET, MINIMUM MAXIMAL MATCHING, and MINIMUM SET COVER). We also apply our framework to constrained combinatorial optimization problems that can be solved in polynomial time, such as MINIMUM EDGE COVER and MAXIMUM MATCHING.

Experimental results on full statevector simulators and tensor network simulators show that SCOOP significantly improves the probability of finding high-quality feasible solutions across various problem classes, on problems that can be encoded quadratically as well as problems that can be encoded using higher-order terms. We also demonstrate the effectiveness of SCOOP on real quantum hardware, specifically IBM Quantum Heron R2 processors with 156 qubits, showing that SCOOP can be applied to practical problems with current quantum hardware limitations.

This work makes significant contributions toward achieving quantum utility by enabling systematic investigation of constrained optimization problems on quantum devices. While current quantum computing research primarily focuses on unconstrained problems like MaxCut, real-world applications typically involve constraints. SCOOP provides a scalable framework for encoding constrained problems, extending quantum computing's practical utility beyond simple unconstrained cases. By eliminating dependency on penalty parameters and providing natural problem formulations, SCOOP makes it feasible to explore quantum solutions for constrained problems where classical exact methods become intractable. This capability is crucial for identifying promising avenues for demonstrating quantum utility across diverse constrained optimization problems, advancing the practical applicability of quantum computing approaches.

Table of Contents

List of Figures

List of Tables

Acknowledgments

My academic journey has been a collaborative effort, and I am deeply grateful to all those who have supported me along the way. First and foremost, I would like to thank my supervisors, Prof. Hausi Müller and Prof. Ulrike Stege, whose unwavering support, guidance, and encouragement gave me the confidence to pursue my Ph.D. and have been instrumental in bringing me to where I am today. Your enthusiasm and passion for research have made this journey not just educational but truly enjoyable. I am particularly grateful for the countless engaging discussions that turned complex research problems into exciting puzzles to solve. I am honored to continue learning from you both as mentors and collaborators in our future research endeavors.

I would also like to extend my thanks to the NSERC CREATE program, and MITACS, which has provided me with the workshops, courses, resources and opportunities to pursue my research. The funding and support from these programs have been instrumental in enabling me to focus on my studies and contribute to the field of quantum computing. I am grateful for the chance to be part of such a vibrant and innovative research community.

My internship at Xanadu Inc. was an amazing experience, and I am thankful to Prof. Olivia Di Matteo and the Xanadu team for their mentorship and support. The privilege of working alongside leading experts in quantum computing has been truly inspiring, and their exceptional expertise has profoundly shaped my understanding of the field. I am deeply indebted to Dr. Maxime Dion for his exceptional mentorship during my internship at Institut Quantique in collaboration with Bank of Canada. I developed a deep understanding of QAOA and optimization techniques that became foundational for my doctoral studies.

This dissertation would not have been possible without the contributions of my collaborators. I am grateful to Dr. Danylo Lykov and Dr. Yuri Alexeev for their invaluable expertise in quantum algorithms and optimization techniques, which have been instrumental in shaping the direction of this research. I am grateful to Prof. Faisal Abu-Khzam for his expertise in parameterized complexity. I am thankful to the Quantum Algorithms Institute (QAI), PINQ2, and IBM Quantum for providing quantum hardware and software. The quantum hardware experiments were possible due to the technical expertise of Dr. Sean Wagner, Dr. Ibrahim Shehzad, and Alexandre Choquette from IBM Quantum. I am thankful to the very enthusiastic Emily Martins and Angadh Singh, undergraduate students in our research group who have worked with me on this research journey.

This research has been enriching and enjoyable due to the support of my colleagues and friends at the Rigi and PITA labs. I am grateful to all of you for your camaraderie, insightful

discussions and feedback, and support. I am grateful for the friendships and connections I have made along the way. I would like to express my heartfelt appreciation to my fellow Ph.D. students — Miguel, Felipe, Ivan, and Tristan. Our shared academic journey, collaborative discussions, and mutual support through challenges have enriched my research experience immensely. I will always treasure the time we spent together at conferences, hackathons, and those extended lab discussions where we thought big thoughts. My Rigi 101 — Lorena, Nina, Stefan, and Charlie: I am grateful for the wisdom and support you provided during my initial years. Samantha, Jose, Saasha, Sajjad, Noah, Addie, and Chris: it has been a pleasure to share this journey with you. I am grateful to my lunch buddies — Abhishake, Fei, Nora, Tania, and Nazma — for the many conversations and laughter-filled moments that made every day enjoyable.

I am profoundly grateful to my family, and words cannot express my appreciation for their unwavering support. To my parents, thank you for everything. To my mom, Padmavati Divakarla, who was not just metaphorically but literally my first physics teacher, thank you for challenging me with the toughest problems. You ignited my passion for problem-solving and showed me the joy of discovery. To my dad, A. V. S. Krishna Rao, thank you for showing me the power of quiet curiosity and meticulous attention to detail — traits that have been significant in my approach to research. To my aunt and uncle, Sridevi and Sudhakar Ganti, in Victoria, your constant support and kindness have meant the world to me. To all my family members, thank you for being endlessly fun and inspiring.

Finally, to my husband, Arun, thank you for being my steady companion through it all—always there, always listening, and always knowing without hesitation that I would get here. Your quiet confidence in me, so effortless and matter-of-fact, has been one of my greatest sources of strength.

Dedication

To the memory of my grandmothers Radhabai Angara and Manorama Divakarla. Strong, visionary women and devoted teachers. Their values continue to inspire me every day.

Part I

Motivation and Context

Chapter 1

Introduction

Contents

At the heart of the field of computation is the goal of expanding the limits of what can be computed, through the development of novel algorithmic techniques and the exploration of innovative computational models that address the constraints imposed by computational complexity. One of the most fascinating computational models to gain practical traction over the past decades is quantum computing [Nielsen and Chuang, 2011], an interdisciplinary field that lies at the intersection of mathematics, quantum physics, computer science, and engineering, and finds applications in areas including optimization, machine learning, and simulation of chemical, physical and biological systems. Quantum computers harness quantum mechanical phenomena such as superposition, entanglement, and interference to perform computing. Quantum computing has the potential to help solve problems that so far have no satisfying method solving them, and to provide significant speedup to solutions when compared to their best classical approaches. In turn, quantum computing may allow to solve problems for inputs that so far are deemed practically intractable.

Quantum Advantage refers to the ability of quantum computers to outperform all known classical methods to solve specific computational problems [IBM Quantum, 2023]. However, realizing quantum advantage in practice is a significant challenge as current quantum computers are limited by factors such as noise and decoherence. *Fault-tolerance* is the ability of quantum computers to perform reliable computations despite noise and errors and is essential for achieving quantum advantage but is currently beyond the capabilities of existing quantum hardware [Kim et al., 2023]. *Quantum Utility* refers to the ability to perform reliable computations at a scale beyond brute force classical computing methods that provide exact solutions to computational problems [Kim et al., 2023]. *Quantum Utility* represents an important milestone in the development of quantum computing, where quantum systems demonstrate practical value by performing specific computations more effectively

than classical computers of similar scale. While not as specific as quantum advantage, which aims to definitively outperform all classical methods, quantum utility focuses on achieving tangible benefits in targeted applications. This intermediate goal helps validate the potential of quantum computing while working within current hardware limitations, providing a pragmatic path toward broader quantum advantage.

While there is no expectation to solve all classically intractable problems in polynomial time using quantum computers [Aaronson, 2013], it is worthwhile investigating how to combine quantum routines with classical problem-solving strategies to speed up solutions for such problems and, in turn, enable faster practical implementations. We focus on the classical problem-solving strategies derived from parameterized complexity [Downey et al., 1999c], which provides a framework for analyzing computational problems with respect to multiple input and output parameters. This multi-faceted view of problems may suggest novel approaches for quantum implementations; a problem that appears challenging from one perspective might reveal exploitable structure when viewed through a different parameterization. Such insights guide our development of quantum encoding strategies that capitalize on alternative problem characterizations.

Combinatorial optimization problems are ideal candidates for demonstrating practical quantum utility [Brooks, 2019, Farhi and Harrow, 2016]. Moreover, *constrained* combinatorial optimization problems are of particular interest, as they arise in various real-world applications, including logistics, finance, and machine learning and has been studied extensively in theoretical computer science (cf., [Karp, 1972, Downey and Fellows, 1995, Papadimitriou, 1997, Vazirani, 2001, Papadimitriou and Steiglitz, 1998]). These problems involve finding the best solution from a finite set of feasible solutions while satisfying a specific set of constraints. The challenge lies in effectively encoding these constraints into quantum circuits to ensure that the solutions generated are both feasible and optimal.

The path forward in computing involves combining quantum and classical computing approaches. Quantum Processing Units (QPUs) will play a key role in accelerating algorithms in optimization, simulation, and machine learning applications beyond what classical computers can achieve on their own, similar to how Graphics Processing Units (GPUs) have become integral to general-purpose computing [Nvidia, 2024]. The effective integration of classical and quantum hardware and software architectures will play a crucial role in achieving quantum utility [Kim et al., 2023] and quantum advantage [Shaydulin et al., 2023, Lanes et al., 2025].

In recent years, rapid progress has been made in developing usable quantum simulators, quantum computers [Goo, 2025b, Rig, 2025b, Mic, 2025, IBM, 2025a, Xan, 2025, D-W, 2025b, hon, 2025], and quantum development kits [Qis, 2025, D-W, 2025a, Microsoft Azure, 2025, Goo, 2025a, Rig, 2025a, Xanadu, 2025] resulting in the field becoming a full fledged technology industry [Karalekas et al., 2020]. One of its biggest challenges is to solve classically intractable problems quantumly. However, challenges with scalability, availability of logical qubits, and decoherence are a hindrance for practically running fault-tolerant quantum computations effectively [Córcoles et al., 2019].

The emerging fields of Hybrid Quantum-Classical (HQC) computation and Quantum High-Performance Computing (QHPC) hold great promise for demonstrating quantum utility. These fields leverage both classical and quantum resources to solve challenging problems that are classically intractable. Variational Quantum Algorithms (VQAs) are the result of one of the most significant developments in hybrid quantum-classical techniques. These techniques can be used to solve practical real-world challenges [Farhi et al., 2014, Peruzzo et al., 2014] that are **NP**-hard problems such as simulating energies in molecules. The most notable hybrid techniques are the Variational Quantum Eigensolver (VQE), Quantum Approximate Optimization Algorithm (QAOA), and the Quantum Support Vector Machine (QSVM).

The Quantum Approximate Optimization Algorithm (QAOA) [Farhi et al., 2014] is a variational quantum algorithm specifically designed to solve *unconstrained* combinatorial optimization problems. It involves finding a best solution from a large (but finite) set of solutions (e.g., from all possible subsets of elements of the problem input). Note that the solution space of such a problem has only feasible solutions, and can be described in the form of Quadratic Unconstrained Binary Optimization (QUBO) or Higher-order Unconstrained Binary Optimization (HUBO) problems.

In contrast, *constrained* combinatorial optimization problems refer to those combinatorial optimization problems where constraints limit the set of feasible solutions (or constrain the solution space). Some of the key methods for modeling constraints in a QUBO or HUBO framework include:

Method 1: introducing *penalty parameters* to discourage solutions that violate the constraints during the optimization process [Lucas, 2014];

Method 2: designing QAOA circuits that operate within a *feasible subspace* [Hadfield et al., 2019] only; and

Method 3: initializing QAOA circuits with quantum states that correspond exclusively to feasible problem configurations [Bärtschi and Eidenbenz, 2019].

These methods come with their own specific challenges: Method 1 requires the determination of (input specific) penalty parameters and allows infeasible solutions (thereby making the solution space larger) while Methods 2 and 3 require more sophisticated quantum circuitry compared to Method 1 to restrict the initial state or restrict the solution space to only feasible solutions. This raises a fundamental question: can we develop a systematic approach to transforming constrained optimization problems into unconstrained versions that avoids both penalty parameter tuning and complex quantum circuits while maintaining solution quality guarantees?

1.1. Problem Statement

Building upon these motivating factors and the broad optimization research context, we formulate the central problem addressed in this dissertation:

The effective solution of constrained and unconstrained combinatorial optimization problems remains a significant challenge in classical computing, with many real-world applications requiring solutions that must satisfy multiple constraints while optimizing specific objectives. While quantum computing shows promise for addressing such computationally intensive problems, the challenge lies in effectively encoding these constraints and obtaining feasible solutions. This dissertation addresses the fundamental question: How can we develop effective encoding strategies for constrained combinatorial optimization problems that leverage both classical and quantum computing resources to consistently produce feasible optimal and near-optimal solutions? Our focus is on developing scalable hybrid approaches that not only maintain solution feasibility but also improve the probability of finding high-quality solutions within the limitations of current quantum computing technologies. For highly practical multi-objective optimization problems, computing not only optimal, but also near-optimal solutions is highly beneficial.

1.2. Challenges considered

This dissertation focuses on two key areas.

The first set of challenges relates to the development of effective problem encoding strategies for constrained combinatorial optimization problems and is described as follows:

- CH1: The transformation of constrained combinatorial optimization problems into quantum circuits formulated for QAOA presents a fundamental challenge: it either risks ineffective encodings that may produce infeasible solutions, or implements complicated quantum circuits (involving a large number of two-qubit or multi-qubit gates) that maintain feasibility but increase computational complexity.
- CH2: Designing encoding strategies that not only target optimal solutions but also effectively capture near-optimal solutions, recognizing that quantum systems' probabilistic nature often returns a distribution of solutions. This requires carefully balancing the encoding's ability to identify global optima while preserving meaningful information about high-quality local optima that may be more practically attainable under current hardware constraints.
- CH3: Balancing encoding approaches between qubit-intensive QUBO formulations that offer shallow circuits but increased probability of infeasible solutions, versus compact higher-order representations that better preserve problem constraints at the cost of deeper circuits, while considering hardware quality limitations and problem characteristics.

The second set of challenges addresses navigating the limitations of current quantum and hybrid quantum-classical computing systems and can be characterized as follows:

CH4: The difficulty in training variational circuits [Bittel and Kliesch, 2021] due to penalty-based encodings includes tuning of both problem parameters and constraint penalty coefficients.

CH5: Impact of qubit decoherence on encoding choices, where deeper circuits face increased quantum noise and error rates, while shallower circuits trade accuracy for better hardware reliability. In the context of combinatorial optimization problems, this presents the challenge of maximizing the probability of obtaining high-quality feasible solutions while minimizing QAOA layer depth (independent of the encoding strategy trade-offs addressed by challenge CH2).

CH6: Performance overhead in hybrid workflows due to communication delays between classical and quantum resources, impacting the efficiency of iterative optimization procedures. The challenge lies in minimizing this overhead while ensuring that the hybrid workflow remains effective.

1.3. Research Questions

In this dissertation, we aim to address the challenges outlined above by investigating the following research questions. These questions are designed to guide our exploration of effective strategies to solve *constrained* combinatorial optimization problems leveraging both classical and quantum computing resources effectively.

RQ1: What characteristics of existing penalty-based encoding approaches for constrained combinatorial optimization problems impact their ability to maintain solution feasibility and optimization quality across different problem classes?

RQ2: How can we leverage the inherent limitations of penalty-based approaches for constrained combinatorial optimization problems to develop novel encoding strategies that enhance solution feasibility?

RQ2.1: How can we effectively capture both optimal and near-optimal solutions while accounting for the probabilistic nature of quantum systems and current hardware constraints?

RQ2.2: How can we leverage classical pre- and post-processing to maximize the probability of obtaining high-quality feasible solutions while minimizing QAOA circuit depth under current hardware noise and decoherence constraints?

RQ3: How can we develop a targeted encoding framework for constrained combinatorial optimization problems that reduces or eliminates penalty coefficient dependencies while maintaining solution quality?

RQ4: Using such a framework, how can we identify and effectively formulate problem types that can be encoded as QUBOs?

RQ5: Using such a framework, which problems can be effectively encoded as HUBOs and to what extent can we optimize the trade-off between QUBO and higher-order encodings to balance circuit depth, qubit requirements, and solution feasibility for different problem classes and hardware constraints?

RQ6: How can we experimentally evaluate the effectiveness of the SCOOP framework, and what insights can we gain about the performance of QUBO and HUBO encodings in practical applications?

RQ6.1: How do these encodings compare to traditional penalty-based approaches in terms of solution feasibility and quality?

RQ6.2: How can we leverage quantum simulators to minimize the communication overhead in hybrid quantum-classical workflows while maintaining the effectiveness of iterative optimization procedures?

RQ6.3: How can SCOOP strengthen quantum computing benchmarking efforts for combinatorial optimization, and how can it be used evaluate the utility-scale performance of quantum algorithms on current hardware platforms?

1.4. Research Objectives

The dissertation's objectives are to develop effective encoding strategies for constrained combinatorial optimization problems that leverage both classical and quantum computing resources, with a focus on maintaining solution feasibility and improving the probability of determining high-quality solutions for constrained combinatorial optimization problems. We characterize the overarching objectives as follows:

O1: Investigate classical pre- and post-processing techniques from parameterized complexity and their potential applicability when solving **NP**-hard problems using hybrid quantum-classical techniques.

O2: Develop novel encoding strategies that:

O2.1: Transform limitations of penalty-based approaches into advantages for solution feasibility.

O2.2: Capture both optimal and near-optimal solutions while considering current hardware noise and decoherence constraints.

O2.3: Integrate classical pre- and post-processing techniques to enhance solution quality while minimizing circuit depth.

O3: Develop a targeted encoding framework for constrained combinatorial optimization problems that minimizes penalty coefficient dependencies while maintaining solution quality.

O4: Leverage tensor network simulation techniques to optimize QAOA circuit parameters while reducing the computational overhead in hybrid quantum-classical workflows.

O5: Realize a hybrid encoding toolkit (comprising libraries, automation, and, frameworks) using various Quantum Development Kits, such as Qiskit and PennyLane to facilitate the implementation of the proposed encoding strategies and hybrid quantum-classical workflows.

O6: Evaluate the performance of the proposed quadratic and higher-order encoding strategies and hybrid quantum-classical workflows on various problems in constrained combinatorial optimization.

O7: Evaluate the performance of the proposed quadratic and higher-order encoding strategies and hybrid quantum-classical workflows on various simulator and hardware platforms, such as PennyLane, Argonne QTensor [Lykov et al., 2021], and IBM Quantum in constrained combinatorial optimization.

O8: Provide insights into the practical implications of the proposed encoding strategies for achieving quantum utility in constrained combinatorial optimization problems.

1.5. Dissertation Outline

The remainder of this dissertation is organized as follows.

Chapter 2 Context and State-of-the-Art Background In this chapter, we provide the necessary background on quantum computing fundamentals, QAOA, and classical optimization techniques, setting the foundation for our proposed SCOOP framework and subsequent analysis of both quadratic and higher-order problem formulations. We also discuss the current state of the art in hybrid quantum-classical approaches to constrained combinatorial optimization problems.

Part II: Unconstrained SCOOP formulations of constrained combinatorial optimization problems

Chapter 3 Beyond Feasibility: The Hidden Value of Infeasible Solutions We introduce the concept of penalty-free optimization through the MinVC problem. We analyze limitations of traditional penalty-based approaches and presents the Maximum Profit Cover problem as an alternative solution method. We show how both optimal and near-optimal solutions can be captured without relying on penalty parameters, establishing foundational concepts that will be generalized in later chapters through the SCOOP framework.

Chapter 4 SCOOP Framework We present the SCOOP framework, which integrates the insights gained from Chapter 3 and provides a structured approach to formulating unconstrained versions of constrained combinatorial optimization problems. The framework

emphasizes the importance of capturing both feasible and infeasible solutions and provides guidelines for adapting existing algorithms to leverage the benefits of the SCOOP approach.

Chapter 5 Quadratic Problem Formulations In this chapter, we apply the SCOOP framework to quadratic problem formulations, demonstrating how it can be used to transform constrained combinatorial optimization problems into unconstrained QUBO problems. We explore the problems of MAXIMUM INDEPENDENT SET, MAXIMUM CLIQUE, MAXIMUM SET PACKING and MAXIMUM 3 SATISFIABILITY and show how the SCOOP framework can be used to derive unconstrained quadratic formulations of these problems.

Chapter 6 Higher-order Problem Formulations We extend the SCOOP framework to higher-order problem formulations, focusing on the NP-hard problems MINIMUM DOMINATING SET, MINIMUM MAXIMAL MATCHING, MINIMUM EDGE DOMINATING SET and MINIMUM SET COVER. We also extend the SCOOP framework to polynomial-time solvable problems MAXIMUM MATCHING and MINIMUM EDGE COVER. We explore the challenges and opportunities presented by higher-order encodings, demonstrating how they can be used to capture complex relationships in constrained combinatorial optimization problems without using penalty parameters.

Part III: Evaluation

Chapter 7 Experimental Setup and Methodology In this chapter, we describe the experimental setup and methodology used to evaluate the performance of the proposed SCOOP framework for selected QUBO and HUBO formulations. We outline the evaluation metrics, datasets, and quantum platforms used in our experiments. We also discuss the implementation details of the hybrid quantum-classical workflows and the tools used for simulation and execution.

Chapter 8 Results and Discussion We present the results of our experiments, comparing the performance of the SCOOP framework's encoding strategies against traditional penalty-based approaches. We analyze the impact of different encoding strategies on solution feasibility, quality, and computational efficiency.

Chapter 9 Evaluation on Quantum Hardware We extend our evaluation to quantum hardware platforms, specifically focusing on the performance of the SCOOP framework's encoding strategies on IBM Quantum devices. We discuss the challenges and limitations encountered during the hardware evaluation and provide insights into the practical implications of our findings.

Part IV: Summary

Chapter 10 Conclusions and Future Work We summarize the key findings of our research, highlighting the contributions of the SCOOP framework to the field of constrained combinatorial optimization. We discuss the implications of our work for future research directions, including potential extensions of the SCOOP framework and its application to other problem domains.

1.6. Chapter Summary

This chapter introduced the fundamental challenges in solving constrained combinatorial optimization problems using hybrid quantum-classical approaches. While quantum computing shows promise for addressing computationally intensive problems, current approaches face significant hardware limitations as well as limitations in effectively encoding constraints and obtaining feasible solutions. We discussed the importance of quantum utility as an intermediate goal towards quantum advantage, focusing on practical applications of quantum computing that can outperform exact classical methods in specific problem domains. We discussed the emergence of hybrid quantum-classical computing as a path towards quantum utility and advantage for not just near-term quantum devices, highlighting key challenges in encoding constrained optimization problems for QAOA implementation. We examined the trade-offs between penalty-based approaches and feasibility-preserving methods. The research questions and objectives outlined focus on developing novel encoding strategies that balance solution feasibility with practical implementation constraints. The next chapter will provide the necessary background on quantum computing fundamentals, QAOA, and classical optimization techniques, setting the foundation for our proposed SCOOP framework and subsequent analysis of both quadratic and higher-order problem formulations.

Chapter 2

Context and State-of-the-Art Background

Contents

In 1981, Richard Feynman challenged the physics community to build a quantum computer with his famous quote: "Nature isn't classical, and . . . if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem,

because it doesn't look so easy." After delivering his famous lectures on *Simulating Physics with Computers* [Feynman, 1982], the field began in earnest with quantum information theory, computing models, and algorithms [Nielsen and Chuang, 2011]—culminating in seminal results, including Shor's 1994 factoring algorithm with exponential speedup [Shor, 1994], and Grover's search (1996) with quadratic speedup [Grover, 1996] over classical algorithms. In recent years, the field of quantum computing has transitioned to a technology industry [Karalekas et al., 2020] with usable near-term quantum devices and roadmaps for fault-tolerant quantum computers [Mount, 2021, Gambetta, 2021]. With near-term quantum devices, an emerging paradigm is hybrid computations involving both classical and quantum resources [Preskill, 2018].

This chapter outlines the key concepts and research areas that form the foundation of the underlying research, including quantum computing, the Quantum Approximate Optimization Algorithm (QAOA), constrained combinatorial optimization, and parameterized complexity.

This chapter is organized as follows. We follow a top-down approach, starting with an overview of the field of combinatorial optimization. Next, we introduce quantum computing in Section 2.3, followed by a discussion of simulating physical systems on quantum computers in Section 2.3.3. We then introduce constrained and unconstrained combinatorial optimization in Section 2.4, followed by a discussion of the Quadratic Unconstrained Binary Optimization (QUBO) and Ising formulations in Section 2.4.1. We then introduce Higher-Order Unconstrained Binary Optimization (HUBO) in Section 2.4.2 and conclude with a discussion of hybrid quantum-classical techniques in Section 2.5.

2.1. Combinatorial Optimization

Combinatorial Optimization Problems (COPs) are problems that require finding an optimal object from a finite set of candidate objects [Schrijver, 2005]. One of the ways to classify COPs is into two categories: *constrained* and *unconstrained* problems. In constrained problems, the feasible solutions must satisfy certain constraints, while in unconstrained problems, all combinations of variables are considered feasible solutions. We define the following terms that are relevant to this dissertation:

- **Combinatorial Optimization Problem (COP):** A COP is a problem that requires finding an optimal object from a finite set of candidate objects.
- **Constrained Combinatorial Optimization Problem (CCOP):** A CCOP is a COP that has constraints that must be satisfied by the feasible solutions.
- **Unconstrained Combinatorial Optimization Problem (UCOP):** A UCOP is a COP that does not have any constraints on the feasible solutions.
- **Binary Variables:** Binary variables are variables that can take on only two values, typically 0 or 1. Binary variables are often used to represent the inclusion or exclusion of an object in a solution.

- **Feasible Solution:** A feasible solution is a solution that satisfies all the constraints of a CCOP. In the context of UCOPs, all combinations of variables are considered feasible solutions.
- **Infeasible Solution:** An infeasible solution is a solution that does not satisfy all the constraints of a CCOP.
- **Optimal Solution:** An optimal solution is a feasible solution that maximizes or minimizes the objective function of an optimization problem.
- **Solution Space:** We define the solution space to be the set of all feasible solutions to a COP. In the case of UCOPs, the solution space is the set of all combinations of binary variables.
- **Objective/Cost Function:** An objective function is a function that assigns a value to each feasible solution of an optimization problem. The goal of the optimization problem is to find the feasible solution that maximizes or minimizes the objective function.

2.2. Computational Complexity Theory

Computational complexity theory concerns classifications of *problems* based on the computational resources required to find their *solutions*. For combinatorial optimization problems (COPs), a *problem* instance consists of a finite set of objects and an objective function, while a *solution* represents a selection or arrangement of these objects that optimizes the objective function.

A problem $\mathcal{P}$, is a relation between its input instances $\mathcal{I}$ and its solutions $\mathcal{S}$ [Garey and Johnson, 1979]. The complexity class **P** concerns decision problems that can be solved deterministically in polynomial time and the complexity class **NP** concerns decision problems that can be solved non-deterministically in polynomial time. The class of **NP**-*complete* problems encompasses all problems in **NP** that are at least as hard as any other problem in **NP**. **NP**-*complete* problems are considered intractable, i.e., for such a problem it is assumed that a deterministic polynomial-time algorithm does not exist [Garey and Johnson, 1979]. **NP**-*Hard* problems are at least as hard as the hardest problems in **NP**, but they do not have to be decision problems. A problem is **NP**-*complete* if it is both in **NP** and **NP**-*hard*. It is famously unknown whether or not **P**=**NP**. The working assumption of computer scientists is that **P**$\neq$**NP**.

Different combinatorial optimization problems have been characterized by varying levels of computational complexity. While some fundamental problems like MINIMUM SPANNING TREE and MAXIMUM MATCHING admit polynomial-time solutions, a significant number of practically important problems, including MAXIMUM INDEPENDENT SET and MINIMUM VERTEX COVER, are **NP**-hard.

Throughout this dissertation, we use the MINIMUM VERTEX COVER problem as a running example to illustrate key concepts. This is a classic **NP**-hard optimization problem serves as an ideal case study because it combines structural simplicity with computational hardness, exhibits clear constraints, and is well-studied in both classical and quantum contexts.

2.2.1. Vertex Cover

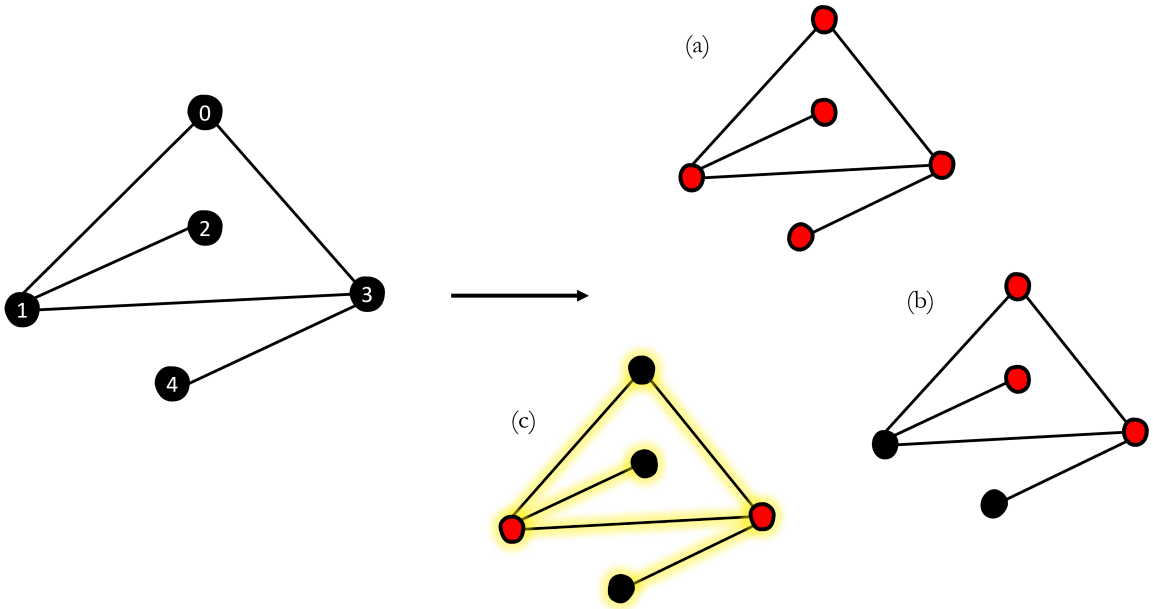


Figure 2.1.|: The graph on the left is a 5 node graph with 5 edges. Graphs (a), (b), and (c) on the right show vertex covers (vertices highlighted in red) of different sizes for a given graph. (c) shows the minimum vertex cover with a size of 2 covering all edges.

Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, a vertex cover VC is a subset of V, where $(u, v) \in E$ implies that $u \in$ VC or $v \in$ VC, i.e., a vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. An example is shown in Fig. 2.1.

2.2.2. Decision and Optimization Problems

A problem $\mathcal{P}$ can be formulated as a decision problem or an optimization problem.

A *decision problem* can be characterized by a tuple $\{\mathcal{I}, \mathcal{S}\}$ with $\mathcal{S} \in \{YES, NO\}$, i.e., a decision problem is one that has a *yes* or *no* answer for any given input. For example, VERTEX COVER is a decision problem with the question "Given a graph, G, does there exist a vertex cover of size at most k?" Decision problems can be expressed as languages. In the case of VERTEX COVER this is $L_{VC} = \{< V, k >\}$, where G is a graph that has a vertex cover of size at most k. Vertex cover is a decision problem that is **NP**-*complete* [Garey and Johnson, 1979].

An *optimization problem* seeks the best solution among a collection of possible solutions and is characterized by a quadruple $\{\mathcal{I}, \mathcal{S}, m, goal\}$. Here, $\mathcal{S}$ is a function that associates an instance of $\mathcal{I}$ to its set of feasible solutions, m is a function that computes a positive integer based on the value of a feasible solution in $\mathcal{S}$, and $goal$ specifies whether $\mathcal{P}$ is a *minimization* or a *maximization*. Continuing with the above example, the problem of finding the MINIMUM VERTEX COVER is an optimization problem and is **NP**-hard. Any optimization problem $\mathcal{P}$ has an associated decision problem $\mathcal{P}_D$ [Ausiello et al., 2012].

2.2.3. Parameterized Complexity

In classical computing, there are many methods to tackle problems for which (likely) no polynomial-time algorithm exists [Karp, 1972, Downey et al., 1999c, Vazirani, 2001, Hromkovič, 2013]. Some examples of such techniques are randomized algorithms, approximation algorithms and heuristic algorithms. All these methods sacrifice solution quality for speed.

Another way to look at computational intractability is through the lens of parameterized complexity. Parameterized complexity characterizes problems concerning parameters of input or output [Downey et al., 1999c]. Formally, a parameterized decision problem can be described as a language $L \subseteq \Sigma^* \times \mathbb{N}$ where Σ is a finite alphabet, $\mathbb{N}$ is a non-negative integer. The parameterized version for VERTEX COVER that we consider here is called the k-VERTEX COVER, and is defined the same as a VERTEX COVER with the addition of considering k, the size of the vertex cover to be determine, as the problem parameter.

2.2.3.1. Fixed-Parameter Tractability

A parameterized problem L is fixed-parameter tractable (or in class FPT) if the question "$(x, k) \in L$" can be decided in $f(k)|x|^{\mathcal{O}(1)}$ running time, where f is an arbitrary function dependent only on parameter k. Note that $f(k)$ can be super-polynomial. While VERTEX COVER is a decision problem, k-VERTEX COVER is a parameterized problem that is fixed-parameter tractable.

Not all parameterized versions of **NP**-*complete* problems are considered fixed-parameter tractable. Instead, a parameterized problem can be hard with respect to a superclass of FPT, falling into the following hierarchy

$$\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \cdots \text{XP}$$

called the W-hierarchy (or the *weft* hierarchy). Here, $W[1]$ is the parameterized analog of **NP**, i.e., for a problem that is $W[1]$ hard, every problem in $W[1]$ can be reduced to it. It is assumed that $W[1] \neq$ FPT. An example of a $W[1]$-hard problem is the k-INDEPENDENT SET, which is the problem of finding a subset I of vertices in a graph such that no two

vertices of I share an edge. The problem parameter is the size of the independent set to be determined.[1]

The Bounded Search Tree technique is a fundamental algorithm design approach for fixed-parameter algorithms that employs recursive branching while ensuring the recursion depth remains bounded by a function of the parameter. This method systematically explores different cases of a problem through recursive branching and backtracking. For vertex cover, this approach creates a search tree where each node represents a partial solution, with branches corresponding to different choices for including vertices in the cover, ensuring the tree's depth remains bounded by the parameter k.

Kernelization: For fixed-parameter algorithms, there exists a kernelization step (i.e., a polynomial-time pre-processing algorithm that reduces the given parameterized decision problem to a (smaller) one that has a size that depends only on the parameter instead of the original input size). Kernelization is a technique by which problem inputs are reduced in size (using polynomial-time pre-processing) before running a computationally expensive algorithm, to achieve efficiency. Problems that are fixed-parameter tractable can be kernelized in polynomial time. For the vertex cover problem, there are several effective pre-processing rules that prune the search space and reduce the size of the input graph, such as removing isolated vertices or vertices of degree greater than k. These rules can be applied iteratively until no further reductions are possible, resulting in a smaller instance of the problem that retains the same solution properties. We describe a few of these reduction rules in Appendix A.

[1] We note that a graph, G has a vertex cover of size k if and only if G has an independent set of size at least $n - k$, where n is the number of vertices in the graph.

2.3. Quantum Computing

Quantum computing leverages the principles of quantum mechanics to perform computations. The fundamental unit of quantum information is the quantum bit or *qubit*, which unlike classical bits, can exist in a superposition of states. Quantum computers can be realized using various physical systems, including superconducting qubits [Martinis, 2004], trapped ions [Bruzewicz et al., 2019], and topological qubits [Aasen et al., 2025]. While various quantum computing models exist, including continuous-variable [Lloyd and Braunstein, 1999] and measurement-based quantum computing [Briegel et al., 2009], throughout this dissertation we focus on the gate-based model unless stated otherwise. The gate-based model uses sequences of quantum gates to manipulate qubits, forms the basis for many quantum algorithms and allows for representation of complex computations as a series of gate operations.

Quantum annealing [Johnson et al., 2011] is another quantum computing paradigm which we mention here as it focuses on solving optimization problems. The *quantum annealer* evolves the system from an initial state to a final state that represents the solution to the optimization problem. This process is guided by the principles of quantum tunneling and adiabatic evolution, allowing the system to explore multiple configurations simultaneously. Quantum annealers, such as those developed by D-Wave Systems [D-W, 2025a], are designed to solve combinatorial optimization problems by finding the ground state of a Hamiltonian that encodes the problem. The quantum annealing process involves gradually changing the Hamiltonian from an initial form to a final form that represents the problem to be solved.

2.3.1. Quantum Mechanics

Quantum mechanics is a set of four postulates that provide a mathematical framework to describe the behavior of quantum systems [Nielsen and Chuang, 2011]:

First Postulate (State space): A quantum system is described by a state vector, which is a complex vector in a Hilbert space. The state of a quantum system can be represented as a linear combination of orthonormal basis states.

Second Postulate (Unitary evolution): The evolution of a quantum system is governed by unitary transformations, which preserve the norm of the state vector.

Third Postulate (Measurement): The act of measurement collapses the quantum state into one of the basis states, with probabilities determined by the amplitudes of the state vector.

Fourth Postulate (Composite systems): The state of a composite quantum system is described by the tensor product of the individual systems' state vectors.

A qubit's state can be represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$ are the *basis states* of a qubit. The coefficients α and β represent the probability *amplitudes* of measuring the qubit in the respective states, with $|\alpha|^2$ and $|\beta|^2$ giving the probabilities of measuring the qubit in state $|0\rangle$ or $|1\rangle$, respectively.

Quantum computations are performed through unitary quantum gates that manipulate these quantum states. Quantum gates can be applied to one or more qubits, and they can be combined to form quantum circuits. Common single-qubit gates include the Pauli gates (X, Y, Z), Hadamard gate (H), and phase gates. Two-qubit gates, such as the controlled-NOT (CNOT) gate, enable interaction between qubits.

The power of quantum computing stems from three key quantum mechanical properties:

- **Superposition**: Allows qubits to exist in a linear combination of states, enabling parallelism in computation.
- **Entanglement**: Enables qubits to be correlated in such a way that the state of one qubit can depend on the state of another, even when they are separated by large distances. This property is crucial for quantum teleportation and quantum cryptography.
- **Interference**: The ability to combine quantum states in a way that can amplify or cancel out certain outcomes, allowing for the construction of algorithms that can outperform classical counterparts.

The Hadamard gate is a fundamental quantum gate that creates superposition states. Entanglement can be achieved through operations such as the Hadamard gate and the controlled-NOT (CNOT) gate.

Quantum algorithms exploit the three quantum mechanical properties to solve problems more efficiently than classical algorithms. Notable quantum algorithms include Shor's algorithm [Shor, 1994] with exponential speedup for factoring large integers and Grover's algorithm [Grover, 1996] with quadratic speedup for unstructured search.

2.3.2. Expectation Values and Measurements

A quantum measurement is a process that extracts information from a quantum system by collapsing or projecting its state vector into one of the basis states. The outcome of a measurement is probabilistic, with the probabilities determined by the amplitudes of the state vector. In quantum mechanics, physical variables are represented by mathematical operators called *observables*. These observables are expressed as Hermitian matrices whose *eigenvalues* correspond to the possible measurement outcomes in experimental settings. The Hamiltonian is a particularly significant observable that represents the total energy of a quantum system. The eigenvalues of the Hamiltonian operator determine the discrete

energy levels that can be measured in the system, making it fundamental to understanding quantum systems' behavior and evolution. In physical systems, Hamiltonians possess a spectrum of ordered eigenvalues ($E_0 \leq E_1 \leq E_2 \leq \cdots$), with corresponding eigenvectors representing the system's possible quantum states, known as energy eigenstates. The ground state, corresponding to the lowest eigenvalue E_0, or lowest energy, represents the system's most natural and stable configuration, which it naturally evolves toward and maintains when undisturbed. While measuring an energy eigenstate yields its corresponding eigenvalue with certainty, superposition states yield probabilistic outcomes based on their decomposition in the energy eigenbasis.[2] This ground state property makes Hamiltonians particularly useful for optimization problems, where the lowest energy configuration often corresponds to the optimal solution. The expectation value of an observable $\hat{O}$ in a quantum state $|\psi\rangle$ is defined as follows:

$$\langle O \rangle = \langle \psi | \hat{O} | \psi \rangle \tag{2.1}$$

where $\langle \psi |$ is the conjugate transpose of the state vector $|\psi\rangle$. The expectation value provides a statistical average of the observable's measurement outcomes when the quantum system is in the state $|\psi\rangle$. This metric guides the design of quantum and hybrid quantum-classical algorithms, as successful algorithms leverage interference to maximize the probability amplitude of desired states while suppressing unwanted ones.

Finding the lowest eigenvalues of Hamiltonians has implications for optimization problems across various domains, including portfolio optimization, logistics planning, and scheduling [Cerezo et al., 2021, Abbas et al., 2024]. By encoding optimization problems into Hamiltonians, we can leverage quantum systems to find optimal solutions, where the ground state eigenvector corresponds to the optimal solution. While classical algorithms struggle with eigenvalue computation, especially for large, dense matrices, quantum algorithms such as Quantum Phase Estimation (QPE) and hybrid quantum-classical approaches (such as Variational Quantum Algorithms) offer promising avenues for finding these eigenvalues.

2.3.3. Simulating Nature on Quantum Computers

Most physical systems exist in continuous states, necessitating approximation through discretization for computational modeling on classical digital computers. While these approximations can be refined with increased computing power, our ability to understand complex physical systems remains fundamentally limited by available computational resources.

Simulation of physical systems on a *quantum* computer requires three fundamental components: (1) a quantum encoding of the physical system's state, (2) the ability to manipulate qubits to model the system's evolution (called unitary evolution), and (3) measuring meaningful physical quantities. The encoding process involves mapping the physical system's degrees of freedom onto quantum states, while the evolution is implemented

[2] Eigenvectors are also referred to as eigenbasis as they are the basis states.

through quantum gates that represent the system's dynamics. While the first two components (encoding and unitary evolution) are deterministic, the measurement process is inherently probabilistic, requiring careful design to ensure that the extracted information is relevant and useful.

2.3.3.1. Hamiltonians

A Hamiltonian, introduced in the earlier section, is a Hermitian mathematical operator (an observable) that describes the total energy of a quantum system. The time evolution of a quantum system is governed by the Schrödinger equation, which relates the Hamiltonian to the state vector's time dependence:

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle \tag{2.2}$$

where $\hat{H}$ is the Hamiltonian operator. The solution to this equation yields the system's state at any time t:

$$|\psi(t)\rangle = e^{-i\hat{H}t/\hbar}|\psi(0)\rangle \tag{2.3}$$

where $|\psi(0)\rangle$ is the initial state of the system. Here $\hbar$ is the reduced Planck constant. The Hamiltonian operator $\hat{H}$ encodes the system's dynamics, including interactions between particles and external fields. The time evolution operator $e^{-i\hat{H}t/\hbar}$, which is unitary when $\hat{H}$ is Hermitian, describes how the quantum state evolves over time, allowing for the simulation of complex physical processes. This unitary nature of quantum evolution is fundamental to quantum computation, as it ensures that quantum operations preserve the normalization of quantum states.

For a Hamiltonian $\hat{H}$ and its eigenstate $|E\rangle$ with energy E, the time evolution can be derived as follows:[3]

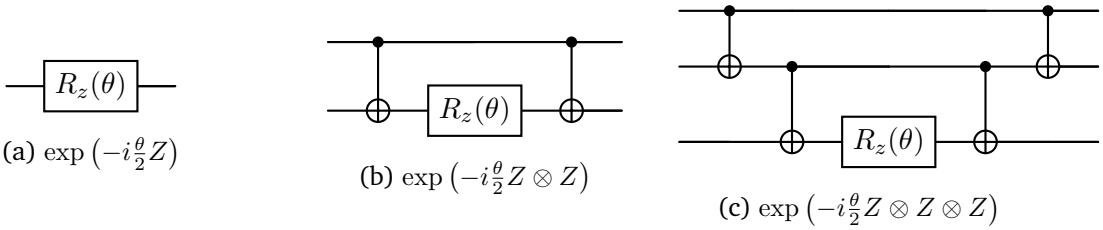
$$\begin{aligned}
U|E\rangle &= e^{-i\hat{H}t}|E\rangle \\
&= \Big(I + (-i\hat{H}t) + \frac{1}{2!}(-i\hat{H}t)^2 + \cdots\Big)|E\rangle \\
&= \Big(|E\rangle + (-it)\hat{H}|E\rangle + \frac{1}{2!}(-it)^2\hat{H}^2|E\rangle + \cdots\Big) \\
&= \Big(|E\rangle + (-it)E|E\rangle + \frac{1}{2!}(-it)^2E^2|E\rangle + \cdots\Big) \\
&= \Big(1 + (-iEt) + \frac{1}{2!}(-iEt)^2 + \cdots\Big)|E\rangle \\
&= e^{-iEt}|E\rangle
\end{aligned}$$

[3]Note that the Planck constant $\hbar$ is often set to 1 in literature, simplifying the time evolution operator to $e^{-i\hat{H}t}$ instead of $e^{-i\hat{H}t/\hbar}$.

2.3.3.2. Unitary Evolution

The time evolution of a quantum state under a Hamiltonian is unitary, meaning it preserves the inner product of states and ensures that probabilities remain valid throughout the computation. The implementation of our framework relies on the following fundamental unitary operations: single-qubit rotations around the Z-axis and X-axis, and two-qubit ZZ rotations. Here we derive these rotations to show how they translate into practical quantum circuit implementations. For a single qubit with Hamiltonian, for example, $\hat{H} = -\alpha Z_0$, the time evolution operator can be implemented using a rotation around the Z-axis. Using the relationship $RZ(\theta) = e^{-i\frac{\theta}{2}Z}$, we can express the evolution as:

$$|\psi(t)\rangle = e^{-i\hat{H}t}|\psi(0)\rangle \tag{2.4}$$
$$= e^{-i(-\alpha Z_0)t}|\psi(0)\rangle \tag{2.5}$$
$$= e^{i\alpha t Z_0}|\psi(0)\rangle \tag{2.6}$$
$$= RZ(-2\alpha t)|\psi(0)\rangle \tag{2.7}$$



(a) $\exp\left(-i\frac{\theta}{2}Z\right)$ (b) $\exp\left(-i\frac{\theta}{2}Z\otimes Z\right)$ (c) $\exp\left(-i\frac{\theta}{2}Z\otimes Z\otimes Z\right)$

Figure 2.2.|: Implementation of single-qubit (Z), two-qubit(ZZ), and three-qubit(ZZZ) interactions using quantum gates.

In a similar manner, two-qubit and multi-qubit Z interactions can be implemented using circuit identities and as shown in Fig. 2.2. For example, the two-qubit interaction ZZ can be implemented using a controlled-Z gate followed by a rotation around the Z-axis on the second qubit. These single and multi-qubit operations form the building blocks for implementing Hamiltonians that encode optimization problems, which are discussed in detail in Section 2.4.

2.3.3.3. Trotterization

Trotterization is a technique used to approximate the time evolution operator for a Hamiltonian that can be expressed as a sum of *non-commuting* terms. The Trotter-Suzuki decomposition [Suzuki, 1993] allows for the splitting of the time evolution operator into a product of exponentials, each corresponding to a single term in the Hamiltonian. This enables the implementation of the time evolution operator using a sequence of quantum gates, making it feasible to simulate complex quantum systems on a quantum computer. For a Hamiltonian that can be expressed as a sum of non-commuting terms

($\hat{H} = \hat{H_A} + \hat{H_B}$), implementing the time evolution operator $e^{-i\hat{H}t}$ requires special consideration. Two operators A and B commute if $AB = BA$, and anti-commute if $AB \neq BA$, with their commutator denoted as $[A, B] = AB - BA$. Unlike commuting operators where $e^{A+B} = e^A e^B$, non-commuting operators follow the more complex Zassenhaus formula, involving nested commutators: $e^{A+B} = e^A e^B e^{-\frac{1}{2}[A,B]} e^{-\frac{1}{6}(2[B,[A,B]]+[A,[A,B]])} \ldots$. The Trotter-Suzuki decomposition provides a practical solution by approximating the evolution as $\exp(A + B) \approx \lim_{n\to\infty}(\exp(\frac{A}{n}) \exp(\frac{B}{n}))^n$. This allows us to implement the simultaneous effect of non-commuting terms by alternating between small time steps of each term, making it feasible to simulate complex quantum systems on gate-based quantum computers. This forms the basis for many quantum algorithms, including the Quantum Approximate Optimization Algorithm (QAOA), which is discussed in detail in Section 2.5.

2.4. Encoding combinatorial optimization problems

So far, we have introduced the concepts of combinatorial optimization problems, quantum computing, and the mathematical framework for simulating physical systems on quantum computers. In this section, we focus on how combinatorial optimization problems can be encoded into quantum systems, particularly in the context of constrained combinatorial optimization problems (CCOPs).

Let $C(x) = \sum_{i=1}^{N} C_i(x)$ be a function, where $\vec{x} = [x_1, x_2, \ldots, x_N]$ holds binary variables. Combinatorial optimization aims to find a binary vector $\vec{x}$ that maximizes or minimizes $C(x)$.

A combinatorial optimization problem can be unconstrained or constrained. Unconstrained problems consider all combinations of binary variables as feasible solutions. The MAXCUT problem is an example of unconstrained optimization, where the objective is to partition a graph's vertices into two disjoint subsets maximizing the sum of weights of edges between them. Every variable assignment corresponds to a cut, making all assignments feasible solutions. Constrained combinatorial optimization problems (CCOPs) impose additional restrictions on the feasible solutions. These constraints can be expressed as linear or non-linear equations, inequalities, or logical conditions that must be satisfied by the binary variables. An example of a CCOP is the MINIMUM VERTEX COVER problem discussed in Section 2.2.1, where the objective is to find a subset of vertices that covers all edges in a graph while minimizing the number of vertices selected. In this case, not all combinations of binary variables cover all edges, making some binary assignments infeasible solutions.

2.4.1. Quadratic Unconstrained Binary Optimization (QUBO)

We now turn our attention to the encoding of combinatorial optimization problems into quantum systems. The most common way to encode combinatorial optimization problems is through the *Quadratic Unconstrained Binary Optimization (QUBO)* formulation [Lucas, 2014]. In a QUBO problem, the objective function is expressed as a quadratic polynomial in binary variables, which can be represented as:

$$C(\vec{x}) = \sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} b_{ij} x_i x_j \tag{2.8}$$

where $x_i \in \{0, 1\}$ are binary variables, $a_i \in \mathbb{R}$ are linear coefficients, and $b_{ij} \in \mathbb{R}$ are quadratic coefficients. The goal is to find the binary vector $\vec{x}$ that minimizes or maximizes the objective function $C(\vec{x})$. The QUBO formulation is particularly well-suited for quantum computing because it can be directly mapped to the Hamiltonian of a quantum system. The objective function can be expressed as a *Hamiltonian operator* using operators described in Section 2.3.3.2, which can then be used in quantum algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) [Farhi et al., 2014]. In QAOA, the

cost Hamiltonian is used to define the cost function that the algorithm seeks to minimize, and the solution corresponds to the ground state of this Hamiltonian.

The QUBO problem provides a framework for representing various combinatorial optimization problems in an unconstrained manner. Many *unconstrained* problems can be represented as QUBO problems using quadratic and linear terms involving binary variables. For example, the problem MaxCut can be defined as a QUBO as follows:

$$\text{Maximize: } C(\vec{x}) = \sum_{(u,v)\in E} (x_u + x_v - 2x_u x_v) \tag{2.9}$$

where $x_u, x_v \in \{0, 1\}$ are binary variables representing which partition a vertex belongs to. For each edge (u, v):

- If vertices are in different partitions ($x_u = 0, x_v = 1$ or $x_u = 1, x_v = 0$), the term contributes $+1$ to the sum.
- If vertices are in the same partition ($x_u = x_v = 0$ or $x_u = x_v = 1$), the term contributes 0.

Thus, maximizing this function finds the partition that cuts the maximum number of edges. To encode binary variables in a quantum system, we use the Ising model. The Ising model is a mathematical model of ferromagnetism [Peierls, 1936], whose basic element is a *spin* which can take one of two values, typically represented as $+1$ or -1. These spins are arranged on a lattice, which can be in one, two, or more dimensions. The Ising model provides a natural framework for formulating QUBO problems on a quantum computer using the transformation $x_i = (1 - s_i)/2$, where the spins $s_i \in \{1, -1\}$. This maps the QUBO problem to an Ising Hamiltonian minimization problem, which is used as the cost Hamiltonian in QAOA [Lucas, 2014].

For the MaxCut problem defined in Eqn. 2.9, we can use the transformation $x_i = (1 - s_i)/2$ to obtain the Ising formulation:

$$\begin{aligned}
C(\vec{x}) &= \sum_{(u,v)\in E} (x_u + x_v - 2x_u x_v) \\
&= \sum_{(u,v)\in E} \Big(\frac{1 - s_u}{2} + \frac{1 - s_v}{2} - \frac{(1 - s_u)(1 - s_v)}{2}\Big) \\
&= \sum_{(u,v)\in E} \Big(\frac{1 - s_u s_v}{2}\Big) \\
&= \frac{|E|}{2} - \frac{1}{2} \sum_{(u,v)\in E} s_u s_v \qquad\qquad = \hat{H}(\vec{s})
\end{aligned}$$

Since problems are usually encoded as minimization problems, we can take the negative of the above expression to obtain the Ising Hamiltonian. Constants can be ignored in the

context of optimization problems, as they do not affect the optimal solution. Thus, we can express the Ising Hamiltonian for the MAXCUT problem as:

$$\text{Minimize: } \hat{H}(\vec{s}) = \frac{1}{2} \sum_{(u,v)\in E} s_u s_v \tag{2.10}$$

Many problems of practical interest are *constrained* optimization problems (i.e., not all assignments of binary variables are feasible). For such problems to be embedded into a QUBO, penalty functions can be applied to a problem to penalize infeasible solutions [Glover et al., 2022]. Penalty function methods offer a simple approach to tackling constrained optimization problems by transforming them into unconstrained optimization problems, allowing the use of algorithms explicitly designed for unconstrained problems. With penalties, infeasible solutions come at a (high) cost, discouraging their exploration in the solution landscape. However, there are limitations to this approach such as the need to choose appropriate penalty parameters and the potential for convergence issues [Smith et al., 1997]. These limitations extend to QAOA when addressing constrained optimization problems. There is a relatively high probability of obtaining infeasible solutions that violate the constraints. Beyond incorporating penalty terms into the objective function, several other considerations arise for QAOA and are described in Section 2.5.4. We also describe the limitations of penalty functions in the context of quantum optimization for the problem MINIMUM VERTEX COVER in Chapter 3.

2.4.2. Higher-Order Unconstrained Binary Optimization (HUBO)

The Ising model [Johnson et al., 2011, Lucas, 2014] is widely used in quantum combinatorial optimization because it provides a natural mathematical structure for encoding binary decision problems in a way that maps directly onto quantum hardware and algorithms like QA or QAOA. The field of quantum optimization relies heavily on Ising model formulations that use quadratic terms due to their direct compatibility with QA hardware. Gate-based hardware has no such restriction, as higher-order terms can be encoded natively using multi-qubit gates [Cowtan et al., 2019]. The higher order unconstrained binary optimization (HUBO) formulation offers a more expressive way to represent many optimization problems by directly incorporating multi-way (K-local) interactions between variables [Wang et al., 2025]. Rapid advancements in gate-based quantum computers, particularly in qubit fidelity and connectivity, are paving the way for more effective handling of multi-qubit interactions [IBM, 2025b], and consequently the encoding of HUBO problems. While QA strictly requires a QUBO formulation, QAOA generally can handle both QUBO and HUBO problems. Initially, lower-order terms and, consequently, QUBOs might be preferred. However, converting a HUBO into a QUBO through quadratization necessitates introducing auxiliary variables and additional penalties [Glover et al., 2022]. In contrast, Campbell et al. [Campbell and Dahl, 2022] investigate the advantages of incorporating higher-order terms (HOTs) into the problem formulation. Their research underscores the

potential of utilizing the inherent capabilities of gate-model quantum computers to manage HOTs, suggesting an expanded application domain for QAOA and paving the way for near-term quantum superiority on NP-hard problems.

Pelofske et al. [Pelofske et al., 2023a] conducted an experimental comparison between QAOA and QA on different hardware platforms. Their study focused on cubic ZZZ interactions, which are natively constructible for QAOA, while QA necessitates order reduction to quadratic interactions using auxiliary variables.

Mandal et al. [Mandal et al., 2020] study quadratization and propose a method of degree reduction that works directly in the Ising space. They note that a *sparse* problem in the Ising space—i.e., its cost Hamiltonian, defined by a sum of local terms that each involves a small, bounded number of variables and each variable appears in only a small number of terms—*is not necessarily sparse* in Boolean space and vice versa. A simple example illustrates this: Consider polynomial $2^n \prod_{i=1}^{n} b_i$. Applying the transformation from binary to Ising variables ($b_i \leftarrow (1 - Z_i)/2$) results in the Hamiltonian $\sum_{I \subseteq \{1,\ldots,n\}} (-1)^{|I|} \prod_{i \in I} Z_i$, which grows exponentially [Mandal et al., 2020]. In the context of the traveling salesperson problem (TSP), Glos et al. [Glos et al., 2022] propose a trade-off between the number of qubits and the circuit depth to demonstrate an efficient binary encoding for TSP.

We note that the feasibility of the encoding of HUBOs or HOTs into a cost Hamiltonians is instance dependent; understanding the structural properties of the problem is crucial for assessing whether or not higher-order terms must be used

2.5. Hybrid Quantum-Classical Techniques

Techniques that combine the strengths of both classical and quantum computing resources are called hybrid quantum-classical techniques. Hybrid approaches can be applied to problem-solving at design time (e.g., problem decomposition and algorithm design techniques) as well as at run-time (e.g., parameterized or variational circuits). Hybrid approaches can also be applied at different levels of abstraction, i.e., at the kernel level, algorithm level, and the model level as envisioned by IBM's quantum roadmap for building an open quantum software ecosystem [Gambetta et al., 2021]. We expand the term to *hybrid quantum building blocks* (or *hybrid toolkit* [Angara et al., 2020] for short) to include quantum algorithms that lend themselves as sub-problems of a larger problem and hybrid algorithmic and architectural design patterns. Examples of such powerful building blocks are *quantum phase estimation (QPE)*, *Grover's search*, *variational quantum eigensolver (VQE)*, *variational quantum factoring (VQF)* [Anschuetz et al., 2018], *quantum approximate optimization algorithm* [Farhi and Harrow, 2016], and solvers for linear systems of equations [Harrow et al., 2009, Lee et al., 2019].

2.5.1. Grover's search and its variants

One of the most famous building blocks is Grover's search [Grover, 1996], a quantum search algorithm that can be used as a subroutine in many different contexts. Given a list of n unstructured (or unsorted) elements, Grover's search finds a specific item in $O(\sqrt{n})$ time. The classical algorithm takes $\Theta(n)$ time. This constitutes a quadratic speedup. The quantum algorithm is based on a black-box transformation and phase kickback and makes use of amplitude amplification. Important variants of Grover's search are the quantum algorithm for finding the minimum in an unstructured set [Dürr and Høyer, 1996], quantum partial search [Grover and Radhakrishnan, 2005], and quantum walk algorithms [Aharonov et al., 2001, Ambainis, 2007].

Since **NP**-complete problems typically can be solved using a brute-force search, Grover's search algorithm can be applied to achieve a quadratic speedup of the brute-force algorithm. More sophisticated classical algorithmic methods for solving **NP**-complete problems include dynamic programming [Bellman, 1962, Held and Karp, 1961, Wang and Tian, 2011], divide and conquer [Ambainis et al., 2019, Ge and Dunjko, 2020], and other methods in the toolkits of exact algorithms [Fomin and Kratsch, 2010] and fixed-parameter tractability [Downey et al., 1998, Cygan et al., 2015].

Ambainis et al. [Ambainis et al., 2019] describe how to apply Grover's search [Grover, 1996] or its variant of finding a minimum item in a set [Dürr and Høyer, 1996] to speedup particular exponential-time dynamic-programming algorithms, for example for the **NP**-complete problems Hamiltonian Circuit, TSP [Bellman, 1962, Held and Karp, 1961] or Minimum Set Cover [Fomin and Kratsch, 2010]. What these dynamic-programming approaches have in common is that they solve subproblems corresponding to subsets of an n-element set. Such a dynamic programming approach in its most basic form typically runs in exponential time in the size of the input. The achieved dynamic programming quantum speed-up relies on pre-computing solutions for sub-problems of a specific size (e.g., say, a quarter of the original instance size) using the classical dynamic programming algorithm, followed by searching—on the remaining subsets—for an answer to the problem using a quantum search in the pre-computed answers for the sub-problems [Angara et al., 2020].

2.5.2. Variational Quantum Algorithm (VQA)

Variational Hybrid Techniques or Variational Quantum Algorithms (VQAs) are emerging general frameworks that have been proposed for many applications suitable for quantum computers such as optimization, simulation of physical systems, and machine learning. The building blocks of variational hybrid techniques are the following:

1. **Cost/Objective function** A cost function $C(\theta)$ encodes the problem's objective function by characterizing it as a quantum operator called a Hamiltonian. VQAs use a quantum computer to estimate this cost function.

2. **Ansatz** An ansatz is a trial state that is prepared using a parameterized quantum circuit. The structure of an ansatz usually depends on the problem being solved (i.e., a problem-inspired ansatz [Hadfield et al., 2019]), but also can be problem-agnostic and hardware-efficient [Kandala et al., 2017a, Rattew et al., 2019].

3. **Classical Optimizer** A classical optimizer is used to optimize the variational parameters. There are various classical optimizers available to use such as SPSA [Spall et al., 1992], COBYLA [Powell, 2007], Gradient Descent [Kantorovich and Akilov, 2014] and the Nelder-Mead [Nelder and Mead, 1965] method and can be chosen according to their performance on the specific optimization landscape.

2.5.3. Variational Quantum Eigensolver (VQE)

VQE is a hybrid algorithm that computes the lowest eigenvalue of a Hamiltonian $\hat{\mathcal{H}}$ [Peruzzo et al., 2014, Kandala et al., 2017b]. VQE techniques are based on the variational method of quantum mechanics, which states that for a given $\hat{\mathcal{H}}$ its expectation value must be greater than or equal to the lowest possible eigenvalue. The Hamiltonian $\hat{\mathcal{H}}$ can be expressed as a set of terms that are realized as separate quantum circuits. The expectation values of its parts are then summed up classically to compute the expectation value of $\hat{\mathcal{H}}$. The initial states for these circuits are selected from a set of states based on an ansatz and are generated by a parameterized circuit. The parameters for the state preparation circuit are computed in a classical optimization loop that minimizes the expectation value of $\hat{\mathcal{H}}$. Compared to Quantum Phase Estimation (QPE) circuits, the depths of the VQE circuits are considerably smaller, which is a big advantage for execution on near-term quantum computers. Progress in computational quantum chemistry and optimization has been driven largely by VQE [Kandala et al., 2017b, McArdle et al., 2020]. While many VQE applications are in molecule simulation, certain machine learning and combinatorial optimization problems can be formulated such that the lowest possible eigenvalue represents the optimal value we would like to obtain. Liu et al. [Liu et al., 2022] propose a notable variant of the VQE, called the Layer-VQE for combinatorial optimization problems and demonstrate it using the networking problem of k-community detection. Instead of using kn qubits to encode the problem (typically as an Ising model Hamiltonian), this work proposes a formulation that requires fewer qubits ($n\lceil \log k \rceil$ qubits).

2.5.4. Quantum Approximate Optimization Algorithm (QAOA)

The quantum approximate optimization algorithm (QAOA) introduced by Farhi et al. [Farhi et al., 2014], is a variational technique to solve combinatorial optimization problems. Here, the quantum part evaluates the objective function and involves alternating between unitaries corresponding to a cost Hamiltonian, H_C, and a mixer Hamiltonian, H_M. A classical optimization loop updates the ansatz parameters. Constraints to the optimization problem can be imposed by adding a penalty term to the problem's objective function (the Lagrange

multiplier approach) or by constructing the variational ansatz in a way such that constraints are satisfied at all times.

QAOA, introduced by Farhi et al. in 2014, is a hybrid quantum-classical algorithm that mimics adiabatic quantum computation on near-term gate-based quantum computers aiming to solve combinatorial optimization problems. Farhi's QAOA, often referred to as *vanilla* QAOA, has inspired numerous variants; selected variants are described at the end of this section.

The key building blocks of a **vanilla QAOA** setup include:

1. An **initial state** $|\psi_0\rangle$ in an equal superposition of all computational basis states, prepared by applying Hadamard gates to qubits initialized in state $|0\rangle^{\otimes n}$.
2. **An alternating ansatz** consisting of:
   - **A cost or phase separator unitary** that encodes the cost function $C(x)$ of the optimization problem being solved by introducing relative phase shifts between the states based on their cost function values. This unitary is generated by the *cost* Hamiltonian (H_C) and for combinatorial optimization problems typically consists of sums of Pauli-Z and ZZ terms. For a minimization problem, the objective is to identify an assignment of variables that yields the smallest possible value of the cost function $C(x)$.
   - A **mixing unitary** that anti-commutes with the cost unitary and enables transitions between the computational basis states. This unitary is generated by the *mixer* Hamiltonian (H_M), commonly chosen as a sum of Pauli-X operators.
   - The number of **layers** p, i.e., the number of times the cost and mixer unitaries are applied alternatively. With increasing alternating layers, QAOA progressively mimics QAA through Trotterization [Farhi et al., 2014].
   - **Parameter vectors**, $\vec{\beta}$ and $\vec{\gamma}$, comprising variational parameters that control the cost and mixer unitaries, respectively, across the p alternating layers.
3. A **classical optimizer** responsible for finding the optimal values of the parameter vectors $\vec{\beta}$ and $\vec{\gamma}$ such that the **expectation value** of the cost function is minimized (for a minimization problem):

$$\min_{\vec{\beta},\vec{\gamma}} \langle\psi(\vec{\beta},\vec{\gamma})|H_C|\psi(\vec{\beta},\vec{\gamma})\rangle$$

   where $|\psi(\vec{\beta},\vec{\gamma})\rangle$ is the state prepared by the alternating ansatz with p layers of cost and mixer unitaries.

The quantum part of QAOA's hybrid routine evaluates the objective function. It alternates between unitaries corresponding to cost Hamiltonian H_C and mixer Hamiltonian H_M, p times.

$$|\psi(\vec{\beta},\vec{\gamma})\rangle = \underbrace{U(\beta_p)U(\gamma_p)\cdots U(\beta_1)U(\gamma_1)}_{p \text{ times}} |\psi_0\rangle$$

where $U(\boldsymbol{\beta}) = e^{-i\boldsymbol{\beta}H_M}$ and $U(\boldsymbol{\gamma}) = e^{-i\boldsymbol{\gamma}H_C}$ characterized by the parameters (β, γ). The goal of QAOA is to determine the optimal parameters $(\boldsymbol{\beta}_{opt}, \boldsymbol{\gamma}_{opt})$ such that the quantum state $|\psi(\boldsymbol{\beta}_{opt}, \boldsymbol{\gamma}_{opt})\rangle$ encodes the solution to the problem. Optimal parameters can be found using different kinds of classical optimizers such as L-BFGS, Adam, COBYLA, and Gradient Descent [Pellow-Jarman et al., 2021]. Implementing QAOA effectively requires careful consideration of several hyperparameters: the circuit depth (number of alternating layers), choice of classical optimizer, optimization parameters (steps and step size), and mitigation of barren plateaus in the optimization landscape [Kulshrestha and Safro, 2022, Huembeli and Dauphin, 2021]. In general, the problem of training variational circuits is NP-hard [Bittel and Kliesch, 2021]. Choosing a classical optimizer is influenced by several factors, including the number of parameters, the problem, and the cost function landscape [Malan and Engelbrecht, 2013, Bonet-Monroig et al., 2023, Sung et al., 2020]. These design choices significantly impact QAOA's performance on constrained optimization problems. Various QAOA variants have been proposed to address these challenges, which we discuss in the following section.

2.5.4.1. Related Work on QAOA

Several QAOA components are tunable, including initialization, ansatz construction, parameter choice, and the classical optimization method. These components significantly influence the algorithm's efficacy and can be tailored to specific problem instances for improved performance. Hadfield et al. [Hadfield et al., 2019] extend QAOA by introducing the Quantum Alternating Operator Ansatz (QAO-Ansatz), which alternates between more general families of unitary operators. This can potentially narrow the algorithm's focus to a more useful set of states. The QAO-Ansatz can be used to guarantee that the state of the circuit never leaves the set of feasible states. However, the circuit is composed of complicated circuitry. For example, multi-controlled Toffoli gates are often used in the ansatz, which are challenging to execute depending on the connectivity of qubits on a quantum computer [He et al., 2017]. Golden et al. [Golden et al., 2023] compare the performance of different variations of QAOA on three problems, Max Bisection, Max k-Vertex Cover, and k-Densest Subgraph, using different kinds of mixers and show a possibility of achieving a super-polynomial advantage over Grover's unstructured search. The problem Max k-Vertex Cover has also been studied by [Cook et al., 2020, Bärtschi and Eidenbenz, 2020], and while it is related to MinVC, is not as complex, due to a deterministic quantum algorithm to prepare Dicke states [Bärtschi and Eidenbenz, 2019] that allows for exploration only in the feasible solution subspace of Max k-Vertex Cover. These problems are either unconstrained or constrained by the Hamming weight and, therefore, do not need penalties in the cost function. Pelofske et al. [Pelofske et al., 2019, Pelofske et al., 2023b] propose a recursive classical decomposition of large problems (as a pre-processing and pruning step) such that they can be solved on quantum annealers. Saleem et al. [Saleem et al., 2023, Tomesh et al., 2023] consider penalty-term approaches, QAO-Ansatz, and introduce a new ansatz variant that adapts to the quantum resources available for the MaxIS.

Other notable variants of QAOA includes QAOA+ [Chalupnik et al., 2022], WS-

QAOA [Egger et al., 2021], Digitized Counterdiabatic QAOA [Chandarana et al., 2022], and multi-angle QAOA [Herrman et al., 2022]. For QAOA+, the authors propose adding a problem-independent layer of parameterized ZZ-gates and the parameterized X-gates (as a mixer) to improve the approximation ratio of MaxCut, which is an unconstrained optimization problem. In WS-QAOA, warm starting methods are used to prepare an initial state that corresponds to the solution of a relaxation of the portfolio optimization problem. In multi-angle QAOA, a parameter is assigned to every element of the ansatz to improve the approximation ratio achieved for MaxCut. Digitized Counterdiabatic QAOA appends a layer (and therefore, an additional variational parameter) to perform counterdiabatic driving to converge to the optimal solution faster (thereby requiring a shorter circuit depth).

The performance of QAOA has been typically studied for Erdős-Rényi Random Graphs with different probabilities of edge connections [Golden et al., 2023, Saleem et al., 2023] as well as bounded-degree graphs (specifically random 3-regular graphs [Shaydulin and Pistoia, 2023,Lykov et al., 2021]). Herrman et al. [Herrman et al., 2021] perform a detailed analysis across different graph structures (up to eight nodes) to understand QAOA MaxCut performance of up to three layers.

2.6. Quantum Computing Software Development Kits

2.6.1. Full statevector simulation

2.6.1.1. Xanadu PennyLane

PennyLane is a cross-platform Python library for differentiable quantum programming, making it a powerful tool for hybrid quantum-classical techniques that involve variational circuits [Bergholm et al., 2018]. PennyLane enables the integration of various libraries such as TensorFlow, PyTorch, and Autograd due to the quantum node abstraction, thereby fitting seamlessly into the existing automatic differentiation methods. In our work, we utilized PennyLane not only for its robust support for variational algorithms but also for its ease of use and the availability of plugins for different quantum simulators and hardware.

2.6.1.2. IBM Qiskit

Qiskit is IBM's comprehensive quantum computing software stack designed for utility-scale quantum applications.

- **Qiskit SDK:** At its foundation lies an open-source SDK enabling work with quantum circuits, operators, and primitives.
- **Qiskit Runtime:** Qiskit Runtime is IBM's quantum computing service that optimizes the execution of quantum programs by reducing the latency between classical and quantum computations [Qis, 2025]. It provides a containerized environment where

both quantum and classical code are executed in proximity to the quantum hardware, significantly reducing the communication overhead in hybrid quantum-classical workflows. The service includes built-in primitives for sampling and estimating expectation values, along with error mitigation techniques that help improve the reliability of results. For variational algorithms like QAOA, Runtime's architecture is particularly beneficial as it minimizes the classical-quantum communication bottleneck during parameter optimization loops. The service also provides dynamic circuit execution capabilities and automated resource estimation, enabling more efficient execution of quantum algorithms on real hardware.

- **Qiskit Transpiler:** Qiskit's transpiler service incorporates heuristics to enhance performance for common quantum tasks.
- **Qiskit Functions:** Qiskit supports both built-in and third-party services for simulation, optimization, and error handling, offering a modular architecture that allows developers to work at their preferred abstraction level.
- **Performance:** Qiskit's codebase is optimized for memory efficiency and improved results, particularly beneficial for variational algorithms and QAOA implementations. Access to IBM Quantum's hardware is provided through cloud services, complemented by tools for noise mitigation and error correction [Qis, 2025].

2.6.2. Tensor Network Simulation with Argonne QTensor

Quantum circuit simulators are essential for understanding how quantum computers work and for developing and testing quantum algorithms and their benchmarking and verification. Variational algorithms that involve iteratively updating the parameters based on the results obtained from quantum circuit executions benefit from fast simulators to obtain parameters. Furthermore, quantum circuit simulators can be used to study the behaviour of such hybrid algorithms under varying parameters.

Full statevector simulation is a common approach to simulating quantum circuits, where the entire quantum state is represented as a vector in a Hilbert space. This method is computationally expensive, especially for larger circuits, as the size of the state vector grows exponentially with the number of qubits. Tensor networks are a powerful computational framework used to efficiently simulate quantum circuits by representing quantum states and operations as interconnected tensors. Instead of working with exponentially large state vectors, tensor networks exploit the structure and entanglement of quantum systems to compress information. Common forms, such as Matrix Product States (MPS), Projected Entangled Pair States (PEPS), or Tree Tensor Networks (TTN) allow for scalable simulations of low-entanglement and shallow circuits [Ibrahim et al., 2022, Orús, 2014]. By contracting these networks in a specific order, one can approximate the evolution of quantum states with significantly reduced computational resources compared to brute-force methods.

Argonne QTensor Simulator [Lykov et al., 2021, Lykov et al., 2022] is a highly efficient quantum circuit simulator library which is founded upon the tensor network contraction

technique [Markov and Shi, 2008], offering a significant performance speedup compared to existing simulators (such as those that run a full amplitude-vector evolution). A core feature of Argonne QTensor is its highly parallelizable evaluation of observables based on the *lightcone* or the *reverse causal cone* approach [Farhi et al., 2020, Streif and Leib, 2020]. This means that if some observable acts only on a small subset of the qubits, then most of the gates in the quantum circuit commute through and cancel out when evaluating the expectation value. The QAOA cost operator is a sum of m independent terms, each of which could be computed separately (i.e., can be computed in parallel). The lightcone (or subgraph) of the computation depends on the number of layers p. For example, when $p = 1$, the simulator only needs to evaluate the circuit on the independent cost term and its immediate neighbourhood. This method works well for sparse or large fixed-degree graphs. However, for dense graphs, since every evaluation requires consideration of numerous nodes, the speedup provided by Argonne QTensor may be negligible.

In Chapter 8, we present experimental results on PennyLane as well as Argonne QTensor simulators. The latter is used to evaluate the performance of our framework on larger problem instances, where PennyLane's full statevector simulation is used for smaller instances.

2.7. Quantum Hardware

Quantum computers leverage different physical implementations, including superconducting qubits [Martinis, 2004], trapped ions [Bruzewicz et al., 2019], and topological qubits [Aasen et al., 2025]. While various quantum computing models exist, including continuous-variable [Lloyd and Braunstein, 1999] and measurement-based quantum computing [Briegel et al., 2009], throughout this dissertation we focus on the gate-based model that uses superconducting qubits unless stated otherwise. The gate-based model uses sequences of quantum gates to manipulate qubits, forms the basis for many quantum algorithms and allows for representation of complex computations as a series of gate operations.

In this dissertation, we use IBM's quantum hardware. IBM's quantum hardware is based on superconducting qubits, which are small circuits that can exist in a superposition of states. These qubits are manipulated using microwave pulses to perform quantum gates. IBM's quantum hardware is accessible through the cloud, allowing users to run experiments on real quantum devices. The hardware is designed to support a wide range of quantum algorithms, including those for optimization problems like QAOA.

IBM Quantum's roadmap outlines progress toward utility-scale quantum computing through a series of hardware advances. The IBM Quantum Starling system aims to enable circuits with 100 million gates operating on 200 logical qubits [4]. Near-term advances like the Nighthawk processor focus on demonstrating quantum utility through support for

[4] https://www.ibm.com/quantum/blog/large-scale-ftqc

more complex circuits, longer-term developments including Kookaburra and Cockatoo processors target fault-tolerance through quantum error correction, modular architectures, and enhanced qubit connectivity. In this dissertation, we make use of the IBM Heron R2 processors, which are superconducting qubit processors with 156 qubits and heavy-hex connectivity. Experimental results on this processor are discussed in Chapter 9.

2.8. Chapter Summary

This chapter established the foundational concepts necessary for understanding hybrid quantum-classical approaches to constrained optimization problems. We examined both gate-based and adiabatic quantum computing models, with particular emphasis on how quantum systems can be used to simulate physical processes through Hamiltonian evolution. The chapter detailed the Quantum Approximate Optimization Algorithm (QAOA), explaining its hybrid nature, variational parameters, and the interplay between cost and mixer Hamiltonians. We explored how QAOA implements optimization problems through quantum circuits and the challenges of parameter optimization.

We presented relevant topics in classical complexity theory concepts including the importance of parameterized complexity in addressing computational intractability. We examined specific pre- and post-processing techniques, using vertex cover as an illustrative example. On the software side, we presented an overview of current quantum software development tools, including PennyLane, Qiskit, and QTensor, highlighting their roles in implementing and simulating quantum algorithms. These frameworks provide essential capabilities for developing and testing hybrid quantum-classical approaches on both simulators and real quantum hardware and are used for evaluating our framework in Chapter 7, Chapter 8 and Chapter 9. This background sets the stage for our subsequent exploration of novel encoding strategies and the SCOOP framework for constrained optimization problems.

Part II

Unconstrained SCOOP formulations of constrained optimization problems

Chapter 3

Beyond Feasibility: The Hidden Value of Infeasible Solutions

Contents

Chapter Contributions

C1: Analysis of penalty-based QUBO formulations revealing that: (1) optimal penalty settings are instance-specific rather than problem-specific, (2) penalties that support finding optimal solutions may not necessarily support finding near-optimal solutions, and (3) strict enforcement of feasibility requires prohibitively large penalty values that impact solution quality and scalability.

C2: Introduction of a penalty-free encoding strategy and its cost Hamiltonian for the constrained optimization problem MINIMUM VERTEX COVER using the MAXIMUM PROFIT COVER problem

This chapter introduces our first problem of study, MINIMUM VERTEX COVER. The MINIMUM VERTEX COVER problem is an NP-hard constrained combinatorial optimization problem and is well studied through the lens of classical complexity theory [Garey and Johnson, 1979] and parameterized complexity [Downey et al., 1999c]. Vertex cover has numerous applications in scheduling, network design, resource allocation, and conflict resolution [Stege, 1999] Quantum computing offers promising avenues to tackle optimization problems, especially classically intractable combinatorial optimization problems. In this chapter, we explore the MINIMUM VERTEX COVER problem and its relationship with the MAXIMUM PROFIT COVER problem [Stege et al., 2002a]. We also discuss how to formulate these problems as

QUBO problems, which can be solved using quantum annealers or quantum approximate optimization algorithms (QAOA) [Angara et al., 2025a].

This chapter is organized as follows: We begin by defining the MINIMUM VERTEX COVER problem and its cost Hamiltonian in Section 3.1. We then discuss penalty-based approaches to vertex cover in Section 3.2, where we analyze the effect of penalty parameters on the solution space. In Section 3.3, we introduce the MAXIMUM PROFIT COVER problem, which is a penalty-free version of the MINIMUM VERTEX COVER problem. We also show how to derive a vertex cover from a profit cover. Finally, we provide an example to illustrate the concepts discussed in this chapter.

3.1. Vertex Cover

A *vertex cover* of a graph[1] $G = (V, E)$ is a subset of vertices VC $\subseteq V$ such that every edge $uv \in E$ shares at least one vertex with VC. The objective is to find a vertex cover of minimum cardinality, referred to as *minimum vertex cover*.

3.1.1. MINIMUM VERTEX COVER

We define the problem MINIMUM VERTEX COVER (MINVC) for a graph $G = (V, E)$, where VC $\subseteq V$, as follows.

$$\begin{aligned} &\text{Minimize:} && |\text{VC}| \\ &\text{Subject to:} && \text{for all edges } uv \in E, \quad u \in \text{VC or } v \in \text{VC} \end{aligned}$$

Any VC $\subseteq V$ that satisfies this constraint is a vertex cover. Any VC that optimally satisfies the objective function is a *minimum vertex cover*. Note that the decision version[2] is fixed-parameter tractable when parameterized by the size of the vertex cover to be determined [Downey et al., 1999a]; its fastest known fixed-parameter algorithm has a time complexity of $O^*\left(1.25284^k\right)$ [Harris and Narayanaswamy, 2024] where k is the size of the desired solution.

[1] All graphs in this chapter are simple undirected graphs

[2] The decision version of MINIMUM VERTEX COVER of MINIMUM VERTEX COVER of MINIMUM VERTEX COVER is known to be an NP-complete problem [Garey and Johnson, 1979].

While there is a 2-approximation algorithm for MINVC that is based on the idea of a maximal matching, when assuming that the Unique Games Conjecture holds, MINVC is hard to approximate within a factor of smaller than 2 [Garey and Johnson, 1979]. Generally, MINVC is known to be APX-complete [Papadimitriou and Yannakakis, 1988]. The best known approximation factor for MINVC is $2 - \Theta\left(1/\sqrt{\log |V|}\right)$ [Karakostas, 2009].

Moreover, the decision version asks, when given $G = (V, E)$ and a positive integer k, whether or not there exists a subset VC $\subseteq V$ of size $|VC| \leq k$.

Cost Hamiltonian for MINIMUM VERTEX COVER

The QUBO problem provides a framework for representing various combinatorial optimization problems in an unconstrained manner. Many *unconstrained* problems can be represented as QUBO problems using quadratic and linear terms involving binary variables. For constrained problems, *penalty parameters* are introduced to accommodate the constraints. Assuming a minimization routine, they add a cost to solutions that violate the constraints, steering the algorithm towards feasible solutions. Solving constrained problems with QAOA requires fine-tuning of penalty parameters in addition to optimizing the variational parameters, to be able to maneuver the cost Hamiltonian landscape. The choice and strength of penalty parameters can significantly impact QAOA performance [Mirkarimi et al., 2024].

The cost Hamiltonian for MINVC in the form of a QUBO can be defined as per the specifications outlined in [Lucas, 2014]. Given VC $\subseteq V$, and $v \in V$, let x_v be a binary variable whose value is 1 if v is included in the vertex cover VC ($v \in$ VC), and 0 otherwise. The constraint that every edge $uv \in E$ has at least one of its vertices in the subset VC is encoded by the following equation:

Cover Constraint encoding Hamiltonian:

$$\hat{H}_E^{VC}(\vec{x}) = \sum_{uv \in E} (1 - x_u)(1 - x_v) = \begin{cases} 1 & \text{if } x_u = x_v = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Given edge $uv \in E$, if both x_u and x_v are assigned to be 0, then uv is uncovered and therefore uv violates the constraint that VC is a vertex cover. This is encoded by the Cover Constraint encoding Hamiltonian. A trivial vertex cover arises when all $x_v \in V$ are set to 1. The objective of minimizing the number of vertices in the subset leads to the following Hamiltonian:

Vertex Cover set size Hamiltonian:

$$\hat{H}_V^{VC}(\vec{x}) = \sum_v x_v \tag{3.2}$$

The total cost Hamiltonian for MINVC is then given by the sum of the edge and vertex costs, with penalty parameters A and B:

$$\hat{H}_{\text{VC}}(\vec{x}) = A \cdot \hat{H}_E^{VC}(\vec{x}) + B \cdot \hat{H}_V^{VC}(\vec{x}). \tag{3.3}$$

The penalty parameters are set with $0 < B < A$, with $A, B \in \mathbb{R}$ [Lucas, 2014] and are used to balance the edge and vertex costs. The penalty value of the edge encoding constraint A must be larger than the vertex penalty B to ensure that feasible solutions (vertex covers) have lower costs than infeasible ones (the bigger multiplier A goes with the

constraints, and the smaller multiplier B relates to the set size). The penalties in the cost Hamiltonian can be alternatively expressed using a single parameter $\lambda = \frac{A}{B}$, commonly referred to as the Lagrange or Lagrangian parameter in optimization theory [Arfken et al., 2013, Saleem et al., 2023]. This allows us to rewrite the cost Hamiltonian as:

$$\hat{H}_{\text{VC}}(\vec{x}) = \lambda \cdot \hat{H}_E^{VC}(\vec{x}) + \hat{H}_V^{VC}(\vec{x}). \tag{3.4}$$

The penalty parameters are typically set to ensure that the cost of a feasible solution is less than the cost of an infeasible solution. The choice of these parameters can significantly influence both the quality of solutions obtained and the probability of finding optimal or near-optimal solutions through QAOA, i.e., the trade-off between solution feasibility and quality. The cost function is designed to minimize the number of vertices in the cover while ensuring that all edges are covered. The following section discusses in detail the effectiveness of penalty parameters on encoding and solving constrained combinatorial optimization problems.

3.2. Analysis of Penalty-Based Approaches to Vertex Cover

The effectiveness of penalty-based formulations of constrained combinatorial optimization problems heavily depends on the choice of penalty parameters. The QUBO formulation, while widely adopted, presents significant challenges when encoding constrained combinatorial optimization problems. The conventional approach to handling constraints involves incorporating them into the objective function as quadratic penalty terms, weighted by penalty parameters. This transformation effectively converts the constrained optimization problem into an unconstrained one, where constraint violations incur additional costs in the objective function. This approach has several limitations:

1. There is no distinction between *hard* and *soft* constraints when using QUBOs to encode constrained problems [Wilson et al., 2022]. Hard constraints are requirements that must be strictly satisfied for a solution to be considered feasible. Soft constraints, in contrast, are preferences that can be violated while maintaining solution feasibility, though such violations may reduce solution quality. The QUBO formulation's limitation is that it treats all constraints uniformly through penalty terms, without distinguishing between these fundamentally different types of constraints [Wilson et al., 2022]. In the context of the MinVC problem, the constraints are hard constraints, meaning that any solution that is not a vertex cover is infeasible. Therefore, all solutions that do not satisfy the constraint of being a vertex cover must have a higher cost than the feasible solutions.

2. There needs to be a balance between feasibility and solution quality as lower penalties allow better exploration but risk infeasible solutions whereas higher penalties may restrict the exploration of the solution space [Roch et al., 2023].

3. Penalty-based approaches can be effective in encoding constraints to find optimal solutions, however, they may not be suitable to find near-optimal solutions (which is important, for example, when finding the optimal solution may be impossible or impractical).

4. Linear constraints that are encoded as quadratic penalty terms may necessitate additional (auxiliary) variables to ensure that the constraints are satisfied. This can lead to an increase in the number of qubits required to represent the problem, which can be a significant limitation for quantum annealers and QAOA implementations [Gabbassov et al., 2023].

5. Quadratic penalty terms can lead to higher connectivity between the problem variables resulting in deeper quantum circuits [Gabbassov et al., 2023].

6. Some technically feasible solutions provide limited practical value. For example, trivially selecting all of the vertices in a vertex cover is feasible, however, it is not an interesting solution.

These limitations are particularly significant considering the probabilistic nature of quantum computation and the approximate nature of QAOA. Furthermore, quantum computational implementations in the near term due to hardware constraints require careful consideration of circuit depth, including the two-qubit gate depth (e.g. CNOT or CZ) to maintain qubit fidelity and solution accuracy. As a variational algorithm, QAOA provides approximate solutions whose quality depends on the number of alternating layers (circuit depth), the effectiveness of the classical optimizer, and the structure of the cost function landscape. This approximation aspect becomes particularly relevant when dealing with constrained problems, where solution feasibility must be balanced against optimization quality.

Table 3.1 illustrates how penalty parameters affect solution feasibility and optimality for the simplest case of MinVC: a graph with a single edge. Blue vertices belong to the vertex cover and white vertices are the ones which do not belong to the vertex cover. Recall the bigger multiplier A goes with the constraints, and the smaller multiplier B relates to the set size. For this minimal example, we observe how the penalty values A and B (see Eqn. 3.3) influence whether low cost solutions maintain vertex cover constraints.

Table 3.1.: The impact of penalty parameters on edge coverage for MINIMUM VERTEX COVER. For a single edge, the first row shows an infeasible solution with cost A (no vertices selected, edge uncovered). The second and third rows show feasible solutions with cost B (one vertex selected, edge covered). The last row shows a suboptimal solution with cost $2B$ (both vertices selected). To maintain feasibility, cover constraint penalty parameter A must exceed set size penalty parameter B to ensure single-vertex solutions are preferred over uncovered edges.

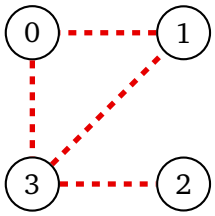
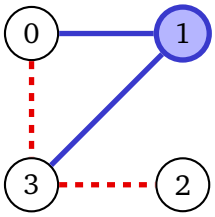
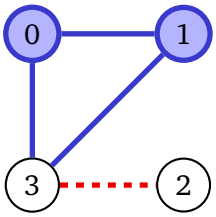
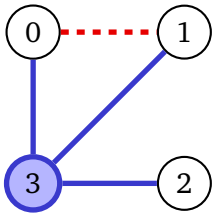
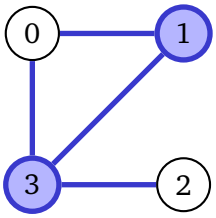
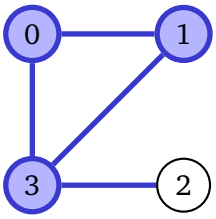
Graph	Cost	Is Vertex Cover?
	A	No
	B	Yes
	B	Yes
	2B	Yes

Table 3.2 illustrates the impact of penalty parameters on solution costs for a sample four-node graph with four edges. The penalty parameters must satisfy $A > B > 0$ [Lucas, 2014] to ensure feasible solutions have lower costs than infeasible ones. More specifically, to guarantee feasibility in MINVC:

- The maximum cost of a feasible solution is bounded by $B|V|$ (when all vertices are selected)
- The minimum cost of an infeasible solution is A (when a single edge is uncovered)
- Therefore, we must have $A > B|V|$ to ensure *all* feasible solutions have lower cost than infeasible ones.

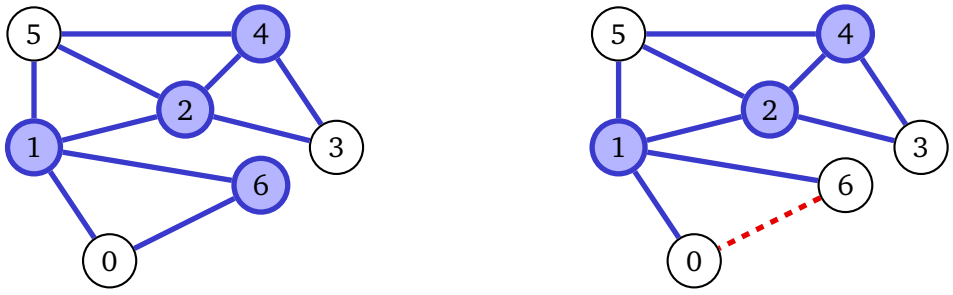
For the single-edge case shown in Table 3.1, this means $A > 2B$ since the maximum feasible solution cost is $2B$ (both vertices selected). This ensures the infeasible solution with cost A (no vertices selected) will always have higher cost than any feasible solution. The penalty strength can affect the number of near-optimal solutions (i.e., solutions that are not optimal but may be of interest). For the graph shown in Table 3.2, $A > 4B$. In practice, this bound is overly conservative as it guarantees feasibility even for trivial solutions (such as selecting all vertices or all but one vertex). Such solutions, while feasible, provide limited practical value and unnecessarily expand the solution space. Therefore, a more nuanced approach to penalty parameter selection can help focus the search on more meaningful regions of the solution space. However, when this bound is not satisfied, the cost hierarchy between feasible and infeasible solutions may be compromised. As illustrated in Table 3.2, the infeasible solution VC $= \{3\}$ has a lower cost than the feasible solution VC $= \{0, 1, 3\}$. This raises a fundamental challenge in penalty parameter selection: determining the optimal threshold that discriminates between high-quality feasible solutions

Table 3.2.: Impact of penalty parameters on solution costs for a four-node graph with four edges. Each row shows a different vertex selection pattern, with associated costs calculated using various penalty parameter combinations (A and B). Red edges indicate uncovered edges (contributing cost A), while blue-colored vertices indicate selected vertices (contributing cost B). This demonstrates how penalty strength affects the relative costs between feasible and infeasible solutions.

Graph	Cost	Is Vertex Cover?	$A = 3$ $B = 2$	$A = 3$ $B = 1$	$A = 4$ $B = 2$
	$4A$	No	12	12	16
	$2A + B$	No	8	7	10
	$A + 2B$	No	7	5	8
	$A + B$	No	5	4	6
	2B	Yes	4	2	4
	3B	Yes	6	3	6

and technically feasible but unproductive solutions. The boundary between these categories is often instance-dependent and cannot be universally prescribed through penalty parameters alone.

Now, consider the infeasible solution VC $= \{3\}$ shown in Table 3.2. While technically violating the vertex cover constraints, this solution exhibits promising structural properties: it leaves only a single edge uncovered while using minimal vertices. Such near-feasible solutions are valuable for constructing optimal vertex covers, suggesting that strict adherence to feasibility might unnecessarily exclude informative intermediate solutions.



(a) A feasible and optimal vertex cover

(b) An infeasible vertex cover

Figure 3.1.|: a) A vertex cover with a calculated cost of 8 (using $\lambda = 1.5$). b) The second-best solution, which is infeasible. For this particular graph, a higher penalty is needed to ensure vertex cover constraints are met.

The penalty parameters can also lead to a situation where the cost of the second-best solution is infeasible. This is particularly problematic when the goal is to find near-optimal solutions. Consider Eqn. 3.3 for penalty parameters to be $A = 3$ and $B = 2$ (i.e., $\lambda = 1.5$). The cost of the feasible and optimal vertex cover (in Fig. 3.1a) is 8. However, for this choice of penalty parameters, the cost of the second best solution is an infeasible solution—depicted in Fig. 3.1b—with a cost of 9. Notably, there exists no feasible vertex cover with a cost between 8 and 9. This is because adding any additional vertex to the optimal cover would increase the cost by at least 2, resulting in a minimum cost of 10 for the next best feasible solution. This shows that there exists an additional tier of challenges when it comes to satisfactorily represent near-optimal solutions using penalties.

3.2.1. Comparing summed probabilities using penalty-term formulations

Fig. 3.2 below shows four *summed probability* plots for the MinVC problem for eight nodes with varying edge probabilities ($d \in \{0.1, 0.3, 0.5, 0.8\}$) and varying penalty parameters. Here, summed probabilities are calculated by summing the probabilities of finding an optimal solution for each graph type.

Penalties are varied with different values for the term A, keeping $A \geq 2$ and $B = 2$ constant. The plots illustrate that no single penalty consistently outperforms others across all graph types. The optimal penalty varies depending on the specific characteristics of each graph. Even within a single class of graphs, certain penalties may perform better for

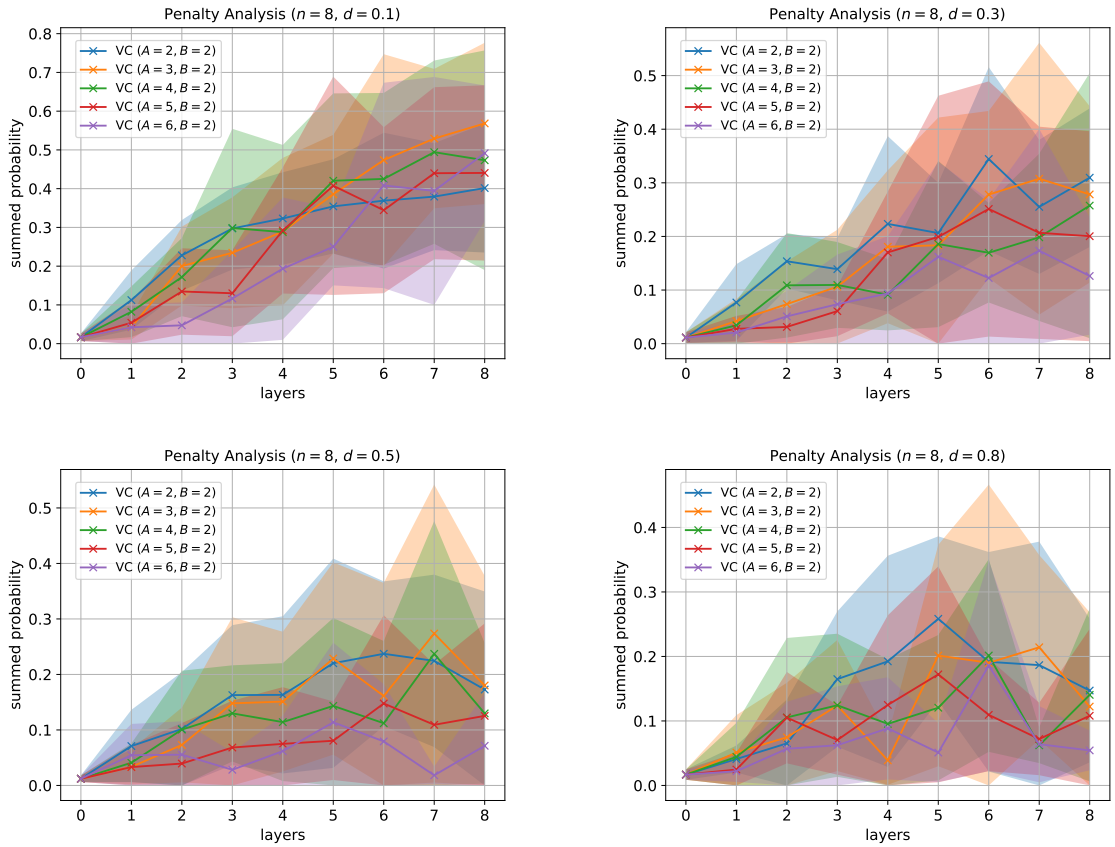


Figure 3.2.|: The plots above show summed probabilities for constrained optimization problems using penalty term formulations for edge densities, $d \in \{0.1, 0.3, 0.5, 0.8\}$ for graphs with eight nodes averaged over ten graphs. It is evident that no single penalty value consistently outperforms the others across all graph types. The optimal penalty varies depending on the specific characteristics of each graph. Even within a single class of graphs, certain penalties may perform better for some graphs while being less effective for others.

some graphs while being less effective for others. This highlights the challenge of selecting appropriate penalty parameters in penalty-based approaches to constrained combinatorial optimization problems.

The next section introduces the MAXIMUM PROFIT COVER problem and its relevance to the MINVC problem. In Chapter 4, we present an approach to generalize this method to other constrained combinatorial optimization problems. Chapter 5 discusses quadratic formulations and Chapter 6 discusses higher-order formulations of various problems based on our approach.

3.3. MAXIMUM PROFIT COVER

Given the limitations of penalty-based approaches, we propose a new approach that aims to solve the MINVC problem using a different *unconstrained* cost function. The new cost Hamiltonian is based on the MAXIMUM PROFIT COVER problem first introduced in [Stege et al., 2002b], which allows us to explore the solution space more effectively and find near-optimal solutions without being constrained by strict feasibility requirements. This raises a natural question: can we systematically discover unconstrained versions for other constrained optimization problems while maintaining similar solution quality guarantees? This question motivates the development of our SCOOP framework, which we present in the next chapter. Consider the MINIMUM VERTEX COVER problem. Suppose we relax the requirement that the subset of vertices that represents a solution must be a vertex cover. If instead we consider the extent to which this subset covers edges and its closeness to being a vertex cover (i.e., the more edge coverage the better), then we obtain the graph problem MAXIMUM PROFIT COVER [Stege et al., 2002b]. For a subset $PC \subseteq V$ in G, the number of edges covered by vertices from PC is the *gain*, and the number of vertices spent to cover these edges is considered the *loss*. The profit of $PC \subseteq V$ for a graph $G = (V, E)$ is then defined as *profit* $=$ gain $-$ loss.

We define the problem MAXPC for a graph $G = (V, E)$ where PC $\subseteq V$.

Maximize: profit $\mathfrak{p}_{\text{PC}}$, where

$$\mathfrak{p}_{\text{PC}} = |E_{\text{PC}}(G, \text{PC})| - |\text{PC}|$$

Here, $E_{\text{PC}}(G, \text{PC})$ represents the edges in G covered by PC:

$$E_{\text{PC}}(G, \text{PC}) = \{uv \in E : u \in \text{PC or } v \in \text{PC}\}$$

We call a subset PC with maximum profit also a *maximum profit cover*. Like MINVC, MAXPC is NP-hard [Stege et al., 2002b].

Cost Hamiltonian for MAXIMUM PROFIT COVER

The binary variables for the QUBO for the MAXPC problem are similar to MINVC. Each x_v is a binary variable with value 1 if v is included in the profit cover PC, and 0 otherwise. The edge and vertex cost Hamiltonians for MAXPC are given as follows:

Cost Hamiltonian for MAXPC

Edge cost:

$$\hat{H}_E^{PC}(\vec{x}) = \sum_{uv \in E} (x_u + x_v - x_u x_v)$$

Vertex cost:

$$\hat{H}_V^{PC}(\vec{x}) = \sum_v x_v$$

The total cost to be maximized for MAXPC is

$$\hat{H}_{\text{PC}}(\vec{x}) = \hat{H}_E^{PC}(\vec{x}) - \hat{H}_V^{PC}(\vec{x}) \tag{3.5}$$

While the definitions for $\hat{H}_{\text{VC}}(\vec{x})$ and $\hat{H}_{\text{PC}}(\vec{x})$ may appear similar, $\hat{H}_{\text{PC}}(\vec{x})$ has no penalty parameters that need to be set, since every binary variable assignment corresponds to a feasible solution.

3.3.1. Finding minimum vertex covers via maximum profit covers

We next point out an important relationship between the decision versions of both problems, which guarantees that we can derive a vertex cover VC for any subset of profit p while guaranteeing the profit of the obtained vertex cover VC.

Theorem 1. *[Stege et al., 2002b] For any graph $G = (V, E)$, G has a vertex cover $VC \subseteq V$ of size k if and only if G has a subset $PC \subseteq V$ with profit $\mathfrak{p}_{PC} = |E| - k$.*

Proof. On the one hand, determining the profit $\mathfrak{p}_{\text{PC}}$ of a vertex cover $\text{VC} \subseteq V$ for G results in $\mathfrak{p}_{\text{PC}} = |E_{\text{ADJ}}(G, \text{VC})| - |\text{VC}|$, where $E_{ADJ}(G, \text{VC})$ are the edges covered by VC. Since VC is a vertex cover, $E_{\text{ADJ}}(G, \text{VC}) = E$ and therefore $\mathfrak{p}_{\text{PC}} = |E| - k$.

On the other hand, consider a subset $PC \subseteq V$ with a profit $\mathfrak{p}_{PC} = |E| - k$. First, consider a case where $E_{\text{ADJ}}(G, \text{PC}) = E$, then $|PC| = k$ and thus PC is a vertex cover of size k. Second, if $E_{\text{ADJ}} \neq E$, not all edges in G are covered by the vertices in PC. In this case we can obtain a vertex cover of size at most $|E| - \mathfrak{p}_{\text{PC}}$ by covering the remaining edges in $E \setminus E_{\text{ADJ}}(G, \text{PC})$ as follows.

(*) Pick an edge $uv \in E \setminus E_{\text{ADJ}}(G, \text{PC})$. To cover edge uv, add one of the vertices, say u, to PC.

Note that the profit of the update set PC remains $\mathfrak{p}_{\text{PC}}$ since one more edge, uv, is covered with one additional vertex, u.

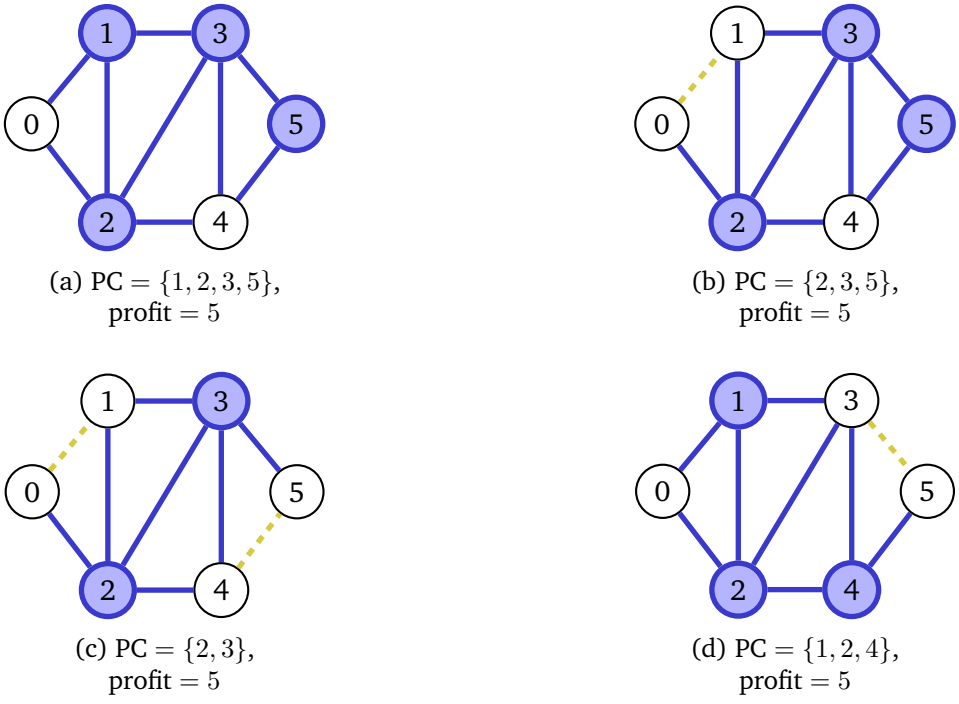
We repeat (*) as long as $E \setminus E_{\text{ADJ}}(G, \text{PC}) \neq \emptyset$. □

Theorem 1 implies that, given a maximum profit cover PC for a graph G, we can obtain a minimum vertex cover for G using the procedure given in the proof. To convert profit cover results to vertex cover, we apply Alg. 1 based on Theorem 1.

Algorithm 1: Converting Profit Cover to Vertex Cover

Require: Graph G, Solution PC
  Let E_{unc} be the set of uncovered edges in G
  for each edge $uv \in E_{\text{unc}}$ **do**
    if $u \notin$ PC and $v \notin$ PC **then**
      Add u to PC
    end if
  end for

3.3.2. An Example



(a) PC $= \{1, 2, 3, 5\}$, profit $= 5$

(b) PC $= \{2, 3, 5\}$, profit $= 5$

(c) PC $= \{2, 3\}$, profit $= 5$

(d) PC $= \{1, 2, 4\}$, profit $= 5$

Figure 3.3.|: An illustration of different profit covers with optimal profit. The graph shown in Fig. 3.3a is also a minimum vertex cover. Graphs in Fig. 3.3b, 3.3c, and 3.3d can be converted into minimum vertex covers using classical post-processing (cf. Sec. 7.2.4).

Fig. 3.3 shows a graph with different choices of subsets of vertices as profit covers (in red or shaded). The selected vertices in Fig. 3.3a form a minimum vertex cover (of size 4) as well as a maximum profit cover (with $\mathfrak{p}_{\text{PC}} = 5$). Fig. 3.3b, 3.3c, and 3.3d are not feasible vertex covers but are feasible profit covers with maximum profit $\mathfrak{p}_{\text{PC}} = 5$.

The maximum profit covers in Fig. 3.3b, 3.3c, and 3.3d can be converted into a minimum vertex cover without changing the profit by adding vertices associated with uncovered edges using classical post processing based on Alg. 1.

MAXIMUM PROFIT COVER (MAXPC) can be utilized to solve MINIMUM VERTEX COVER (MINVC) by post-processing solutions of profit cover instances into vertex covers while maintaining the quality of the solution. Since MAXIMUM PROFIT COVER is an unconstrained problem, it can be written as a penalty-free unconstrained binary form. As MINVC, MAXPC can be represented as a QUBO.

Building upon the insights from this chapter, Chapter 4 introduces SCOOP—a systematic framework for transforming constrained optimization problems into their unconstrained counterparts while preserving solution quality characteristics.

3.4. Chapter Summary

This chapter examined the challenges of solving constrained optimization problems through the lens of the MINIMUM VERTEX COVER (MINVC) problem, introducing a novel penalty-free approach. We began by analyzing traditional penalty-based QUBO formulations, revealing several fundamental limitations including the challenge of balancing feasibility and solution quality through penalty parameters, and the inability to distinguish between hard and soft constraints in the problem formulation. We illustrated how penalty parameters affect solution hierarchies and demonstrated cases where infeasible solutions provide valuable insights for constructing optimal vertex covers. This chapter marks a fundamental shift in approaching constrained optimization problems: rather than enforcing constraints through penalties, we transform problems to naturally capture both optimal and near-optimal solutions while maintaining solution quality characteristics. This relationship between vertex covers and profit covers establishes the foundation for our SCOOP framework, as described in Chapter 4, setting the stage for generalizing this approach to other constrained optimization problems.

Chapter 4

SCOOP Framework

Contents

Chapter Contributions

C3: Development of the SCOOP framework: A systematic approach for transforming constrained optimization problems into their unconstrained counterparts ensuring that the resulting problem pairs are ***s**olution-enhanceable, **c**onstrained-unconstrained, **o**bjective-function-compatible **o**ptimization **p**roblem twins*.

In this chapter, we formalize a blueprint that guides us in deriving an unconstrained Combinatorial Optimization Problem (COP) P_U, from a constrained one, P_C. P_U possesses the necessary properties to obtain optimal and near-optimal solutions using the vanilla QAOA approach for P_U, followed by efficient post-processing. In Sec. 3.3.1, we derived the unconstrained problem Maximum Profit Cover from the constrained problem Minimum Vertex Cover. This chapter is organized as follows. We first introduce the SCOOP framework, which describes how to derive an unconstrained COP P_U from a constrained COP P_C [Angara et al., 2025b].

4.1. Formulation of the SCOOP Framework

In general terms, a combinatorial optimization problem P takes as input, an instance x from its input domain $\mathcal{I}$ (e.g., a graph from the set of all simple undirected graphs). For a given $x \in I$, a solution is $s \in \text{Sol}(x)$, where $\text{Sol}(x)$ denotes the general set of possible solutions for x (e.g., if the instance is a graph $G = (V, E)$ and every solution is a subset of vertices, then $\text{Sol}(G) = \mathcal{P}(V)$, the powerset of V). The objective function of P for an instance $x \in I$ can be described as $\texttt{obj}^x : \text{Sol}(x) \longrightarrow \mathbb{Q}$ (e.g., if the measure of the solution is size, and a solution to a graph problem is a vertex subset, say $V' \subseteq V$, then $\texttt{obj}^x(s) =$

$\texttt{obj}^G(V') = |V'|$). A typical format for describing a COP P: For $x \in I$ and $s \in \text{Sol}(x)$,

$$\text{Optimize:} \quad \texttt{obj}^x(s)$$

Note that in the case of an unconstrained COP, any $s \in \text{Sol}(x)$ is a feasible solution. When talking about a COP that is unconstrained, we may call it P_U. If instead the COP considered is a *constrained* COP, then the set of feasible solutions is a subset of $\text{Sol}(x)$, denoted $\text{Sol}_C(x)$ where $\text{Sol}_C(x) \subset \text{Sol}(x)$.

A typical format for describing a constrained COP P_C is: for $x \in I$ and $s \in \text{Sol}(x)$,

$$\begin{aligned} &\text{Optimize:} \quad \texttt{obj}^x(s) \\ &\text{Subject to:} \quad s \in \text{Sol}_C(x) \end{aligned}$$

Before describing the process of how to derive an unconstrained COP P_U from a given constrained COP P_C, we define a relationship between the problems that is sufficient to obtain optimal and near-optimal solutions using the vanilla QAOA approach for P_U, followed by efficient post-processing.

We say that a constrained COP P_C with input domain I_C, feasible solutions Sol_C, and objective function $\texttt{obj}_C$, and an unconstrained COP P_U with input domain I_U, solutions $\text{Sol}_U = \text{Sol}$, and objective function $\texttt{obj}_U$, are ***s**olution-enhanceable, **c**onstrained-unconstrained, **o**bjective-function-compatible **o**ptimization **p**roblem twins* or *SCOOP twins* if the following conditions hold:

1. **Identical Input Domains**: The problems share the same input domain I, i.e., $I = I_C = I_U$.
2. **Solution Containment**: for every instance $x \in I$, $\text{Sol}_C(x) \subseteq \text{Sol}_U(x)$. That is, each feasible solution for P_C is a solution for P_U.
3. **Objective Function Compatibility**: For any $x \in I$ there exists a polynomial-time computable function $f_x : \mathbb{Q} \to \mathbb{Q}$ such that for all $s \in \text{Sol}_C(x)$: $\texttt{obj}_C^x(s) = f_x(\texttt{obj}_U^x(s))$ and $\texttt{obj}_U^x(s) = f_x^{-1}(\texttt{obj}_C^x(s))$.
4. **Solution Enhance-ability**: For each $x \in I$, and each $s \in \text{Sol}_U(x)$, $s \in \text{Sol}_C(x)$ or there exists $s' \in \text{Sol}_C(x)$, where s' is computable in polynomial time from s. Furthermore, $\texttt{obj}_U^x(s')$ is at least as good as $\texttt{obj}_U^x(s)$ and $\texttt{obj}_C^x(s')$ is at least as good as $f_x(\texttt{obj}_U^x(s))$.

Fig. 4.1 outlines the steps to solve a constrained problem using the SCOOP framework. We next describe a process that may allow, given a constrained problem P_C, to derive an unconstrained problem P_U such that P_C and P_U are SCOOP twins. Let P_C be a constrained combinatorial optimization problem where for $x \in I$, a feasible solution $s \in \text{Sol}_C(x)$ has cost $\texttt{obj}_C^x(s)$.

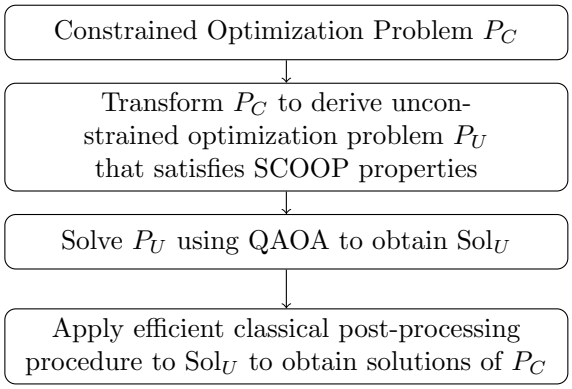


Figure 4.1.|: **SCOOP Framework**: This flowchart outlines the steps to solve a constrained optimization problem P_C using its unconstrained SCOOP twin P_U.

4.2. SCOOP Framework applied to MinVC

We discuss this using the example of Minimum Vertex Cover discussed in Chapter 3 and in [Angara et al., 2025a], and describe three steps as general guidance, before illustrating this framework on various problems in Chapters 5 and 6.

Returning to the examples considered in [Stege et al., 2002b, Angara et al., 2025a]: To obtain the unconstrained problem MaxPC from MinVC, observe that any subset PC $\subseteq V$ for the given graph $G = (V, E)$ is a solution for the MinVC problem as long as PC satisfies that it is a vertex cover for G, that is every edge in E is covered by a vertex in PC. To obtain an unconstrained problem, we characterize subsets of vertices by how many edges of the graph they cover minus the investment to achieve the amount of covered edges, |PC|. In other words, we develop a measure for the extent that a solution satisfies the constraint.

We can generalize this process as follows.

Step 1 Identify the constraints of P_C.

Step 2 Consider any infeasible solution s' for P_C. Identify a way to quantify the extent to which the solution satisfies the constraint, deriving $\texttt{obj}^x_U(.)$ by relating it to $\texttt{obj}^x_C(s')$.

Step 3 Develop P_U using the cost function $\texttt{obj}^x_U(.)$.

Step 4 Prove objective function compatibility.

Step 5 Prove solution enhance-ability.

Note that the existence of the SCOOP twin P_U for a given P_C yields NP-hardness for P_U as long as P_C is NP-hard.

Recall the problem Minimum Vertex Cover (MinVC) for a graph $G = (V, E)$, where VC $\subseteq V$, is defined as follows.

Minimize: |VC|
Subject to: (1) for all edges $uv \in E$, $u \in$ VC or $v \in$ VC

Any VC $\subseteq V$ that satisfies this constraint is a vertex cover. Any VC that optimally satisfies the objective function is a *minimum vertex cover*.

Here, $\texttt{obj}^G_{\text{VC}}(\text{VC}) = |\text{VC}|$. Solutions to the problem are subsets of vertices (i.e., $\text{Sol}(G) = |\text{VC}|$) that must satisfy the coverage constraint i.e., (1), describing $\text{Sol}_C(G) \subset \text{Sol}(G)$. To quantify how much a subset VC of vertices satisfies the constraint, when VC is not feasible, that is VC does not satisfy constraint (1), we say that an edge $(u, v) \in E$ is *covered* by VC if $u \in$ VC or $v \in$ VC. Let $E_{\text{PC}(G,\text{PC})} = \{e \in E \mid e \text{ is covered by PC}\}$. Wee can define $\texttt{obj}^G_{\text{PC}}(\text{PC}) = |E_{\text{PC}(G,\text{PC})}| - |\text{PC}|$.

Now, we are ready to define the problem MAXIMUM PROFIT COVER (MAXPC): for a graph $G = (V, E)$ and PC $\subseteq V$,

$$\text{Maximize:} \quad \text{profit } \mathfrak{p}_{\text{PC}} \text{ where } \mathfrak{p}_{\text{PC}} = \texttt{obj}^{G}_{\text{PC}}(\text{PC})$$

In Sec. 3.3.1 we proved the equivalence between MINVC and MAXPC, therefore indicating that they are indeed SCOOP twins.

4.3. Chapter Summary

This chapter introduced the SCOOP framework—a systematic approach for transforming constrained optimization problems into their unconstrained counterparts. We formalized the concept of SCOOP twins, which are problem pairs that satisfy four key properties: *identical input domains*, *solution containment*, *objective function compatibility*, and *solution enhance-ability*.

Using the MINIMUM VERTEX COVER problem as our primary example, we demonstrated how to systematically derive an unconstrained equivalent problem MAXIMUM PROFIT COVER. The framework preserves solution quality while eliminating the need for penalty parameters, providing a natural way to handle certain kinds of constraints in quantum optimization. In the following two chapters, we document that our framework applies various NP-hard problems, including MAXIMUM INDEPENDENT SET, MAXIMUM CLIQUE, MAXIMUM SET PACKING, MAX3SAT, MINIMUM DOMINATING SET, MINIMUM MAXIMAL MATCHING, and MINIMUM SET COVER. We also show that the SCOOP framework can be applied to constrained combinatorial optimization problems that can be solved in polynomial time, such as MINIMUM EDGE COVER and MAXIMUM MATCHING.

Chapter 5

Quadratic Problem Formulations

Contents

Chapter Contributions

C4: Application of the SCOOP framework to the problems: MAXIMUM INDEPENDENT SET, MAXIMUM CLIQUE, MAXIMUM SET PACKING

C5: Non-SCOOP QUBO formulations of the MAX3SAT problem

This chapter presents the SCOOP twins of selected constrained optimization problems that can be written as a QUBO.[1] For each problem, we state the constrained formulation and the unconstrained equivalent, as well as their respective cost operators in the form of a QUBO, and their relationships. The first two problems are MAXIMUM INDEPENDENT SET (MAXIS) and MAXIMUM CLIQUE.

[1] Although higher-order formulations can be converted into QUBOs by quadratization, the problems addressed in this chapter do not inherently require such a transformation

The MinVC, MaxIS and MaxCl problems are known to be NP-hard [Garey and Johnson, 1979]. Reviewing the relationship between the three problems shows that they permit an efficient translation of solutions for instances between the problems. Therefore, regardless which of MinVC, MaxIS and MaxCl is the problem under consideration, algorithmic explorations for one of them are useful and applicable to all of them.

MinVC, MaxIS and MaxCl are equivalent in the sense that they are pairwise bi-directionally polynomial-time reducible to each other: for any graph $G = (V, E)$ and its *complement graph* $G_c = (V, E_c)$ where $E_c = (V \times V) \setminus E$, VC $\subseteq V$ is a vertex cover for G if and only if $V \setminus$ VC is an independent set for G if and only if $V \setminus$ VC is a clique for G_c [Garey and Johnson, 1979].[2]

In Sec. 5.5.1 and 5.4 we showMax3SAT and Maximum Set Packing problems are constrained combinatorial optimization problems due to their relationship to Maximum Independent Set and thus can be tackled using our SCOOP approach.

5.1. Independent Set

5.1.1. Maximum Independent Set (MaxIS)

We define the problem Maximum Independent Set (MaxIS) for a graph $G = (V, E)$, where $IS \subseteq V$, as follows.

$$\begin{aligned} &\text{Maximize:} && |IS| \\ &\text{Subject to:} && \text{for all vertices } u, v \in IS, \quad uv \notin E \end{aligned}$$

We denote any subset of V that satisfies the constraint an *independent set*. Any largest independent set is also referred to as *maximum independent set*. Independent sets find applications in areas that require elements of a set to be uncorrelated (such as portfolio optimization or supply chain optimization) or in collision avoidance such as antenna placement strategies. MaxIS is APX-hard and Poly-APX-complete for general graphs [Bazgan et al., 2005]. Approximation algorithms with constant approximation factor exist for bounded-degree graphs [Berman and Fürer, 1994, de Berg et al., 2023]. In contrast to the fixed-parameter tractability result for the decision version of Minimum Vertex Cover, the decision version of Maximum Independent Set for general graphs is $W[1]$-complete when parameterized by the size of the independent set to be determined, but fixed-parameter tractable for planar graphs [Downey and Fellows, 2013b]. Inapproximability results state that MaxIS cannot be approximated within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, unless P = NP [Håstad, 1999, Zuckerman, 2006]. Exact algorithms for MaxIS include Robson's $O(2^{n/4})$ algorithm [Robson, 2001] and more recently a $O(1.1664^n)$-algorithm by Xiao and Nagamochi [Xiao and Nagamochi, 2017].

[2] The "\" symbol denotes the setminus operation.

Due to their close relationship, the above findings for MAXIMUM INDEPENDENT SET transfer to MAXIMUM CLIQUE, which is defined in the next section.

The cost Hamiltonian for MAXIS is defined as follows. Given $S \subseteq V$, and $v \in V$, let x_v be a binary variable whose value is 1 if v is included the independent set S ($v \in S$), and 0 otherwise.

The objective of maximizing the number of vertices in the subset leads to

$$\hat{H}_V^{IS}(\vec{x}) = B \sum_v x_v.$$

The constraint that there does not exist an edge $(u, v) \in E'$, with both $u, v \in S$ can be expressed as follows:

$$\hat{H}_E^{IS}(\vec{x}) = -A \sum_{uv \in E} x_u x_v.$$

Given edge $uv \in E$, if both x_u and x_v are assigned to be 1, then the edge constraint is violated (and therefore adds a penalty of $-A$ to the cost Hamiltonian). A trivial independent set arises when any one single node in x_v is set to 1.

The total cost for the MAXIS problem is

$$\hat{H}_{\text{IS}}(\vec{x}) = \hat{H}_E^{IS}(\vec{x}) + \hat{H}_V^{IS}(\vec{x}). \tag{5.1}$$

Note that this is formulated as a maximization problem. To convert it into a minimization problem, one can change the sign of the objective function, which allows us to maximize the original function indirectly.

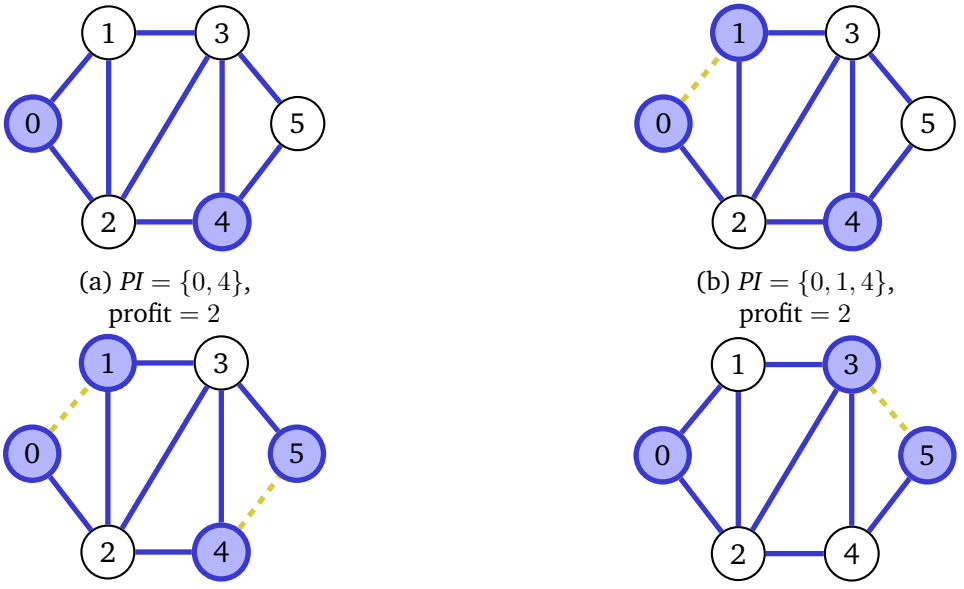
5.1.2. MAXIMUM PROFIT INDEPENDENCE (MAXPI)

Using the SCOOP framework defined in Chapter 4, we derive the unconstrained problem MAXIMUM PROFIT INDEPENDENCE or MAXPI and prove that they are SCOOP twins using Theorem 2.

Step 1: Identify constraints
To maintain feasibility, MAXIS requires that the selected subset *IS* $\subseteq V$ contains no adjacent vertices, thereby forming an independent set.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of vertices *violates independence* and its *closeness* to being an independent set (i.e., the fewer edges between vertices, the better).

Step 3: Develop objective function $\texttt{obj}_{PI}^{G}$
For a subset *PI* $\subseteq V$ in G,
$\texttt{obj}_{PI}^{G}(PI) = |PI| - |E_{\text{PI}}(G, PI)|$, where $E_{\text{PI}}(G, PI)$ is the set of edges with both endpoints in *PI*. We denote the value of the objective function $\texttt{obj}_{PI}^{G}(PI)$ as the *profit* $\texttt{p}_{PI}$.



(a) $PI = \{0, 4\}$, profit $= 2$

(b) $PI = \{0, 1, 4\}$, profit $= 2$

(c) $PI = \{0, 1, 4, 5\}$, profit $= 2$

(d) $PI = \{0, 3, 5\}$, profit $= 2$

Figure 5.1.|: An illustration of different profit independent sets with optimal profit. The graph shown in Fig. 5.1a is also a maximum independent set. Graphs in Fig. 5.1b, 5.1c, and 5.1d can be converted into maximum independent sets using classical post-processing (cf. Sec. 7.2.4).

We formulate the problem MAXIMUM PROFIT INDEPENDENCE (MAXPI) [Van Rooij, 2003] for a graph $G = (V, E)$ and $PI \subseteq V$:

$$\text{Maximize: profit } \mathfrak{p}_{\text{PI}}, \text{where } \mathfrak{p}_{\text{PI}} = \texttt{obj}_{PI}^{G}(PI)$$

Step 4: Objective function compatibility
From Theorem 2 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = k$ and $f_G^{-1}(k) = \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 2 describes a polynomial-time post-processing procedure that converts a solution *PI* for MAXPI to an independent set, i.e., a solution for MAXIS while preserving solution quality.

Cost Hamiltonian for MAXIMUM PROFIT INDEPENDENCE

The binary variables for the MAXPI problem are similar to MAXIS: each x_v is a binary variable with value 1 if v is included in the Profit Independence set *PI*, and 0 otherwise. The edge and vertex cost Hamiltonians for MAXPI are given as follows:

Cost Hamiltonian for MAXPI

Edge cost:

$$\hat{H}_E^{PI}(\vec{x}) = \sum_{uv \in E} (x_u x_v)$$

Vertex cost:

$$\hat{H}_V^{PI}(\vec{x}) = \sum_{v} x_v$$

The total cost that is maximized for MAXPI is

$$\hat{H}_{\text{PI}}(\vec{x}) = \hat{H}_V^{PI}(\vec{x}) - \hat{H}_E^{PI}(\vec{x}) \tag{5.2}$$

While $\hat{H}_{\text{IS}}(\vec{x})$ and $\hat{H}_{\text{PI}}(\vec{x})$ may appear similar, $\hat{H}_{\text{PI}}(\vec{x})$ has no penalty parameters that needs to be set, since every binary variable assignment corresponds to a feasible solution.

5.1.2.1. Finding maximum independent sets via maximum profit independent sets

Similar to the connection between the decision versions of MINVC and MAXPC, maximum independent sets can also be identified through profit independent sets. Fig. 5.1a shows a maximum independent set and a maximum profit cover (with profit $\mathfrak{p}_{\text{PI}} = 2$). Fig. 5.1b, 5.1c, and 5.1d show feasible (and maximum) profit independent sets that are infeasible independent sets. The following theorem shows the relationship between profit independence and independent sets which can then be subsequently used to convert profit independent sets to independent sets.

Theorem 2. *[Van Rooij, 2003] For any graph $G = (V, E)$, G has an independent set $IS \subseteq V$ of size k if a only if G has a subset $PI \subseteq V$ with profit $\mathfrak{p}_{PI} = k$.*

Proof. The proof is based on the relationship between vertex cover and independent set. The proof is divided into two parts:

($\Leftarrow$) First, we show that if G has an independent set $IS \subseteq V$ of size k, then there exists a subset $PI \subseteq V$ with profit $\mathfrak{p}_{\text{PI}} = k$. Let *IS* be an independent set of size k. We can define a subset $PI = IS$, which is also an independent set. The profit of *PI* is given by $\mathfrak{p}_{\text{PI}} = |PI| - |E_{\text{PI}}(G, PI)|$. Since *IS* is an independent set, $E_{\text{PI}}(G, PI) = \emptyset$, and therefore $\mathfrak{p}_{\text{PI}} = |IS| = k$.

($\Rightarrow$)

Consider a subset $PI \subseteq V$ with profit $\mathfrak{p}_{\text{PI}} = k$. We need to show that there exists an independent set $IS \subseteq V$ of size k. If $E_{\text{PI}}(G, PI) = \emptyset$, then *PI* is an independent set of size k. If $E_{\text{PI}}(G, PI) \neq \emptyset$, then not all vertices in *pi* are independent. In this case, we can obtain an independent set of size at least k by removing vertices from *PI* as follows.

(*) Pick an edge $uv \in E_{\text{PI}}(G, PI)$ with both endpoints in *PI*. Remove one of the vertices, say u, from *PI*. Note that the profit of the updated set *PI* remains $\mathfrak{p}_{\text{PI}}$ (or only increases) since removal of one vertex u from *PI* reduces $|PI|$, but also removes one or more edge(s) from $E_{\text{PI}}(G, PI)$, thus keeping the profit unchanged.

We repeat (*) as long as $E \setminus E_{\text{PI}}(G, PI) \neq \emptyset$. □

Theorem 2 implies that, given a maximum profit independence *PI* for a graph G, we can obtain a minimum vertex cover for G. To convert profit independence results to independent sets, we apply Alg. 2 based on Theorem 2.

Algorithm 2: Converting Profit Independent Set to Independent Set

Require: Graph $G = (V, E)$, Solution *PI*
  Let $E_{\text{conf}} \subseteq E$ be the set of edges with both endpoints in *PI*
  for each edge $uv \in E_{\text{conf}}$ **do**
    if $u \in PI$ and $v \in PI$ **then**
      Remove u from *PI*
    end if
  end for

5.2. Clique

The *clique* problem involves finding a subset of vertices in a graph such that every two vertices in the subset are adjacent, forming a complete subgraph. The clique problem has both constrained and profit variants, which we discuss below. Due to its relationship to vertex cover and independent set, applications discussed earlier can also be reformulated as clique problems. One interesting application area is that of molecular biology and drug discovery. The docking problem involves finding a correct docking pose of a small molecule ("ligand") to a protein and can be encoded using the clique problem [Wurtz et al., 2022].

5.2.1. Maximum Clique (MaxCl)

We define the problem Maximum Clique (MaxCl) for a graph $G = (V, E)$, where $\mathsf{Cl} \subseteq V$, as follows.

$$\begin{aligned} &\text{Maximize:} && |\mathsf{Cl}| \\ &\text{Subject to:} && \text{for all vertices } u, v \in \mathsf{Cl}, \quad uv \in E \end{aligned}$$

We denote any subset of V that satisfies the constraint a *clique*. Any largest clique is also referred to as *maximum clique*.

Lucas [Lucas, 2014] provides the following QUBO formulations for the decision version and optimization version of MaxCl. The decision version of the clique problem asks whether an undirected graph $G = (V, E)$ contains a complete subgraph (clique) of size K. This can be expressed as a QUBO problem using binary variables $x_v \in \{0, 1\}$, where $x_v = 1$ if vertex $v \in V$ is included in the potential clique.

The objective function is defined as follows:

$$H(\vec{x}) = A\left(K - \sum_{v \in V} x_v\right)^2 + B\left[\frac{K(K-1)}{2} - \sum_{(u,v) \in E} x_u x_v\right], \tag{5.3}$$

where $A, B > 0$ are penalty parameters.

The first term penalizes solutions that do not select exactly K vertices, while the second term penalizes pairs of selected vertices that are not connected by an edge. A ground state (i.e., minimizing configuration) with $H = 0$ exists if and only if a clique of size K exists in the graph.

To ensure correctness, the coefficients should satisfy $A > KB$, which ensures that over-selection of vertices does not compensate for missing edges.

The optimization version of the clique problem can be formulated as a QUBO model by combining binary selection variables for vertices with auxiliary binary variables to encode the clique size.

As before, we use $x_v \in \{0,1\}$ indicate whether vertex $v \in V$ is included in the candidate clique. To encode the size of the clique, introduce auxiliary binary variables $y_i \in \{0,1\}$ for $i = 2, \ldots, N$, where $N = |V|$, such that exactly one y_i is set to 1 and it indicates that the clique has size i.

The Hamiltonian is given by:

$$H(x,y) = A\left(1 - \sum_{i=2}^{N} y_i\right)^2 \tag{5.4}$$

$$+ A\left(\sum_{i=2}^{N} i y_i - \sum_{v \in V} x_v\right)^2 \tag{5.5}$$

$$+ B\left[\frac{1}{2}\left(\sum_{i=2}^{N} i y_i\right)\left(\sum_{i=2}^{N} i y_i - 1\right) - \sum_{(u,v)\in E} x_u x_v\right] \tag{5.6}$$

$$- C\sum_{v \in V} x_v, \tag{5.7}$$

where $A, B, C > 0$ are penalty weights.

- The first two terms (with coefficient A) ensure that exactly one y_i is active and that the number of selected vertices matches the chosen clique size.
- The third term (with coefficient B) penalizes the lack of required edges among selected vertices; a clique of size K should have $\frac{K(K-1)}{2}$ edges.
- The final term (with coefficient C) encourages inclusion of more vertices, thus maximizing the size of the clique.

To ensure correct encoding, we must choose penalty weights such that constraint violations are energetically more costly than any gain from increasing clique size. One such sufficient condition is:

$$A > NB \quad \text{and} \quad C < A - NB.$$

To reduce the number of auxiliary y_i variables, one may encode the clique size using $\lceil \log_2 N \rceil$ binary variables via a weighted sum.

Due to the relationship between independent set and clique, we use the following QUBO formulation for MaxCl which is derived due to its relationship with MaxIS:[3]

$$\hat{H}^{Cl}_{E_c}(\vec{x}) = -A \sum_{uv \in (V\times V)\setminus E} x_u x_v$$

$$\hat{H}^{Cl}_{V}(\vec{x}) = B \sum_{v} x_v$$

[3]Recall: E_c denotes the set of edges in the complement graph G_c of G

The total cost for MAXCL is

$$\hat{H}_{\text{MCl}}(\vec{x}) = \hat{H}^{Cl}_{E_c}(\vec{x}) + \hat{H}^{Cl}_{V}(\vec{x}). \tag{5.8}$$

5.2.2. MAXIMUM PROFIT CLIQUE (MAXPCL)

The problem MAXIMUM PROFIT CLIQUE (MAXPCL) [Scott, 2004] involves the relaxation of the definition of the concept clique to include less-than-complete sub-graphs.

We define the problem MAXIMUM PROFIT CLIQUE MAXPCL for a given graph $G = (V, E)$ where PCl $\subseteq V$, as follows.

Maximize: $\mathfrak{p}_{\text{Cl}}$, where

$$\begin{aligned}\mathfrak{p}_{\text{Cl}} &= |\text{PCl}| - |E_{\text{PCl}}(G, \text{PCl})| \\ &= |PI| - |E_{\text{PI}}(G_c, PI)|\end{aligned}$$

Here, $E_{\text{PCl}}(G, \text{PCl})$ represents are the edges in the complement graph G_c with both endpoints in PCl:

$$E_{\text{PCl}}(G, \text{PCl}) = \{uv \in (V \times V) \setminus E : u, v \in \text{PCl}\}$$

We call a subset PCl with maximum profit also a *maximum profit clique*. MAXPCL is NP-hard [Scott, 2004].

Cliques and independent sets are related: an independent set (*IS*) in a graph's complement (G_c) is equivalent to a clique in the original graph (G). This applies to Profit Independence and Profit Clique as well (cf. Fig. 5.2). Nevertheless, we derive the unconstrained problem MAXIMUM PROFIT CLIQUE (MAXPCL) and prove that they are SCOOP twins using Theorem 3.

Step 1: Identify constraints
To maintain feasibility, MAXCL requires that the selected subset Cl $\subseteq V$ contains no non-adjacent vertices, thereby forming a clique.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of vertices *violates clique* and its *closeness* to being a clique (i.e., the fewer non-adjacent pairs, the better).

Step 3: Develop objective function $\texttt{obj}^{G}_{PCl}$
For a subset PCl $\subseteq V$ in G,
$\texttt{obj}^{G}_{PCl}(\text{PCl}) = |\text{PCl}| - |E_{\text{PCl}}(G, \text{PCl})|$, where $E_{\text{PCl}}(G, \text{PCl})$ is the set of edges in the complement graph G_c with both endpoints in PCl. We denote the value of the objective function $\texttt{obj}^{G}_{PCl}(\text{PCl})$ as the *profit* $\mathfrak{p}_{\text{PCl}}$.

Step 4: Objective function compatibility
From Theorem 3 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = k$ and $f_G^{-1}(k) = \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 3 describes a polynomial-time post-processing procedure that converts a solution PCl for MAXPCL to a clique, i.e., a solution for MAXCL while preserving solution quality.

Theorem 3. *For any graph $G = (V, E)$, G has a clique Cl $\subseteq V$ of size k if and only if G has a subset PCl $\subseteq V$ with profit $\mathfrak{p}_{PCl} = k$.*

Proof.

($\Leftarrow$) First, we show that if G has a clique Cl $\subseteq V$ of size k, then there exists a subset PCl $\subseteq V$ with profit $\mathfrak{p}_{\text{PCl}} = k$. Let Cl be a clique of size k. The profit of PCl is given by $\mathfrak{p}_{\text{PCl}} = |\text{PCl}| - |E_{\text{PCl}}(G, \text{PCl})|$. Since Cl is a clique, $E_{\text{PCl}}(G, \text{PCl}) = \emptyset$ (no non-edges in complement graph), and therefore $\mathfrak{p}_{\text{PCl}} = |\text{Cl}| = k$.

($\Rightarrow$) Consider a subset PCl $\subseteq V$ with profit $\mathfrak{p}_{\text{PCl}} = k$. We need to show that there exists a clique Cl $\subseteq V$ of size k. If $E_{\text{PCl}}(G, \text{PCl}) = \emptyset$, then PCl is already a clique of size k. If $E_{\text{PCl}}(G, \text{PCl}) \neq \emptyset$, then not all vertices in PCl form a clique. We can obtain a clique of size at least k by removing vertices as follows:

(*) Pick a non-edge $uv \in E_{\text{PCl}}(G, \text{PCl})$ from the complement graph with both endpoints in PCl. Remove one of the vertices, say u, from PCl. Note that the profit remains $\mathfrak{p}_{\text{PCl}}$ since removing vertex u reduces |PCl| but also removes one non-edge from $E_{\text{PCl}}(G, \text{PCl})$, keeping the profit unchanged.

Repeat (*) until $E_{\text{PCl}}(G, \text{PCl}) = \emptyset$, yielding a clique. □

rem

Algorithm 3: Converting Profit Clique to Clique

Require: Graph $G = (V, E)$, Solution PCl
  Let E_{PCl} be the set of edges in the complement graph G_c with both endpoints in PCl
  for each edge $uv \in E_{\text{PCl}}$ **do**
    if $u \in$ PCl and $v \in$ PCl **then**
      Remove u from PCl
    end if
  end for

Cost Hamiltonian for MAXIMUM PROFIT CLIQUE

The binary variables for the MAXPCL problem are similar to MAXPI. x_v is a binary variable whose value is 1 if v is included in the profit clique set PCl, and 0 otherwise. The cost Hamiltonian for MAXPCL is given as follows:

Cost Hamiltonian for MAXPCL

Edge cost:

$$\hat{H}_{E_c}^{PCl}(\vec{x}) = \sum_{uv \in E_c} (x_u x_v)$$

Vertex cost:

$$\hat{H}_{V}^{PCl}(\vec{x}) = \sum_{v} x_v$$

The total cost that needs to be maximized for MAXPCL is

$$\hat{H}_{\text{PCl}}(\vec{x}) = \hat{H}_{V}^{PCl}(\vec{x}) - \hat{H}_{E_c}^{PCl}(\vec{x}) \tag{5.9}$$

5.3. Relationships among vertex cover, independent set, and clique and their profit variants

Fig. 5.2 illustrates the connections between the constrained and profit (unconstrained) variants of Vertex Cover, Independent Set, and Clique for a given graph G. This diagram visualizes how these problems relate to one another and to their profit equivalents. We summarize the relationships among all six problems in Fig. 5.2. Let $G = (V, E)$. Then G has a vertex cover of size k if and only if G has an independent set of size $|V| - k$ if and only if G's complement G_c has a clique of size $|V| - k$ if and only if G has a profit cover of profit $|E| - k$ if and only if G has a profit independent set of profit $|V| - k$ if and only if G_c has a profit clique of profit $|V| - k$.

The interrelated nature of these profit problems means that solving one can provide solutions for others through simple transformations. In particular, a solution to MAXPC can be transformed classically to obtain solutions for MAXVC, MAXIS, and MAXCL. This extends beyond these problems to other optimization problems such as MAXPSP and MAX3SAT, which we discuss in subsequent sections. This interconnected nature has important implications for our experimental evaluation: analyzing QAOA's performance on any one of these profit problems provides valuable insights into its effectiveness across the entire family of related problems.

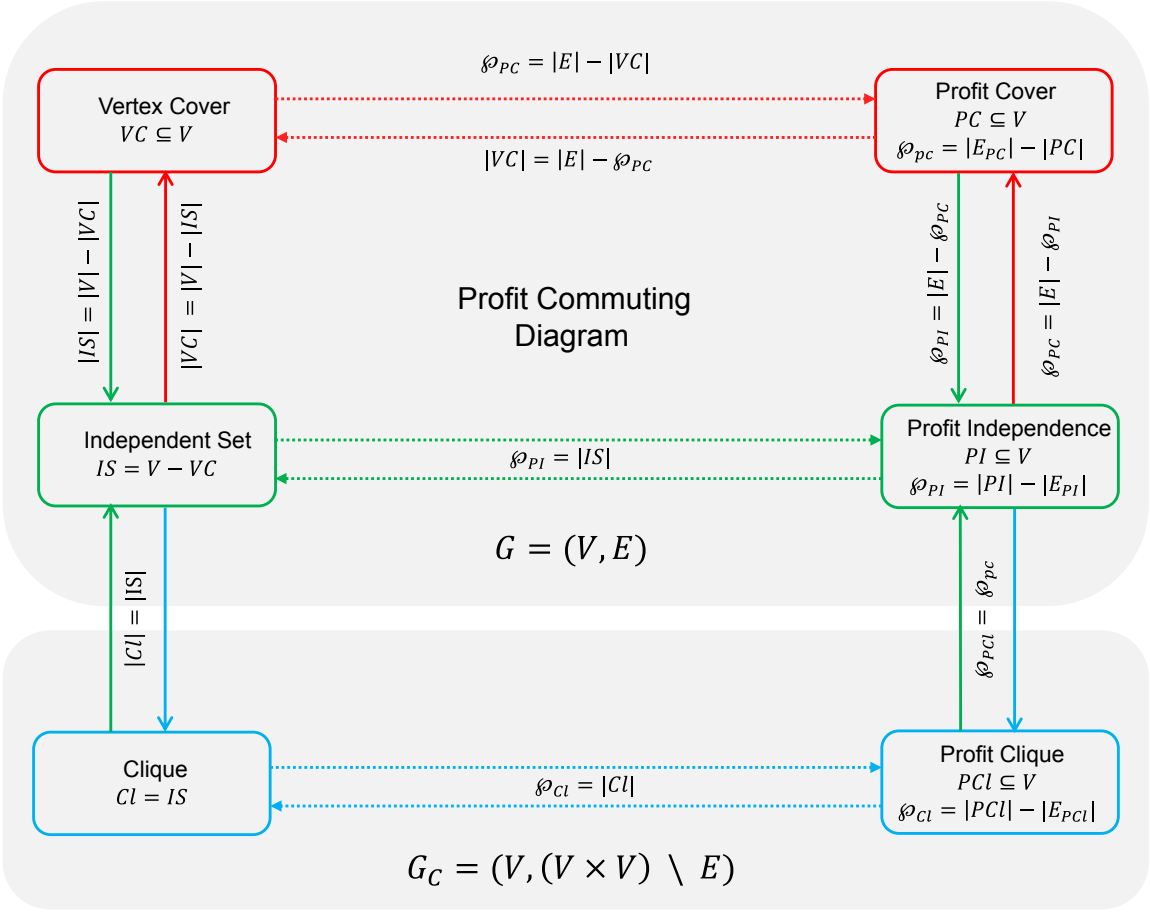


Figure 5.2.|: **Profit Commuting Diagram:** Relationship between constrained and profit (unconstrained) variants of vertex cover, independent set, and clique. Note that VC/VC and IS/PI correspond to the vertex/profit cover and independent set/profit independence of graph G whereas Cl/PCl refer to Clique/Profit Clique for the complement graph G_c.

5.4. Set Packing

Given a family of sets $\mathcal{S}$, a *set packing* is a subfamily $\mathcal{S}' \subseteq \mathcal{S}$, where the elements in $\mathcal{S}'$ are pairwise disjoint from each other, i.e., for any two sets $S_i, S_j \in \mathcal{S}'$, $S_i \cap S_j = \emptyset$. The problem of finding such a set is known as set packing [Karp, 1972].

5.4.1. Maximum Set Packing (MaxSP)

The optimization version of this problem is Maximum Set Packing (MaxSP) for a family of sets $\mathcal{S}$, where $\mathcal{S}' \subseteq \mathcal{S}$, as follows.

Maximize: $|\mathcal{S}'|$
Subject to: $\forall S_i, S_j \in \mathcal{S}', S_i \cap S_j = \emptyset$

The problem can be formulated as a QUBO problem by defining binary variables x_i for each set $S_i \in \mathcal{S}$, where $x_i = 1$ if S_i is included in the packing and 0 otherwise [Lucas, 2014].

The objective of maximizing the number of sets leads to

$$\hat{H}_S^{SP}(\vec{x}) = B \sum_i x_i.$$

The constraint that there does not exist a pair of sets $S_i, S_j \in \mathcal{S}'$ such that $S_i \cap S_j \neq \emptyset$ is given by

$$\hat{H}_D^{SP}(\vec{x}) = -A \sum_{i,j:S_i \cap S_j \neq \emptyset} x_i x_j,$$

where the sum is taken over all pairs of sets S_i, S_j that intersect. If both x_i and x_j are assigned to be 1, then the disjoint constraint is violated (and therefore adds a penalty of $-A$ to the cost Hamiltonian). A trivial packing arises when any one single set in $\mathcal{S}$ is set to 1. The total cost for the MaxSP problem is

$$\hat{H}_{\text{SP}}(\vec{x}) = \hat{H}_S^{SP}(\vec{x}) + \hat{H}_D^{SP}(\vec{x}). \tag{5.10}$$

Similar to the MaxIS and MaxCl problems, this is formulated as a maximization problem. To convert it into a minimization problem, one can change the sign of the objective function, which allows us to maximize the original function indirectly.

5.4.2. Maximum Profitable Set Packing (MaxPSP)

Using the SCOOP framework defined in Chapter 4, we derive the unconstrained problem Maximum Profitable Set Packing or MaxPSP and prove that they are SCOOP twins using Theorem 4.

Step 1: Identify constraints
To maintain feasibility, MaxSP requires that the selected subset $\mathcal{S}' \subseteq \mathcal{S}$ contains no intersecting sets, thereby forming a packing.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of sets *violates packing* and its *closeness* to being a packing (i.e., the fewer intersections between sets, the better).

Step 3: Develop objective function $\mathtt{obj}_{PSP}^{G}$
For a subset $\mathcal{PSP} \subseteq \mathcal{S}$ in $\mathcal{S}$,

$\texttt{obj}^G_{PSP}(\mathcal{PSP}) = |\mathcal{PSP}| - |D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP})|$, where $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP})$ is the set of pairs of sets in $\mathcal{PSP}$ that intersect. We denote the value of the objective function $\texttt{obj}^G_{PSP}(\mathcal{PSP})$ as the *profit* $\mathfrak{p}_{\mathcal{PSP}}$.

Step 4: Objective function compatibility
From Theorem 4 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = k$ and $f_G^{-1}(k) = \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 4 describes a polynomial-time post-processing procedure that converts a solution $\mathcal{PSP}$ for MAXPSP to a packing, i.e., a solution for MAXSP while preserving solution quality.

Algorithm 4: Converting Profit Set Packing to Set Packing

Require: Family of sets $\mathcal{S}$, Solution $\mathcal{PSP}$
  Let $D_{\mathcal{PSP}}$ be the set of pairs of sets in $\mathcal{PSP}$ that intersect
  for each pair of sets $S_i, S_j \in D_{\mathcal{PSP}}$ **do**
    if $S_i \cap S_j \neq \emptyset$ **then**
      Remove S_i from $\mathcal{PSP}$
    end if
  end for

Cost Hamiltonian for MAXIMUM PROFITABLE SET PACKING

The binary variables for the MAXPSP problem are similar to MAXSP: each x_i is a binary variable with value 1 if S_i is included in the profitable packing $\mathcal{PSP}$, and 0 otherwise. The cost Hamiltonian for MAXPSP is given as follows:

Cost Hamiltonian for MAXPSP

Set cost:

$$\hat{H}_S^{PSP}(\vec{x}) = \sum_i x_i$$

Intersection cost:

$$\hat{H}_D^{PSP}(\vec{x}) = \sum_{i,j:S_i \cap S_j \neq \emptyset} (x_i x_j)$$

The total cost that is maximized for MAXPSP is

$$\hat{H}_{\text{PSP}}(\vec{x}) = \hat{H}_S^{PSP}(\vec{x}) - \hat{H}_D^{PSP}(\vec{x}) \tag{5.11}$$

Notably, the cost Hamiltonians for MAXIS and MAXSP are structurally similar, with the main difference being that the inputs for MAXSP are sets of elements rather than vertices [Lucas, 2014]. Similarly, the cost Hamiltonian for MAXPSP is also similar to the cost Hamiltonian for MAXPI, also with the main difference being that the inputs for MAXPSP are sets rather than pairs of vertices.

5.4.2.1. Finding maximum set packings via maximum profitable set packings

Similar to the connection between the decision versions of MAXVC and MAXPC, maximum set packings can also be identified through profitable set packings. The following theorem shows the relationship between profitable packing and set packing which can then be subsequently used to convert profitable packings to packings.

Theorem 4. *For any family of sets $\mathcal{S}$, $\mathcal{S}$ has a packing of size k if a only if $\mathcal{S}$ has a subset $\mathcal{PSP} \subseteq \mathcal{S}$ with profit $\mathfrak{p}_{\mathcal{PSP}} = k$.*

Proof. The proof is based on the relationship between packing and disjoint sets. The proof is divided into two parts:

($\Leftarrow$) First, we show that if $\mathcal{S}$ has a packing of size k, then there exists a subset $\mathcal{PSP} \subseteq \mathcal{S}$ with profit $\mathfrak{p}_{\mathcal{PSP}} = k$. The profit of $\mathcal{PSP}$ is given by $\mathfrak{p}_{\mathcal{PSP}} = |\mathcal{PSP}| - |D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP})|$. Since $\mathcal{P}$ is a packing, $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP}) = \emptyset$, and therefore $\mathfrak{p}_{\mathcal{PSP}} = |\mathcal{P}| = k$.

($\Rightarrow$) Consider a subset $\mathcal{PSP} \subseteq \mathcal{S}$ with profit $\mathfrak{p}_{\mathcal{PSP}} = k$. We need to show that there exists a packing of size k. If $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP}) = \emptyset$, then $\mathcal{PSP}$ is a packing of size k. If $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP}) \neq \emptyset$, then not all sets in $\mathcal{PSP}$ are disjoint. In this case, we can obtain a packing of size at least k by removing sets from $\mathcal{PSP}$ as follows.

(*) Pick a pair of sets $S_i, S_j \in D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP})$ that intersect. Remove one of the sets, say S_i, from $\mathcal{PSP}$. Note that the profit of the updated set $\mathcal{PSP}$ remains $\mathfrak{p}_{\mathcal{PSP}}$ since removal of one set S_i from $\mathcal{PSP}$ reduces $|\mathcal{PSP}|$, but also removes one pair S_i, S_j from $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP})$, thus keeping the profit unchanged. We repeat (*) as long as $D_{\mathcal{PSP}}(\mathcal{S}, \mathcal{PSP}) \neq \emptyset$. □

Theorem 4 implies that, given a maximum profitable set packing $\mathcal{PSP}$ for a family of sets $\mathcal{S}$, we can obtain a maximum packing for $\mathcal{S}$. To convert profitable set packing results to packings, we apply Alg. 4 based on Theorem 4.

5.5. 3SAT: An odd one out

The problems discussed so far, MAXVC, MAXIS, MAXCL, and MAXPSP, all share a common theme: they involve identifying subsets of vertices or sets that satisfy certain constraints, such as independence or disjointness. These constrained problems can be formulated using the QUBO framework by using penalties or by using their unconstrained SCOOP twins, where the goal is to maximize a profit function derived from the constraints.

While satisfiability problems (SAT) do not fit neatly into the SCOOP framework's approach, we demonstrate how SCOOP can be adapted to solve these problems.

SAT is a decision problem that asks whether there exists an assignment of truth values to variables in a logical formula such that the formula evaluates to true.

Let $X = \{x_1, \ldots, x_n\}$ be a set of boolean variables. A *literal* is either a variable x_i or its negation $\neg x_i$. A clause is a disjunction of *literals* (e.g., $x_1 \vee \neg x_2 \vee x_3$). A formula is in Conjunctive Normal Form (CNF) if it is a conjunction (AND, $\wedge$) of clauses, where each clause is a disjunction (OR, $\vee$) of literals. A 3-CNF formula ϕ is a conjunction of clauses where each clause contains exactly three literals.

$$\bigwedge_{ijk} \Big(y_i \vee y_j \vee y_k \Big) \quad \text{where } y_i \in \{x_i, \neg x_i\} \tag{5.12}$$

For example,

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee x_4)$$

is a 3-CNF formula with three clauses and five variables.

Below is a brief overview of the variants of the satisfiability problem relevant to this chapter:

- **Boolean Satisfiability Problem (SAT)**: Given a Boolean formula, determine if there exists an assignment of truth values to its variables that makes the formula true. The problem is NP-complete [Cook, 1971].
- **3-SAT**: A specific case of SAT where the formula is in Conjunctive Normal Form (CNF) with each clause containing exactly three literals. The problem is NP-complete [Karp, 1972].
- **Maximum 3-Satisfiability (MAX3SAT)**: An optimization version of 3-SAT that seeks to maximize the number of satisfied clauses in a 3-CNF formula. The problem is NP-hard [Papadimitriou and Yannakakis, 1988].
- **Decision Version of MAX3SAT**: The decision variant of MAX3SAT asks whether there exists an assignment satisfying at least k clauses, where k is a given threshold [Garey and Johnson, 1979].

5.5.1. MAXIMUM 3-SATISFIABILITY (MAX3SAT)

Let ϕ be a 3-CNF formula with a set of n clauses $C = \{c_1, c_2 \dots c_m\}$ over n variables $X = \{x_1, x_2 \dots x_n\}$.

Each clause is of the form $c_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$, where each $\ell_{ik} = x_i$ or $\ell_{ik} = \neg x_i$, where $x_i \in X$.

A truth assignment $\tau(\ell_{i1}, \ell_{i2}, \ell_{i3})$ evaluates to true if $(\ell_{i1} \vee \ell_{i2} \vee \ell_{i3}) = 1$, and false otherwise.

The problem MAXIMUM 3-SATISFIABILITY (MAX3SAT) is defined as follows:

$$\text{Maximize:} \quad \sum_{c \in C} \mathtt{sat}(c, \tau)$$

where $\mathtt{sat}(c, \tau)$ is 1 if clause c is satisfied by truth assignment τ, and 0 otherwise.

A formula is *satisfiable* if there exists an assignment that satisfies all clauses. The objective of MAX3SAT is to find an assignment that satisfies the maximum number of clauses.

Cost Hamiltonian for MAX3SAT

For the problem MAX3SAT, we define binary variables for each literal in the formula. Let $\ell_{jk}(x_k)$ represent the encoding of the k-th variable as it appears in clause j:

$$\ell_{jk}(x_k) = \begin{cases} x_k & \text{if literal } x_k \text{ appears in clause } j \\ 1 - x_k & \text{if literal } \neg x_k \text{ appears in clause } j \end{cases} \tag{5.13}$$

Each clause's disjunction (OR) of literals can be transformed into a product of binary variables:

$$(x_1 \vee \neg x_2 \vee x_3) \to x_1(1 - x_2)x_3$$

Since this formulation involves cubic terms (products of three binary variables), it results in a higher-order (HUBO) rather than quadratic (QUBO) encoding. In the HUBO formulation of the MAX3SAT problem, there are no constraints and it is an *unconstrained* optimization problem. The goal is to maximize the number of satisfied clauses by finding an optimal assignment of the binary variables.

The cost Hamiltonian for MAX3SAT in its higher-order form is given by:

Cost Hamiltonian for MAX3SAT

The total cost that is maximized for MAX3SAT is

$$\hat{H}_{\text{Max3SAT}}(\vec{x}) = \sum_{j \in \phi} \left(1 - \prod_{k \in j} (1 - \ell_{jk}(x_k))\right) \tag{5.14}$$

Note that any clause containing a literal and its negation (e.g., $x_i \vee \neg x_i \vee x_j$) is always satisfied. Therefore, such clauses can be removed from the formula without affecting the optimal solution, allowing us to work with a simplified problem instance. The cost Hamiltonian $\hat{H}_{\text{Max3SAT}}(\vec{x})$ is a sum of products of binary variables corresponding to the literals in each clause. An assignment of the binary variables that maximizes the total cost corresponds to satisfying as many clauses as possible.

MAX3SAT presents an interesting case that does not directly apply the SCOOP framework. Unlike the other problems we have examined, MAX3SAT is naturally unconstrained. However, its direct formulation leads to cubic terms (HUBO) rather than quadratic terms. Various quadratization techniques have been proposed to transform the MAX3SAT HUBO into a QUBO. Chancellor et al. [Chancellor et al., 2016] discuss a method to convert MAX3SAT HUBO into a QUBO by introducing m auxiliary variables, where m is the number of clauses, and therefore, the number of cubic terms. This results in a QUBO with $n + m$ variables. Converting this into a minimization problem, they assign a cost of g to every clause with an assignment that does not satisfy it. Choi [Choi, 2010] proposes a penalty-based problem Hamiltonian with $3m$ qubits based on the reduction of MAX3SAT problem to the MAXIS problem [Garey and Johnson, 1979, Dietzfelbinger, 2008]. Zielinski et al. [Zielinski et al., 2024] provide approximate QUBO representations of MAX3SAT for quantum annealers that use $n + m$ qubits but have fewer couplings than the method by Chancellor et al. [Chancellor et al., 2016].

These quadratization approaches, while converting the HUBO to a QUBO formulation, rely on penalty terms and auxiliary variables that introduce additional complexity to quantum implementations. The penalty parameters require careful tuning, and the auxiliary variables increase qubit requirements, both of which present significant challenges for near-term quantum devices [Gabor et al., 2019].

MAX3SAT's relationship with MAXIS suggests another path: we could solve MAX3SAT through MAXIS's SCOOP twin, MAXPI. Although this approach still requires additional qubits, it eliminates the need for penalty parameter tuning.

5.5.1.1. Finding solution to MAX3SAT via maximum profit independent sets

To bridge between MAX3SAT and graph-based problems like MAXIS and MAXPI, we must first transform the clause-based structure of MAX3SAT into a graph representation. The creation of graph G_ϕ is based on the classic polynomial time reduction from 3SAT to Independent set [Garey and Johnson, 1979], is as follows.

Given 3-CNF formula $\phi = (\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge (\ell_{21} \vee \ell_{22} \vee \ell_{23}) \wedge \cdots \wedge (\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})$ with m clauses over n variables $\{x_1, x_2, \ldots x_n\}$ and each $\ell_{jk} = x_i$ or $\ell_{jk} = \neg x_i$ for some $x_i \in \{x_1, x_2, \ldots x_n\}$, then $G_\phi = (V, E)$ with $V = \{\ell_{11}, \ldots \ell_{m3}\}$, i.e., for each clause we create three vertices representing its literals. The edge set E of G_ϕ are as follows:

- For each j create an edges (ℓ_{j1}, ℓ_{j2}), (ℓ_{j2}, ℓ_{j3}), (ℓ_{j1}, ℓ_{j3}). We refer to the vertices corresponding to a clause also as *triangle*.
- For each j, j' where $j \not= j'$, and each $\ell_{jk}, \ell_{j'k'}$ with $\ell_{jk} = x_i$, create edge $(\ell_{jk}, \ell_{j'k'})$ if $\ell_{j'k'} = \neg x_i$.

We note that graph G_ϕ has $3m$ vertices. Furthermore, 3SAT problem instance ϕ with m clauses has a satisfying assignment if there is an independent set of size m for G_ϕ [Garey and Johnson, 1979, Dietzfelbinger, 2008].

The following theorem establishes that, given a profit independent set PI for G_ϕ with $|\text{PI}| = k$, we can determine a truth assignment for ϕ that satisfies k clauses.

Theorem 5. *A 3-CNF formula ϕ with m clauses over n variables $\{x_1, x_2, \ldots x_n\}$, has a truth assignment with k satisfiable clauses if and only if the gadget graph $G_\phi = (V, E)$ created from ϕ has a subset PI $\subseteq V$ with profit $\mathfrak{p}_{PI} = k$.*

Proof.

($\Leftarrow$) Recall that, given a profit independent set PI $\subseteq V$ with profit $\mathfrak{p}_{\text{PI}} = k$, we can construct an independent set *IS* $\subseteq V$ of size $|\text{IS}| \geq k$ (Theorem 2) using the post-processing algorithm outlined in Alg. 2.

Since *IS* is an independent set, for G_ϕ there do not exists two vertices in *IS*, representing a literal x_i and its negation $\neg x_i$, respectively. Furthermore, since the vertices representing literals in a clause are pairwise connected, for each triangle subgraph of G_ϕ, which corresponds to a clause, at most one vertex is a member of IS.

At least k clauses of G_ϕ are satisfiable by assigning each variable in $x_i \in \{x_1, \ldots x_n\}$ a truth value as follows. If there exists a literal $\ell_{jk} = x_i$ and $\ell_{jk} \in$ IS then assign x_i to *true*. If instead, literal $\ell_{jk} = \neg x_i$ and $\ell_{jk} \in$ IS then assign x_i to *false*.

($\Rightarrow$) This direction follows directly from the polynomial time reduction from 3SAT to Independent Set [Garey and Johnson, 1979] in combination with Theorem 2.

□

From Theorem 5 we observe the following: The gadget graph construction ensures that each selected vertex must be an independent one in its clause-triangle, yielding profit m if and only if all clauses are satisfiable.

Corollary 1. *A 3-CNF formula ϕ with m clauses and n literals, is satisfiable if and only if the gadget graph $G_\phi = (V, E)$ created from ϕ has a subset PI $\subseteq V$ with profit $\mathfrak{p}_{PI} = m$.*

In summary, Theorem 5 implies that, given a maximum profit independence PI with a profit of $\mathfrak{p}_{\text{PI}} = k$ for a graph G, we can obtain a truth assignment for the 3-CNF formula ϕ that satisfies k clauses. Corollary 1 implies that, given a maximum profit independence PI of profit $\mathfrak{p}_{\text{PI}} = m$ for a graph G, we can obtain a a satisfiable assignment for the 3-CNF formula ϕ. To convert all profit independence results to independent sets, we apply Alg. 2 based on Theorem 2 defined in Section 5.1.

5.5.2. SCOOP inspired process for MAX3SAT

Due to the nature of the MAX3SAT problem, the conditions for the SCOOP framework are not directly applicable as they were for the other problems discussed in this chapter. MAX3SAT is inherently an unconstrained optimization problem, as it seeks to maximize the number of satisfied clauses without any additional constraints on the assignments. Since the direct encoding of MAX3SAT as an unconstrained binary formulation requires higher-order terms, the quadratization of this problem into a QUBO using MAXIS introduces a constrained variant. As shown in the previous section, combined with pre-processing (construction of G_ϕ) and post-processing (transforming a profit independent set for G_ϕ into a truth assignment of ϕ) enables us to solve MAX3SAT with QAOA with a penalty-free QUBO. Nevertheless, MAX3SAT and MAXPI do not satisfy the properties of being SCOOP twins: MAX3SAT and MAXPI do not share the same input domain, as MAX3SAT operates on clauses and literals, while MAXPI operates on vertices in a graph. Since they do not share the same input domain, we cannot directly compare solutions between the two problems for solution containment, objective function compatibility, or solution enhance-ability as required by the SCOOP framework.

Algorithm 5: Converting Profit Independent Set to MAX3SAT truth assignment

Require: Formula ϕ, Graph $G_\phi = (V, E)$, Solution PI
  Let $E_{\text{conf}} \subseteq E$ be the set of edges with both endpoints in PI
  for each edge $uv \in E_{\text{conf}}$ **do**
    if $u \in$ PI and $v \in$ PI **then**
      Remove u from PI
    end if
  end for
  for each clause $c_j \in C$ **do**
    Let $V_{j\triangle} = \{\ell_{j1}, \ell_{j2}, \ell_{j3}\} \subseteq V$
    for each literal $\ell_{jk} \in V_{j\triangle}$ **do**
      if $\exists \ell_{jk} = x_i$ and $\ell_{jk} \in$ IS **then**
        $x_i =$ TRUE
        break
      end if
      if $\exists \ell_{jk} = \neg x_i$ and $\ell_{jk} \in$ IS **then**
        $x_i =$ FALSE
        break
      end if
    end for
  end for

5.6. Chapter Summary

This chapter demonstrated the broad applicability of our SCOOP framework to problems that can be naturally encoded as QUBOs. We examined several NP-hard problems including MAXIMUM INDEPENDENT SET, MAXIMUM CLIQUE, MAXIMUM SET PACKING, and MAX3SAT, and derived their SCOOP twins that also have QUBO formulations For Max3SAT, we demonstrated an interesting case where the problem, while unconstrained in its original form (a HUBO), requires careful consideration when transformed into a QUBO representation. The shared structure with Maximum Independent Set enabled us to apply similar penalty-free techniques, showcasing the framework's flexibility.

In the next chapter we extend our framework to selected NP-hard and polynomial-time solvable problems that require higher-order formulations (HUBOs).

Chapter 6

Higher-Order Problem Formulations

Contents

Chapter Contributions

C6: Derivations of the unconstrained SCOOP twins of NP-hard and classically tractable problems that have a higher-order unconstrained binary optimization (HUBO) formulation, including MINIMUM EDGE DOMINATING SET, MINIMUM MAXIMAL MATCHING, MINIMUM SET COVER,MINIMUM EDGE COVER, and MAXIMUM MATCHING

C7: HUBO formulations of NP-hard problems including MINIMUM DOMINATING SET, MINIMUM MAXIMAL MATCHING, MINIMUM EDGE DOMINATING SET, MINIMUM SET COVER and their respective SCOOP twins

C8: HUBO formulations of classically tractable problems including MINIMUM EDGE COVER and MAXIMUM MATCHING, and their SCOOP twins, demonstrating SCOOP's versatility across complexity classes

This chapter presents higher-order unconstrained binary optimization (HUBO) formulations for several combinatorial optimization problems that cannot be described as QUBOs without quadratization. While any HUBO can be converted to a QUBO through quadratization techniques, the problems in this chapter have natural higher-order formulations that directly capture their underlying structure. For each problem, we present both the constrained formulation and its unconstrained HUBO equivalent, along with their relationships and corresponding cost operators.

The problems we examine include MINIMUM DOMINATING SET, MINIMUM MAXIMAL MATCHING, and MINIMUM SET COVER, all of which are NP-hard. We also explore polynomial-time solvable problems like MINIMUM EDGE COVER, demonstrating SCOOP's versatility across computational complexity classes. For each problem, we show how their natural higher-order formulations relate to their constrained counterparts and examine the implications for quantum optimization.

6.1. Dominating Set

A *dominating set* in an undirected graph $G = (V, E)$[1] with vertex set V and edge set E is a subset $DS \in V$ such that every vertex in G is "dominated" by DS. A vertex is considered to dominate itself and all of its adjacent vertices. This ensures that all vertices in the graph are either part of the dominating set or directly connected to a vertex in it. A smallest possible dominating set is referred to as *minimum dominating set*. An application area is to model wireless sensor network design, where a minimal number of sensor nodes must cover the entire sensing area for cost-effective monitoring. The dominating set problem also appears

[1] In this chapter, we refer to simple undirected graphs without isolated vertices.

in social network influence, where the goal is to identify a minimal group of individuals who can directly or indirectly influence the entire network.

6.1.1. MINIMUM DOMINATING SET (MINDS)

We define the problem MINIMUM DOMINATING SET (MINDS) for a graph $G = (V, E)$ as follows. A subset of vertices DS $\subseteq V$ is a *dominating set* if every vertex $v \in V$ is either in DS or adjacent to at least one vertex in DS. The objective is to find the smallest such set:

Minimize: |DS|
Subject to: for all $v \in V$
$v \in DS$ or there exists $u \in N(v)$ such that $u \in$ DS

Cost Hamiltonians for MINDS

We describe cost Hamiltonians for MINDS ($\hat{H}_{\text{DS}}$) using higher-order unconstrained binary formulations. Let $\vec{x} = (x_1, x_2, \ldots, x_{|V|})$ be a vector of binary variables, where $x_i \in \{0, 1\}$ for each $i \in V$, then,

$$\begin{aligned}\hat{H}_{\text{DS}}(\vec{x}) &= \hat{H}_N^{DS}(\vec{x}) + \hat{H}_V^{DS}(\vec{x}) \\ &= A \sum_{i \in V} \left[\left((1 - x_i) \prod_{j \in N(i)} (1 - x_j) \right) \right] + B \sum_{i \in V} x_i,\end{aligned}$$

with, $A > B$.

We formulated the HUBO of the constrained COP MINDS following the guidelines of Glover et al. [Glover et al., 2022]. The goal of this HUBO is to minimize $\hat{H}_{\text{DS}}(\vec{x})$. $\hat{H}_{\text{DS}}(\vec{x})$ imposes a penalty for vertices that are not dominated by a factor of A. Its minimum value is $\hat{H}_{\text{DS}}(\vec{x}) = B \sum_{i \in V} x_i$.

Dinneen et al. [Dinneen and Hua, 2017a] provide a different, quadratic, formulation for MINDS, which requires additional variables/qubits to balance penalties where more than one vertex is selected in the dominating set. We present the QUBO below, in Eqn. 6.1for reference.

Let $x_i \in \{0, 1\}$ be binary variables indicating whether vertex $v_i \in V$ is in the dominating set. For each vertex v_i, we also define auxiliary binary variables $y_{i,k} \in \{0, 1\}$ for $0 \leq k \leq \lfloor \log_2(\deg(v_i)) \rfloor$ to encode a balancing integer.

The QUBO objective function is:

$$F(x,y) = \sum_{v_i \in V} x_i + A \sum_{v_i \in V} P_i, \tag{6.1}$$

where

$$P_i = \left(1 - \left(x_i + \sum_{v_j \in N(v_i)} x_j\right) + \sum_{k=0}^{\lfloor \log_2(\deg(v_i)) \rfloor} 2^k y_{i,k}\right)^2, \tag{6.2}$$

and $A > 1$ is a penalty parameter to enforce constraint satisfaction.

In this formulation, $\sum x_i$ minimizes the number of selected vertices (i.e., the size of the dominating set). P_i penalizes any vertex v_i that is not dominated (i.e., neither $x_i = 1$ nor one of its neighbors is in the set). The auxiliary variables $y_{i,k}$ compensate for potential over-penalization when more than one vertex in $\{v_i\} \cup N(v_i)$ is selected.

The optimal solution is obtained by minimizing $F(x,y)$, and the dominating set is recovered as:

$$D(x^*) = \{v_i \in V \mid x_i = 1\}.$$

The number of binary variables used is $O(n + n \log n)$ in the worst case. We go around the necessity of auxiliary variables by using the higher-order formulation.

6.1.2. Maximum Profit Domination (MaxPD)

Using our SCOOP framework for MinDS, we derive the unconstrained problem Maximum Profit Domination or MaxPD. Theorem 6 below concludes the proof that MinDS and MaxPD are SCOOP twins.

Step 1: Identify constraints
The constraint that MinDS must satisfy to ensure feasibility is that the subset DS $\subseteq V$ is a dominating set.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of vertices *dominates vertices* and its *closeness* to being a dominating set (i.e., the more vertices are dominated, the better).

Step 3: Develop objective function $\mathtt{obj}^{\mathsf{G}}_{\mathsf{PD}}$
Let $V_{\text{PD}}(G, \text{PD})$ denote the set of vertices dominated by a subset PD $\subseteq V$ in graph G,

$$V_{\text{PD}}(G, \text{PD}) = \{v \in V \mid v \text{ is dominated by PD}\}$$

For a subset PD $\subseteq V$ in G,
$\mathtt{obj}^G_{\text{PD}}(\text{PD}) = |V_{\text{PD}}(G, \text{PD})| - |\text{PD}|$. We also refer to the value produced by the objective function $\mathtt{obj}^G_{\text{PD}}(\text{PD})$ as *profit*, $\mathfrak{p}_{\text{PD}}$.

Before moving on to Step 4, we formulate the problem Maximum Profit Domination (MaxPD) [Fernau and Stege, 2019] for a graph $G = (V, E)$ and PD $\subseteq V$:

$$\text{Maximize: profit } \mathfrak{p}_{\text{PD}}, \text{where } \mathfrak{p}_{\text{PD}} = \texttt{obj}_{\text{PD}}^{G}(\text{PD})$$

Step 4: Objective function compatibility
From Theorem 6 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = |V| - k$ and $f_G^{-1}(k) = |V| - \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 6 describes a polynomial-time post-processing procedure that converts a solution PD for MAXPD to a domintating set, i.e., a solution for MINDS while preserving solution quality.

Algorithm 6: Converting solutions: MAXPD to MINDS adapted from [Fernau and Stege, 2019]

Require: Graph G, Solution PD
  Let V_{nd} be the set of non-dominated vertices in G
  for each vertex $v \in V_{\text{nd}}$ **do**
    if $N(v) \not\subseteq$ PD and $v \notin$ PD **then**
      Add v to PD
    end if
  end for

Cost Hamiltonian for MAXIMUM PROFIT DOMINATION (MAXPD)

For the unconstrained COP MAXPD, the goal is to maximize the unconstrained objective $\hat{H}_{\text{PD}}(\vec{x})$. The term $\hat{H}_N^{PD}(\vec{x})$ keeps track of all vertices that are dominated—the contribution of each vertex is 1 if dominated and 0 otherwise.

Cost Hamiltonian for MAXPD

Neighborhood cost:

$$\hat{H}_N^{PD}(\vec{x}) = \sum_{i \in V} \left[1 - \left((1 - x_i) \prod_{j \in N(i)} (1 - x_j) \right) \right]$$

Vertex cost:

$$\hat{H}_V^{PD}(\vec{x}) = \sum_{i \in V} x_i$$

The total cost that is maximized for MAXPD is

$$\hat{H}_{\text{PD}}(\vec{x}) = \hat{H}_N^{PD}(\vec{x}) - \hat{H}_V^{PD}(\vec{x}) \tag{6.3}$$

6.1.2.1. Finding minimum dominating sets via maximum profit domination

We now establish the relationship between MINDS and MAXPD. Theorem 6 demonstrates this equivalence, showing how maximum profit domination solutions can be transformed into minimum dominating sets while preserving solution quality characteristics.

Theorem 6. *[Fernau and Stege, 2019] For any graph $G = (V, E)$, G has a dominating set $DS \subseteq V$ of size k if and only if G has a subset $PD \subseteq V$ with profit $\mathfrak{p}_{PD} = |V| - k$.*

Proof.

($\Rightarrow$) Determining the profit $\mathfrak{p}_{\text{PD}}$ of a dominating set DS $\subseteq V$ for G results in $\mathfrak{p}_{\text{PD}} = \texttt{obj}^{G}_{\text{PD}}(DS)$. Since DS is a dominating set $\mathfrak{p}_{\text{PD}} = |E| - k$.

($\Leftarrow$) Consider a subset $PD \subseteq V$ with profit $\mathfrak{p}_{\text{PD}} = |V| - k$. If PD is a dominating set, then |PD| $= k$ and thus PD is a dominating set of size k.

If instead PD is not a dominating set then we can obtain a dominating set of size at most $|V| - \mathfrak{p}_{\text{PD}}$ by applying Alg. 6. Note that the algorithm at no step when expanding PD reduces the profit. □

Theorem 6 implies that, given an optimal solution to MAXPD PD for a graph G, we can obtain a minimum dominating set for G using Algorithm 6.

6.2. Maximal Matching

A *matching* in a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that no two edges in M share a vertex.

A *maximal matching* M is a matching that cannot be extended by adding any other edge from E without violating the matching property. Formally, M is maximal if, for every edge $(u, v) \in E \setminus M$, $M \cup (u, v)$ is not a matching in G.

6.2.1. Minimum Maximal Matching (M^3)

A *minimum maximal matching* is a maximal matching with the least possible number of edges. That is, if $\mathcal{M}$ is the set of all maximal matchings in G, a minimum maximal matching M^* satisfies

Minimize: $|M^*|$
Subject to: (1) M^* is a matching: for every pair of edges $ab, cd \in M^*, ab \neq cd$, a, b, c and d are pairwise distinct
(2) matching M^* is a maximal: there is no edge $e \in E$ with $M^* \cup \{e\}$ is a matching

Here, $\texttt{obj}^G_{M^3}(M) = |M|$. Solutions to the problem are subsets of edges (i.e., $\text{Sol}(G) = |E|$) that must satisfy constraints (1) and (2), describing $\text{Sol}_C(G) \subset \text{Sol}(G)$.

Cost Hamiltonians for M^3

In 2014, Lucas [Lucas, 2014] provided the QUBO formulation for M^3. Let $\vec{x} = (x_1, x_2, \ldots, x_{|E|})$ be a vector of binary variables, where $x_e \in \{0, 1\}$ for each $e \in E$ representing whether or not edge e is in the matching. The Hamiltonian $\hat{H}_{M^3}(\vec{x}, \vec{y})$ is defined as follows:

$$\hat{H}_{M^3}(\vec{x}, \vec{y}) = \hat{H}_A^{M^3}(\vec{x}) + \hat{H}_B^{M^3}(\vec{y}) + \hat{H}_C^{M^3}(\vec{x}),$$

where:

$$\hat{H}_A^{M^3}(\vec{x}) = A \sum_{v \in V} \sum_{\{e_1, e_2\} \subset \partial v} x_{e_1} x_{e_2}$$

penalizes having two matched edges incident to the same vertex v, where ∂v is the set of edges incident to v. This term enforces the matching constraint.

Auxiliary binary variables, $\vec{y}$, are defined such that $y_v = \sum_{e \in \partial v} x_e$ (only valid for states with $H_A = 0$) which indicates if a vertex v has a matched edge. Then,

$$\hat{H}_B^{M^3}(\vec{y}) = B \sum_{e=(u,v) \in E} (1 - y_u)(1 - y_v)$$

penalizes states where an edge (u, v) could be added to the matching (i.e., both $y_u = 0$ and $y_v = 0$) without violating the matching constraint, thus enforcing maximality. Finally, since

$$\hat{H}_C^{M^3}(\vec{x}) = C \sum_{e \in E} x_e$$

counts the number of matched edges, the optimal value of the total Hamiltonian corresponds to a minimum maximal matching. Lucas highlights the absence of prior work on M^3; our current analysis confirms that it remains unaddressed in the literature from the perspective of QAOA or QA.

6.2.2. Maximum Profitable Edge Set (MaxPES)

We now define the unconstrained problem Maximum Profitable Edge Set (MaxPES) as a SCOOP twin of M^3. To quantify how much a subset E' of edges satisfies the constraints, when E' is not feasible, that is E' does not satisfy one or both of the constraints, we say that an edge $e \in E$ is *covered* by E' if there is an edge $e' \in E'$ such that e and e' share a common endpoint. In other words every, edges covered by E' are all edges in E' plus all edges in $E \setminus E'$ that are "adjacent" to E'.

The profit formulation MaxPES of M^3 is derived by relaxing the two constraints of M^3: we neither require the set of edges to be a matching nor maximal. Using the SCOOP framework, this process can by summarized as follows.

Step 1: Identify constraints
The constraints that any solution $M \subseteq E$ to M^3 must satisfy to ensure feasibility is that M is a matching that is maximal.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset *covers edges*. An edge is covered by itself or is adjacent to an edge in the subset.

Step 3: Develop objective function $\mathtt{obj}_{\mathrm{PES}}$
For a subset PES $\subseteq E$ in G,
Let $E_{\mathrm{PES}}(G, \mathrm{PES}) = \{e \in E \mid e \text{ is covered by PES}\}$ be the set of edges covered by PES. $\mathtt{obj}_{\mathrm{PES}}^{G}(\mathrm{PES}) = |\{e \in E \mid e \text{ is covered by PES}\}| - |\mathrm{PES}|$. We refer to the value produced by the objective function $\mathtt{obj}_{\mathrm{PES}}^{G}(\mathrm{PES})$ as *profit*, $\mathfrak{p}_{\mathrm{PES}}$.

Step 4: Objective function compatibility
From Theorem 7 we can derive function $f_G(.)$. Here, $f_G(\mathfrak{p}) = |E| - k$ and $f_G^{-1}(k) = |E| - \mathfrak{p}$.

Step 5: Solution enhance-ability
The proof of Theorem 7 shows a two-step process that converts any solution PES $\subseteq E$ for a given MAXPES-instance $G = (V, E)$ to a maximal matching, and therefore to a solution for M^3 while preserving solution quality. Alg. 7 describes the first step of the polynomial-time post-processing procedure, i.e. converting PES to a *maximal* subset of edges—a subset PES′ where the addition of any additional edge from E would reduce the profit–while preserving solution quality. Alg. 8 describes the second step that converts PES′ to a *maximal* matching, again while preserving solution quality.

We say that a subset that covers all edges is a *maximal profitable edge set* and a subset PES $\subseteq E$ that maximizes $\texttt{obj}^G_{\text{PES}}(\text{PES})$ is a *maximum profitable edge set*.

Algorithm 7: Converting solutions: profitable edge sets to maximal profitable edge sets

Require: Graph G, solution PES
  Let PES′ = PES
  Let E_{unc} be the set of uncovered edges in G
  for each edge $(u, v) \in E_{\text{unc}}$ **do**
    if $N_e(u, v) \not\subseteq$ PES and $(u, v) \notin$ PES **then**
      Add (u, v) to PES′
    end if
  end for

Algorithm 8: Converting a maximal profitable edge set into a maximal matching; adapted from [Yannakakis and Gavril, 1980]

Require: Graph G, maximal profitable edge set PES′
  for each pair of adjacent edges $(u, v), (v, w) \in$ PES′ **do**
    if PES′ $\setminus (u, v)$ covers all edges **then**
      Remove (u, v) from PES′ and break
    end if
    if PES′ $\setminus (v, w)$ covers all edges **then**
      Remove (v, w) from PES′ and break
    else
      Let S be the set of edges incident to w
      Pick an edge $(w, z) \in S$, $z \neq v$, such that (w, z) is covered only by (v, w)
      PES′ := (PES′ $\setminus (v, w)) \cup (w, z)$
    end if
  end for

Cost Hamiltonians for MAXPES

We describe the cost Hamiltonian for $\hat{H}_{\text{PES}}$ below.

Let $\vec{x} = (x_1, x_2, \ldots, x_{|E|})$ be a vector of binary variables, where $x_i \in \{0, 1\}$ for each $i \in E$.

Cost Hamiltonian for MAXPES

Edge Neighborhood cost:

$$\hat{H}_N^{PES}(\vec{x}) = \sum_{i \in E} \left[1 - \left((1 - x_i) \prod_{j \in N_e(i)} (1 - x_j) \right) \right]$$

Edge cost:

$$\hat{H}_V^{PES}(\vec{x}) = \sum_{i \in E} x_i$$

The total cost that is maximized for MAXPES is

$$\hat{H}_{PES}(\vec{x}) = \hat{H}_N^{PES}(\vec{x}) - \hat{H}_V^{PES}(\vec{x}) \tag{6.4}$$

For MAXPES, the goal is to maximize the unconstrained objective $\hat{H}_{PES}(\vec{x})$. The term $\hat{H}_N^{PES}(\vec{x})$ keeps track of all edges that are covered (the contribution of each edge that is covered is 1 and 0 otherwise).

Determining minimum maximal matchings via maximum profitable edge sets

The following theorem implies that we can derive, from any given subset of edges with profit $\mathfrak{p}$, a maximal matching *MM* with profit at least $\mathfrak{p}$.

Theorem 7. *Let $G = (V, E)$ be an undirected simple graph. G has a maximal matching $MM \subseteq E$ of size k if and only if G has a subset PES $\subseteq E$ with profit $\mathfrak{p}_{PES} = |E| - k$.*

Proof.

($\Rightarrow$) Consider a maximal matching $MM \subseteq E$ for G, $|MM| = k$. *MM* has a profit of $\mathfrak{p}_{\text{PES}} = |E_{\text{PES}}(G, MM)| - |MM|$, where $E_{PES}(G, MM)$ are the edges covered by *MM*. Since *MM* is a maximal matching, it covers all edges in G. Therefore, $|E_{\text{PES}}(G, MM)| = |E|$ and $\mathfrak{p}_{\text{PES}} = |E| - |k|$.

($\Leftarrow$) Let subset PES $\subseteq E$ with $\mathfrak{p}_{\text{PES}} = |E| - k$. For the case where PES is a maximal matching, there is nothing to prove.

If PES is not a maximal matching and $E_{\text{PES}}(G, \text{PES}) = E$, then $|\text{PES}| = k$, and PES covers all edges, which means that PES is a maximal profitable edge set.

If PES is neither a maximal matching nor a maximal profitable edge set, then PES does not cover all edges. One can obtain a maximal profitable edge set PES′ of size at most $|E| - \mathfrak{p}_{\text{PES}}$ and profit at least $\mathfrak{p}_{\text{PES}}$ by covering the remaining edges in $E \setminus E_{\text{PES}}(G, \text{PES})$ with post-processing Alg. 7.

Finally, if the maximal profitable edges set PES′ is not a matching, it can be converted in polynomial time into a maximal matching *MM* with $|MM| \leq |PES'|$ (Alg. 8).

□

Theorem 7 implies that, given a maximum profitable edge set PES for a graph G, we can obtain a minimum maximal matching for G using the procedure given in the proof. Generally, to convert any solution to instances for MAXPES to maximal matchings for the same instance without sacrificing the solution quality, we apply the 2-step procedure.

6.3. Edge Dominating Set

An *edge dominating set* in an undirected graph $G = (V, E)$ with vertex set V and edge set E is a subset EDS $\subseteq E$ such that every *edge* in G is "dominated" by EDS. An edge is considered to dominate itself and all of its adjacent vertices. This ensures that all vertices in the graph are either part of the edge dominating set or directly connected to an edge in it. A smallest possible edge dominating set is referred to as *minimum edge dominating set*.

6.3.1. Minimum Edge Dominating Set (MinEDS)

We define the problem Minimum Edge Dominating Set (MinEDS) for a graph $G = (V, E)$ as follows. A subset of vertices EDS $\subseteq E$ is an *edge dominating set* if every edge $e \in E$ is either in EDS or adjacent to at least one edge in EDS. The objective is to find the smallest such set:

$$\begin{aligned}&\text{Minimize:} \quad |\text{EDS}|\\ &\text{Subject to:} \quad \text{for all } e \in E\\ &\qquad e \in \textit{EDS} \;\; \text{or} \;\; \text{there exists } g \in N_e(e) \text{ such that } g \in \text{EDS}\end{aligned}$$

Here, $N_e(e)$ denotes the set of edges adjacent to edge e, where two edges are considered adjacent if they share a common vertex.

Cost Hamiltonians for MinEDS

We describe cost Hamiltonians for MinEDS ($\hat{H}_{\text{EDS}}$) using higher-order unconstrained binary formulations. Let $\vec{x} = (x_1, x_2, \ldots, x_{|E|})$ be a vector of binary variables, where $x_i \in \{0, 1\}$ for each $i \in E$, then,

$$\begin{aligned}\hat{H}_{\text{EDS}}(\vec{x}) &= \hat{H}_{N_e}^{EDS}(\vec{x}) + \hat{H}_{E}^{EDS}(\vec{x})\\ &= A\sum_{i\in E}\left[\Big((1-x_i)\prod_{j\in N_e(i)}(1-x_j)\Big)\right] + B\sum_{i\in E}x_i,\end{aligned}$$

with, $A > B$.

Similar to the MinDS problem, we formulated the HUBO of the MinEDS following the guidelines of Glover et al. [Glover et al., 2022]. The goal of this HUBO is to minimize $\hat{H}_{\text{EDS}}(\vec{x})$. $\hat{H}_{\text{EDS}}(\vec{x})$ imposes a penalty for edges that are not dominated by a factor of A. Its minimum value is $\hat{H}_{\text{EDS}}(\vec{x}) = B\sum_{i\in E} x_i$.

6.3.2. Using MAXPES as the SCOOP Twin for MINEDS

The MAXIMUM PROFITABLE EDGE SET (MAXPES) problem defined in Section 6.2.2 serves as a SCOOP twin for MINEDS as well.

We can derive the SCOOP twin MAXPES from the constrained MINEDS as follows:

Step 1: Identify constraints
The constraint that MINEDS must satisfy to ensure feasibility is that the subset EDS $\subseteq E$ is an edge dominating set.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of edges *dominates edges* and its *closeness* to being an edge dominating set (i.e., the more edges are dominated, the better).

Step 3: Develop objective function $\texttt{obj}^{G}_{\textbf{PES}}$
For a subset PES $\subseteq E$ in G,
Let $E_{\text{PES}}(G, \text{PES})$ denote the set of edges dominated by a subset PES $\subseteq E$ in graph G,

$$E_{\text{PES}}(G, \text{PES}) = \{e \in E \mid e \text{ is dominated by PES}\}$$

$\texttt{obj}^{G}_{\text{PES}}(\text{PES}) = |E_{\text{PES}}(G, \text{PES})| - |\text{PES}|$. We also refer to the value produced by the objective function $\texttt{obj}^{G}_{\text{PES}}(\text{PES})$ as *profit*, $\mathfrak{p}_{\text{PES}}$.

Step 4: Objective function compatibility
From Theorem 8 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = |E| - k$ and $f_G^{-1}(k) = |E| - \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 7 defined previously, describes the polynomial-time post-processing procedure that converts a solution PES for MAXPES to maximal profitable edge sets. A similar algorithm (Alg. 9) can be used to convert solutions of MAXPES to MINEDS

Algorithm 9: Converting solutions: MAXPES to MINEDS

Require: Graph G, Solution PES
  Let E_{nd} be the set of non-dominated edges in G
  for each edge $e \in E_{\text{nd}}$ **do**
    if $N_e(e) \not\subseteq$ PES and $e \notin$ PES **then**
      Add e to PES
    end if
  end for

The cost Hamiltonian for MAXPES is defined in Sec. 6.2.2

6.3.2.1. Finding minimum edge dominating sets via maximum profitable edge sets

We now establish the relationship between MINEDS and MAXPES. Theorem 8 demonstrates this equivalence, showing how maximum profit domination solutions can be transformed into minimum dominating sets while preserving solution quality characteristics.

Theorem 8. *For any graph $G = (V, E)$, G has an edge dominating set EDS $\subseteq E$ of size k if and only if G has a subset PES $\subseteq E$ with profit $\mathfrak{p}_{PES} = |E| - k$.*

Proof.

($\Rightarrow$) Determining the profit $\mathfrak{p}_{\text{PES}}$ of an edge dominating set EDS $\subseteq E$ for G results in $\mathfrak{p}_{\text{PES}} = \texttt{obj}^{G}_{\text{PES}}(\text{EDS})$. Since EDS is an edge dominating set $\mathfrak{p}_{\text{PES}} = |E| - k$.

($\Leftarrow$) Consider a subset $PES \subseteq V$ with profit $\mathfrak{p}_{\text{PES}} = |E| - k$. If PES is an edge dominating set, then $|\text{PES}| = k$ and thus PES is an edge dominating set of size k.

If instead PES is not a dominating set then we can obtain a dominating set of size at most $|E| - \mathfrak{p}_{\text{PES}}$ by applying Alg. 9. Note that the algorithm at no step when expanding PES reduces the profit. □

Theorem 8 implies that, given an optimal solution to MAXPES PES for a graph G, we can obtain a minimum edge dominating set for G using Algorithm 9.

6.3.3. Using MAXPES for MINIMUM INDEPENDENT EDGE DOMINATING SET

We would like to point out a connection between the problems investigated in the section with two other famous NP-hard problems. In 1980, Yannakakis and Garvil [Yannakakis and Gavril, 1980] pointed out the transformational equivalence between the problems M^3 and MINIMUM EDGE DOMINATING SET (MINEDS), the problem that for an undirected graph seeks a smallest subset of edges that covers every edge in the graph. When following the SCOOP framework one can see that the constrained problem M^3 and Minimum Edge Dominating Set share unconstrained SCOOP twin MAXPES. However, to convert solutions for MAXPES into solutions for MINEDS, Alg. 7 is sufficient. Another problem investigated in this work is MINIMUM INDEPENDENT EDGE DOMINATING SET (MINIEDS). An *independent edge dominating* set is a dominating set that is also a matching. While there is no difference between a minimum independent edge dominating set and a minimum maximal matching, not every edge dominating set is also a maximal matching. Similarly, not every maximal matching is also an edge dominating set. MAXPES, however, again serves a SCOOP twin for MINIEDS as any subset of edges of profit $\mathfrak{p}$ can be converted into a matching of profit at least $\mathfrak{p}$.

6.4. Set Cover

In the previous sections, we applied our SCOOP framework to two different graph problems. Now, we apply the SCOOP framework to a different type of constrained combinatorial optimization problem, Set Cover.

Given a universe U consisting of n elements and a family $\mathcal{C}$ consisting of m subsets of U whose union equals U, a *set cover* is a subfamily $\mathcal{C}' \subseteq \mathcal{C}$ where the union of elements in $\mathcal{C}'$ equals U. We begin by defining the problem before applying our SCOOP framework to derive and solve the problem via its SCOOP twin.

6.4.1. Minimum Set Cover (MinSC)

We define the optimization version of the problem Minimum Set Cover (MinSC) for a universe U, a family of subsets $\mathcal{C} \subseteq U$, and a subfamily $\mathcal{C}' \subseteq \mathcal{C}$ as follows:

$$\begin{aligned} &\text{Minimize: } |\mathcal{C}'| \\ &\text{Subject to: } \bigcup_{C_i \in \mathcal{C}'} C_i = U \end{aligned}$$

Cost Hamiltonian for MinSC

Lucas [Lucas, 2014] defines a QUBO for MinSC using the following binary variables:

- $y_i = \begin{cases} 1 & C_i \in \mathcal{C}' \\ 0 & C_i \notin \mathcal{C}' \end{cases}$
- $x_{\alpha,m} = \begin{cases} 1 & \text{Number of } C_i\text{'s} \in \mathcal{C} \text{ with } \alpha \in C_i \text{ is } m \geq 1 \\ 0 & \text{otherwise} \end{cases}$

The QUBO for MinSC with penalties A and B is then defined on binary variables $\vec{x}$ and $\vec{y}$ as follows:

$$\hat{H}_{SC}(\vec{x}, \vec{y}) = \hat{H}_E^{SC}(\vec{x}, \vec{y}) + \hat{H}_S^{SC}(\vec{y})$$

where

$$\hat{H}_E^{SC}(\vec{x}, \vec{y}) = A \sum_{\alpha=1}^{n} \left(1 - \sum_{m=1}^{N} x_{\alpha,m}\right)^2 + A \sum_{\alpha=1}^{n} \left(\sum_{m=1}^{N} m x_{\alpha,m} - \sum_{i:\alpha \in C_i} y_i\right)^2$$

and

$$\hat{H}_S^{SC}(\vec{y}) = B \sum_{i=1}^{N} y_i$$

For MINSC, the QUBO formulation uses *penalties* to enforce the constraint that each element of the universe U is covered. The penalty also activates when the variable $x_{\alpha,m}$ is counted more than once, which is necessary because we aim to count coverage only a single time (and penalize otherwise) even if the element appears in multiple subsets.

6.4.2. MAXIMUM PROFIT SET COVERAGE (MAXPSC)

Step 1: Identify constraints
The constraints that any solution $\mathcal{C}' \subseteq \mathcal{C}$ to MINSC must satisfy to ensure feasibility is that $\mathcal{C}'$ is a set cover for U, i.e., $\bigcup_{C_i \in \mathcal{C}'} C_i = U$.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subfamily of $\mathcal{C}$ *covers the elements of* U.

Step 3: Develop objective function $\mathtt{obj}_{\mathbf{PSC}}^{(U,\mathcal{C})}$
For a subfamily $\mathcal{PSC} \subseteq \mathcal{C}$,

$\mathtt{obj}_{\text{PSC}}^{(U,\mathcal{C})}(\mathcal{PSC}) = |\bigcup_{C_i \in \mathcal{PSC}} C_i| - |\mathcal{PSC}|$. We refer to the value produced by the objective function $\mathfrak{p}_{\text{PSC}} = \mathtt{obj}_{\text{PSC}}^{(U,\mathcal{C})}(\mathcal{PSC})$ as *profit*.

Step 4: Objective function compatibility
From Theorem 9 below we can derive function $f_U(.)$. Here, $f_U(\mathfrak{p}) = |U| - k$ and $f_U^{-1}(k) = |U| - \mathfrak{p}$.

Before moving on to Step 5, we formulate the problem MAXIMUM PROFITABLE SET COVERAGE or MAXPSC for a universe U, a family of subsets $\mathcal{C}$ and a subfamily $\mathcal{PSC} \subseteq \mathcal{U}$,

$$\text{Maximize: profit } \mathfrak{p}_{\text{PSC}}, \text{where } \mathfrak{p}_{\text{PSC}} = \mathtt{obj}_{\text{PSC}}^{(U,\mathcal{C})}(\mathcal{PSC})$$

Step 5: Solution enhance-ability
The proof of Theorem 9 below shows a process that converts any solution $\mathcal{PSC} \subseteq \mathcal{C}$ for a given MAXPSC-instance $(U, \mathcal{C})$ into a set cover while preserving solution quality.

Algorithm 10: Converting solutions: MAXPSC to MINSC

Require: Universe U, family of subsets $\mathcal{C}$, Solution $\mathcal{PSC}$
  Let U_{unc} be the set of uncovered elements in U
  for each element $x \in U_{\text{unc}}$ **do**
    for each subfamily $C_i \in \mathcal{C} \setminus \mathcal{PSC}$ **do**
      if $x \in C_i$ **then**
        Add C_i to *PSC*
      end if
    end for
  end for

Cost Hamiltonians for MAXPSC

In contrast to MINSC, we formulate MAXPSC as a HUBO that naturally captures the requirement that each element is counted only once even if it appears in multiple subsets, without the need for penalty terms. To describe MAXPSC as unconstrained binary optimization problem, we define the HUBO as follows:

Cost Hamiltonian for MAXPSC

We begin by defining the binary variables y_i expressing whether or not subset C_i is selected into the cover, and $x_{\alpha,i}$ that tells us whether or not element $\alpha \in U \cap C_i$:

- $y_i = \begin{cases} 1 & C_i \in \mathcal{C}' \\ 0 & C_i \notin \mathcal{C}' \end{cases}$
- $x_{\alpha,i} = \begin{cases} 1 & \alpha \in U \text{ and } \alpha \in C_i \text{ for } C_i \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$

The total cost to maximize for MAXPSC is

$$H_{PSC}(\vec{x},\vec{y}) = H_E^{PSC}(\vec{x},\vec{y}) - H_S^{PSC}(\vec{y}) \tag{6.5}$$

$$= \sum_{\alpha=1}^{|U|}\left(1 - \prod_{i=1}^{|\mathcal{C}|}(1 - x_{\alpha,i}y_i)\right) - \sum_{i=1}^{|\mathcal{C}|} y_i \tag{6.6}$$

where $H_S^{PSC}(\vec{y})$ corresponds to the size of $\mathcal{C}'$, and $H_E^{PSC}(\vec{x},\vec{y})$ to the number of elements of U that are covered by the chosen subsets.

6.4.2.1. Finding minimum set covers via maximum profit set coverage

We now establish the relationship between MINSC and MAXPSC. Theorem 9 demonstrates this equivalence, showing how maximum profit set coverage solutions can be transformed into minimum set covers while preserving solution quality characteristics.

Theorem 9. *For any universe U and family $\mathcal{C}$ of subsets of U whose union equals U, U has a set cover of size k if and only if there is a subfamily $\mathcal{PSC} \subseteq \mathcal{C}$ with profit $\mathfrak{p}_{PSC} = |U| - k$.*

Proof. Given: $U, \mathcal{C}$ with $\bigcup_{C_i \in \mathcal{C}} C_i = U$.

($\Rightarrow$) If there is a set cover $\mathcal{C}' \subseteq \mathcal{C}$ with $|\mathcal{C}'| = k$, then $\bigcup_{C_i \in \mathcal{C}'} C_i = U$. Thus, the profit of $\mathcal{C}'$ is $\mathfrak{p}_{\text{PSC}} = |\bigcup_{C_i \in \mathcal{C}'} C_i| - k = |U| - k$.

($\Leftarrow$) If there is a subfamily $\mathcal{PSC} \subseteq \mathcal{C}$ with profit $\mathfrak{p}_{\text{PSC}}$, then $|\bigcup_{C_i \in \mathcal{PSC}} C_i| - |\mathcal{PSC}| \geq \mathfrak{p}_{\text{PSC}}$. If $\mathcal{PSC}$ is a set cover, then $\bigcup_{C_i \in \mathcal{PSC}} C_i = U$. Thus, $|U| - |\mathcal{PSC}| \geq \mathfrak{p}_{\text{PSC}}$, yielding $|U| - \mathfrak{p}_{\text{PSC}} \geq |\mathcal{PSC}|$. If $\mathcal{PSC}$ is not a minimum set cover, then there exists $x \in U$ such that for all $C_i \subseteq \mathcal{PSC}$, $x \notin C_i$; thus, $\bigcup_{C_i \in \mathcal{PSC}} C_i \neq U$ and $|\bigcup_{C_i \in C'} C_i| < |U|$. In this case we can obtain a set cover of size at most $|U| - |\mathcal{PSC}|$ by applying Alg. 10.

□

The proof of Theorem 9, given a maximum profitable set coverage $\mathcal{PSC}$ for a universe U and family $\mathcal{C}$ of subsets of U such that $\bigcup_{C_i \in \mathcal{C}} C_i = U$, tells us that we can determine in polynomial time a minimum set cover for U using the procedure outlined in the proof above.

Up to this point, we have explored NP-hard problems. In the following section, we examine two problems that can be solved classically in polynomial time.

6.5. A polynomial-time solvable problem: Edge Cover

An *edge cover* in an undirected graph $G = (V, E)$ is a subset of edges $(\text{EC}) \subseteq E$ such that every vertex in V is incident to at least one edge in (EC). An edge cover is called *minimum edge cover* if it has the smallest possible number of edges. The *minimum edge cover* problem is polynomial time solvable. While MINEC is polynomial-time solvable [Garey and Johnson, 1979], it merits investigation in the quantum computing context for two compelling reasons. First, it provides a valuable benchmarking framework for quantum algorithms, as its classical tractability enables exact solution verification across all problem sizes. This contrasts with NP-hard problems, where classical verification becomes intractable beyond modest sizes, limiting meaningful performance comparisons. Second, it presents an academically interesting case where a polynomially-solvable problem has a higher-order Hamiltonian with constraints for quantum implementation, exhibiting non-trivial challenges, offering insights into solvability of combinatorial optimization problems with QAOA.

6.5.1. MINIMUM EDGE COVER (MINEC)

We define the problem MINIMUM EDGE COVER (MINEC) for a graph $G = (V, E)$ as follows. A subset of edges $(\text{EC}) \subseteq E$ is an *edge cover* if every vertex $v \in V$ is incident to at least one edge in (EC). The objective is to find the smallest such set:

$$\begin{aligned} &\text{Minimize:} && |(\text{EC})| \\ &\text{Subject to:} && \text{for all } v \in V \\ &&& \exists e \in \delta(v) \text{ such that } \ e \in \text{EC} \\ &&& \text{where } \delta(v) = \{e \in E : v \in e\} \end{aligned}$$

In contrast, Dinneen et al. [Dinneen and Hua, 2017b] provide a QUBO formulation for this problem. However, it uses auxiliary variables for its encoding. The QUBO is similar to that of MINDS, which we discuss in Section 6.1.

Binary variables $x_{ij} \in \{0, 1\}$ are defined for each edge $e_{ij} \in E$, with $x_{ij} = 1$ meaning that edge e_{ij} is included in the edge cover. For each vertex $v_i \in V$, let $I(v_i)$ denote the set of edges incident to v_i, and let $\deg(v_i)$ be its degree.

To avoid over-penalizing valid solutions, we introduce auxiliary binary variables $y_{i,k} \in \{0, 1\}$ for $0 \leq k \leq \lfloor \log_2(\deg(v_i) - 1) \rfloor$ to encode a balancing term.

The objective function to be minimized is:

$$F(x, y) = \sum_{e_{ij} \in E} x_{ij} + A \sum_{v_i \in V} P_i, \tag{6.7}$$

where

$$P_i = \left(1 - \sum_{e_{ij} \in I(v_i)} x_{ij} + \sum_{k=0}^{\lfloor \log_2(\deg(v_i)-1) \rfloor} 2^k y_{i,k}\right)^2, \tag{6.8}$$

and $A > 1$ is a penalty parameter that ensures the cover constraints are satisfied. Here, $\sum x_{ij}$ penalizes the total number of edges selected, promoting smaller edge covers. P_i enforces that each vertex is covered by at least one selected edge. The auxiliary $y_{i,k}$ variables help balance the penalty if more than one incident edge is selected for a vertex.

A feasible edge cover $C(x^*) \subseteq E$ is extracted from the optimal assignment as:

$$C(x^*) = \{e_{ij} \in E \mid x_{ij} = 1\}.$$

This QUBO formulation requires $O(m + n \log n)$ binary variables for a graph with m edges and n vertices.

6.5.1.1. Cost Hamiltonian for MinEC

We describe cost Hamiltonians for MinEC ($\hat{H}_{\text{EC}}$) using a higher-order unconstrained binary formulation. Let $\vec{x} = (x_1, x_2, \ldots, x_{|E|})$ be a vector of binary variables, where $x_i \in \{0, 1\}$ for each $i \in E$, then,

$$\begin{aligned}\hat{H}_{\text{EC}}(\vec{x}) &= \hat{H}_N^{\text{EC}}(\vec{x}) + \hat{H}_V^{\text{EC}}(\vec{x}) \\ &= A \sum_{i \in V} \left[\Big(\prod_{j \in \delta(i)} (1 - x_j) \Big) \right] + B \sum_{i \in E} x_i,\end{aligned}$$

The first term, $\hat{H}_N^{\text{EC}}(\vec{x})$ ensures that every vertex $v \in V$ is incident to at least one edge in the edge cover, while the second term, $\hat{H}_V^{\text{EC}}(\vec{x})$, minimizes the number of edges in the edge cover.

6.5.2. Maximum Profit Edge Cover (MaxPEC)

Using our SCOOP framework for MinEC, we derive the unconstrained problem Maximum Profit Edge Cover or MaxPEC. Theorem 10 below concludes the proof that MinEC and MaxPEC are SCOOP twins.

Step 1: Identify constraints
The constraint that MinEC must satisfy to ensure feasibility is that the subset (EC) $\subseteq E$ covers all vertices in V.

Step 2: Quantify feasibility of infeasible solution
Consider the *extent* to which a subset of edges *covers vertices* and its *closeness* to being an edge cover (i.e., the more vertices are covered, the better).

Step 3: Develop objective function $\texttt{obj}^G_{\text{PEC}}$
For a subset PEC $\subseteq E$ in G,

Let $V_{\text{PEC}}(G, \text{PEC})$ denote the set of vertices covered by a subset PEC $\subseteq E$ in graph G,

$$V_{\text{PEC}}(G, \text{PEC}) = \{v \in V \mid v \text{ is covered by PEC}\}$$

$\texttt{obj}^G_{\text{PEC}}(PEC) = |V_{\text{PEC}}(G, \text{PEC})| - |\text{PEC}|$. We also refer to the value produced by the objective function $\texttt{obj}^G_{\text{PEC}}(\text{PEC})$ as *profit*, $\mathfrak{p}_{\text{PEC}}$.

We formulate the problem MAXIMUM PROFIT EDGE COVER (MAXPEC) for a graph $G = (V, E)$ and PEC $\subseteq E$:

$$\text{Maximize: profit } \mathfrak{p}_{\text{PEC}}, \text{where } \mathfrak{p}_{\text{PEC}} = \texttt{obj}^G_{\text{PEC}}(PEC)$$

Step 4: Objective function compatibility
From Theorem 10 below, we can derive function $f_G(.)$. That is, $f_G(\mathfrak{p}) = |V| - k$ and $f_G^{-1}(k) = |V| - \mathfrak{p}$.

Step 5: Solution enhance-ability
Alg. 11 describes a polynomial-time post-processing procedure that converts a solution PEC for MAXPEC to an edge cover, i.e., a solution for MINEC while preserving solution quality.

Algorithm 11: Converting solutions of MAXPEC to MINEC

Require: Graph G, Solution PEC
  Let V_{nc} be the set of non-covered vertices in G
  for each vertex $v \in V_{\text{nc}}$ **do**
    Add an edge (v, u) to PEC, where u is any vertex in $\delta(v)$
  end for

Cost Hamiltonian for MAXPEC

We describe the cost Hamiltonian for MAXPEC using higher-order unconstrained binary formulation. Let $\vec{x} = (x_1, x_2, \ldots, x_{|E|})$ be a vector of binary variables, where $x_i \in \{0, 1\}$ for each $i \in E$, then,

Cost Hamiltonian for MAXPEC

Neighborhood cost:

$$\hat{H}_N^{\text{PEC}}(\vec{x}) = \sum_{i \in V} \left[1 - \Big(\prod_{j \in \delta(i)} (1 - x_j) \Big) \right]$$

Vertex cost:

$$\hat{H}_E^{\text{PEC}}(\vec{x}) = \sum_{i \in E} x_i$$

The total cost that is maximized for MAXPD is

$$\hat{H}_{\text{PD}}(\vec{x}) = \hat{H}_N^{\text{PEC}}(\vec{x}) - \hat{H}_V^{PEC}(\vec{x}) \tag{6.9}$$

The first term, $\hat{H}_N^{\text{PEC}}(\vec{x})$ counts the number of vertices $i \in V$ that are incident to at least one edge in the edge cover, while the second term, $\hat{H}_E^{\text{PEC}}(\vec{x})$ counts the number of edges in the edge cover. The objective is to maximize the profit, which is equivalent to maximizing the cost Hamiltonian $\hat{H}_{\text{PEC}}(\vec{x})$.

6.5.2.1. Finding minimum edge covers via maximum profit edge covers

We now establish the relationship between MINEC and MAXPEC. Theorem 10 demonstrates this equivalence, showing how maximum profit edge covers can be transformed into minimum edge covers while preserving solution quality characteristics.

Theorem 10. *For any graph $G = (V, E)$, G has an edge cover $(EC) \subseteq E$ of size k if and only if G has a subset PEC $\subseteq E$ with profit $\mathfrak{p}_{PEC} = |V| - k$.*

Proof.

($\Rightarrow$) Consider an edge cover EC $\subseteq E$, with $|\text{EC}| = k$. The profit of EC is $\mathfrak{p}_{\text{PEC}} = |V_{\text{PEC}}(G, \text{EC})| - |\text{EC}|$. Since EC is a feasible edge cover, $V_{\text{PEC}}(G, \text{EC}) = V$, and $\mathfrak{p}_{\text{PEC}} = |V| - k$.

($\Leftarrow$)

Consider a profit edge cover, PEC $\subseteq E$ with a profit $\mathfrak{p}_{\text{PEC}} = |V| - k$. In the case where PEC is an edge cover, there is nothing to prove as PEC covers all vertices.

If instead PEC is not an edge cover, then we need to prove that we can obtain an edge cover of size at most $|V| - \mathfrak{p}_{\text{PEC}}$. For PEC, let V_{nc} be the set of uncovered vertices, i.e., $V_{nc} = V - V_{\text{PEC}}$.

A vertex $v \in V_{nc}$ implies that none of its incident edges belongs to the profit edge cover, PEC. Since covering a vertex requires only a single incident edge, we can arbitrarily select any edge $e \in E$ incident to v and incorporate it into PEC. This expansion process preserves

the profit value, as each edge addition corresponds to at least one newly covered vertex. We repeat this procedure until all vertices are covered. The formal procedure is detailed in Alg. 11.

□

6.6. A polynomial-time solvable problem: Maximum Matching

Recall from Section 6.2, *matching* in an undirected graph $G = (V, E)$ is a subset of edges $\mathcal{M} \subseteq E$ such that no two edges in $\mathcal{M}$ share a vertex. A matching is called *maximum matching* if it has the largest possible number of edges. MINIMUM MAXIMAL MATCHING (M^3) is an NP-hard problem described in Section 6.2. Here we describe the polynomial time solvable MAXIMUM MATCHING.

The MAXIMUM MATCHING problem differs from maximal matching in that it seeks the matching of maximum cardinality. While finding a minimum maximal matching is NP-hard, the MAXIMUM MATCHING problem is polynomial-time solvable, with strong connections to the edge cover problem. Edmonds' work in 1965 [Edmonds, 1965] provided the first polynomial-time algorithm with $O(n^4)$ complexity. Subsequent improvements have yielded increasingly efficient algorithms [Kameda and Munro, 1974, Micali and Vazirani, 1980].

Despite its classical tractability, examining MAXIMUM MATCHING from a quantum perspective offers valuable insights. Of particular interest is its relationship with the edge cover problem: traditionally, maximum matchings serve as a foundation for constructing edge covers through post-processing of uncovered vertices. Our approach inverts this relationship, deriving maximum matchings through a chain of transformations: profit edge cover $\rightarrow$ edge cover $\rightarrow$ maximum matching.

Benchmarking large problem instances that are classically intractable is a significant challenge in quantum computing, as we lack reference solutions to evaluate the performance of quantum techniques. However, the MAXM problem is solvable in polynomial time, yet its Hamiltonian structure is similar to those of M^3, MINDS, and MINIEDS. As a result, obtaining results for QAOA on MAXM can provide valuable insights into where QAOA performs well or faces challenges.

6.6.1. MAXIMUM MATCHING (MAXM)

We define the problem MAXIMUM MATCHING (MAXM) for a graph $G = (V, E)$ as follows. A subset of edges $\mathcal{M} \subseteq E$ is a *matching* if no two edges in $\mathcal{M}$ share a vertex. The objective is to find the largest such set:

$$\begin{aligned} &\text{Maximize:} \quad |\mathcal{M}| \\ &\text{Subject to:} \quad \text{for all } v \in V \\ &\qquad\qquad \sum_{e \in \delta(v)} x_e \leq 1 \\ &\qquad\qquad \text{where } \delta(v) = \{e \in E : v \in e\} \end{aligned}$$

6.6.2. Maximum Matchings via Profit Edge Cover

Using the unconstrained problem MAXPEC, our approach leverages the relationship between profit edge covers and matchings through the following chain of transformations. Given a profit edge cover (PEC), we can derive an edge cover (EC), which in turn yields a matching.

Theorem 11. *For any graph $G = (V, E)$, G has a matching $\mathcal{M} \subseteq E$ of size k if and only if G has a subset PEC $\subseteq E$ with profit $\mathfrak{p}_{PEC} = |V| - k$.*

Proof.

($\Rightarrow$) Consider a matching $\mathcal{M} \subseteq E$, with $|\mathcal{M}| = k$. We distinguish two cases:

Case 1: $\mathcal{M}$ is a feasible edge cover and therefore a profit edge cover, i.e., every vertex in V is incident to at least one edge in $\mathcal{M}$. In this case, we can define a profit edge cover PEC $= \mathcal{M}$, which has a profit of $\mathfrak{p}_{\text{PEC}} = |V| - k$.

Case 2: $\mathcal{M}$ is not a feasible edge cover, i.e., there exists at least one vertex $v \in V$ that is not incident to any edge in $\mathcal{M}$. In this case, we construct an edge cover, EC, by adding edges incident to the uncovered vertices using Alg. 11. The resulting edge cover EC has a profit no worse than $\mathfrak{p}_{\text{PEC}} \geq |V| - k$.

($\Leftarrow$) Consider a profit edge cover, PEC $\subseteq E$, |PEC| $= k$, with a profit $\mathfrak{p}_{\text{PEC}} = |V| - k$.

Case 1: If PEC is a matching and an edge cover, then we are done as PEC is a matching of size k.

Case 2: If PEC is not a matching but is an edge cover, then we need to prove that we can obtain a matching of size at least k. If PEC is not a matching, that means there exists at least one vertex $v \in V$ that is incident to more than one edge in PEC. We can select one edge incident to v and remove the others. The profit value remains unchanged. The proof of correctness is as follows: Let (u, v) be the edge selected to remain in PEC. Let (v, w) be an edge in PEC incident to v. Now, (v, w) can be safely removed from PEC (does not affect the profit value) as it would only affect coverage of the one endpoint, w (as v is already covered by (u, v)).

Case 3: If PEC is not an edge cover, then we need to prove that we can obtain a matching of size at least k. For PEC, let V_{nc} be the set of uncovered vertices, i.e., $V_{nc} = V \setminus V_{\text{PEC}}$.Then we can use the procedure outlined in Alg. 11 to add edges incident to the uncovered vertices until all vertices are covered to form an edge cover. This would yield a matching of size at least k as it falls under the same logic as the above two cases. □

The quality of the derived matching depends on the profit value of the initial PEC:

- When PEC achieves maximum profit, it corresponds to a minimum edge cover, from which we can derive a maximum matching

- For suboptimal profit values, corresponding edge covers would derive matchings

6.7. Chapter Summary

This chapter extended the SCOOP framework to problems requiring higher-order (HUBO) formulations, demonstrating its versatility across both NP-hard and polynomial-time solvable problems. While higher-order formulations can be quadratized using auxiliary variables, we focused on problems whose natural formulations inherently require higher-order terms to capture their underlying structure.

We began by analyzing selected NP-hard graph problems, focusing on dominating set variants and matching problems that inherently require higher-order Hamiltonian formulations. These formulations demonstrated how our framework handles higher-order constraints while maintaining solution quality characteristics. We explored the Minimum Set Cover problem and found it noteworthy that its counterpart, the Maximum Set Packing problem, can also be expressed as a QUBO formulation. We explored the MINIMUM SET COVER HUBO and find it noteworthy that its problem pair, the MAXIMUM SET PACKING problem, has a QUBO formulation. We then explored two polynomial-time solvable problems, specifically MINIMUM EDGE COVER and MAXIMUM MATCHING. The successful application of SCOOP to these higher-order problems establishes its broader utility as a framework for quantum optimization, setting the stage for experimental evaluation in subsequent chapters.

Part III

Evaluation

Chapter 7

Experimental Setup and Methodology

Contents

Chapter Contributions

C9: *Development of software infrastructure for the SCOOP framework:*

- C9.1: *Implementation of a suite of QUBO/HUBO encodings of SCOOP twins in PennyLane, Qiskit and QTensor*
- C9.2: *Creation of test datasets with random graphs of different sizes ($n = 5$ to $n = 70$) of varying densities ($d = [0.1, 0.3, 0.5, 0.8]$) and 3-regular graphs for benchmarking*
- C9.3: *Extensions to QTensor's architecture to constrained optimization problems*
- C9.4: *Classical post-processing routines for assessing solution feasibility*

This chapter describes the experimental setup and methodology used to evaluate the performance of our proposed approach for solving unconstrained SCOOP problem twins of

constrained combinatorial optimization problems using quantum approximate optimization algorithm (QAOA). We focus on comparing the selected QUBO and HUBO profit formulations with penalty-term constrained formulations. We provide the experimental setup to evaluate/compare our formulations, including the cost Hamiltonians, mixer Hamiltonian, and classical algorithms to calculate the exact reference solutions and the evaluation platforms used.

Our primary goal is to evaluate and contrast the quality of solutions obtained for profit formulations and penalty-term constrained formulations using QAOA.

Koch et al. [Koch et al., 2025] outline three key metrics for benchmarking quantum combinatorial optimization algorithms: solution quality, computational resource requirements, and quantum hardware runtime measurements. We present the results of our simulator experiments in Chapter 8 and quantum hardware experiments in Chapter 9.

For solution quality assessment, we examine the summed probabilities (alternatively, winning probabilities) of optimal and near-optimal solutions. Unlike penalty-based approaches where solutions must be classified as both feasible and optimal, our SCOOP framework guarantees feasibility, allowing us to focus solely on optimality measures. We evaluate solution quality through probability distributions of solutions, summed probabilities of optimal solutions, expectation values, and approximation ratios across different problem instances.

Regarding computational resources, rather than making exact timing measurements, we use problem size to determine the appropriate simulation approach. We perform full statevector simulation for smaller instances (up to 14 nodes) and tensor network simulation for larger instances (15-70 nodes) due to the practical limitations of classical simulation resources while enabling evaluation across a broad range of problem sizes.

7.1. QAOA Setup

In this section, we describe the cost Hamiltonians used for selected constrained and unconstrained SCOOP problem twins. For every binary variable x_i in the QUBO/HUBO formulations, we use spin variables s_i to encode the variable in the Hamiltonian using the transformation $x_i \to \frac{1-s_i}{2}$ (cf. Section 2.4). These spin variables $s_i \in \{-1, 1\}$ can then be reformulated to Pauli-Z operators, where each s_i is replaced by the corresponding 2×2 Pauli matrix:

$$\hat{Z}_i = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{7.1}$$

The operators $\hat{Z}_i$ act on the quantum computer's computational space — a Hilbert space of size $2^n \times 2^n$ for n qubits. Products of Pauli operators like $\hat{Z}_i\hat{Z}_j$ are understood as tensor products $\hat{Z}_i \otimes \hat{Z}_j$ embedded in this larger Hilbert space. The cost Hamiltonians are derived

from the corresponding QUBO/HUBO formulations of the problems, as described in Chapters 5 and 6. The cost Hamiltonians are used in the QAOA algorithm to find solutions to the profit relaxation of the constrained problems.

7.1.1. Cost Hamiltonians for selected CCOPs and their SCOOP problem twins

Cost Hamiltonians for MinVC and MaxPC

The cost Hamiltonian for MinVC is derived from the QUBO formulation in Eqn. 3.3 and is based on the profit relaxation of the problem. The profit relaxation of MinVC is defined as follows:

$$\hat{H}_{\text{VC}}(\vec{s}) = \frac{A}{4} \sum_{uv \in E} (s_u s_v + s_u + s_v) - \frac{B}{2} \sum_{v \in V} s_v + \Delta_{\text{VC}}. \tag{7.2}$$

Here, s_u and s_v are Ising variables that take the values $s_i \in \{+1, -1\}$ compared to $x_i \in \{0, 1\}$.

Formulated using the Pauli-Z operators, the Hamiltonian $\hat{H}_{\text{VC}}$ can be expressed as:

$$\hat{H}_{\text{VC}}^{\min} = \frac{A}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v + \hat{Z}_u + \hat{Z}_v) - \frac{B}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{VC}} \tag{7.3}$$

Here, $\Delta_{\text{VC}} = \frac{1}{4}A|E| + \frac{1}{2}B|V|$, and penalties A and B in $\hat{H}_{\text{VC}}^{\min}$ are set to 3 and 2, respectively, unless stated otherwise. Note that $\frac{\hat{H}_{\text{VC}}}{B}$ corresponds to the size of the solution state for (only) the cases that correspond to a feasible vertex cover solution.

Our choice of penalties is based on PennyLane's implementation [Xanadu, 2024] of MinVC. It is difficult to set desirable penalties [Glover et al., 2022], as they depend on the particular input graphs. Small penalties increases the likelihood of infeasible solutions being returned and values that are too large may lead to slower convergence. For the subsequent problem formulations, we express the Hamiltonians using Pauli $\hat{Z}_i$ operators. The corresponding Hamiltonian $\hat{H}_{\text{PC}}$ for MaxPC, is based on its QUBO as in Eqn. 3.5:

$$\hat{H}_{\text{PC}}^{\min} = \frac{1}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v + \hat{Z}_u + \hat{Z}_v) - \frac{1}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{PC}} \tag{7.4}$$

where $\Delta_{\text{PC}} = \frac{|V|}{2} - \frac{3|E|}{4}$. $\hat{H}_{\text{PC}}$ can be obtained by setting $A = B = 1$ in Eqn. 7.2, but it is important to highlight that this does not imply that all constrained problems can be transformed into profit problems by setting penalties to 1. For instance, setting $A = 1$ in the Dominating Set QUBO [Dinneen and Hua, 2017b] does not transform it to the QUBO for Profit Domination.

Any solution obtained using the maximum profit Hamiltonian is either a vertex cover or can be converted into a vertex cover by a classical polynomial-time post-processing step, and without loss of profit (cf. Theorem 1).

Cost Hamiltonians for MaxIS and MaxPI

The following cost Hamiltonians are used for MaxIS and MaxPI:

$$\hat{H}_{\text{IS}}^{\min} = \frac{A}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v - \hat{Z}_u - \hat{Z}_v) + \frac{B}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{IS}} \tag{7.5}$$

where $\Delta_{\text{IS}} = \frac{A}{4}|E| - \frac{B}{2}|V|$.

$$\hat{H}_{\text{PI}}^{\min} = \frac{1}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v - \hat{Z}_u - \hat{Z}_v) + \frac{1}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{PI}} \tag{7.6}$$

where $\Delta_{\text{PI}} = \frac{1}{4}|E| - \frac{1}{2}|V|$.

Cost Hamiltonians for MaxCl and MaxPCl

Similar Cost Hamiltonians are used for MaxCl and MaxPCl.

$$\hat{H}_{\text{Cl}}^{\min} = \frac{A}{4} \sum_{uv \in (V \times V) \setminus E} (\hat{Z}_u \hat{Z}_v - \hat{Z}_u - \hat{Z}_v) + \frac{B}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{Cl}} \tag{7.7}$$

where $\Delta_{\text{Cl}} = \frac{A}{4}|(V \times V) \setminus E| - \frac{B}{2}|V|$.

$$\hat{H}_{\text{PCl}}^{\min} = \frac{1}{4} \sum_{uv \in (V \times V) \setminus E} (\hat{Z}_u \hat{Z}_v - \hat{Z}_u - \hat{Z}_v) + \frac{1}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{PCl}} \tag{7.8}$$

where $\Delta_{\text{PCl}} = \frac{1}{4}|(V \times V) \setminus E| - \frac{1}{2}|V|$.

Note that all Cost Hamiltonians are formulated as minimization problems.

Cost Hamiltonians for MinDS and MaxPD

The cost Hamiltonians for MinDS and MaxPD are defined as follows:

$$\hat{H}_{DS}^{\min} = A \sum_{i \in V} \left[\left(\frac{1 + \hat{Z}_i}{2} \right) \prod_{j \in N(i)} \left(\frac{1 + \hat{Z}_j}{2} \right) \right] + B \sum_{i \in V} \frac{1 - \hat{Z}_i}{2} \tag{7.9}$$

$$\hat{H}_{PD}^{\max} = \sum_{i \in V} \left[1 - \left(\left(\frac{1 + \hat{Z}_i}{2} \right) \prod_{j \in N(i)} \left(\frac{1 + \hat{Z}_j}{2} \right) \right) \right] - \sum_{i \in V} \frac{1 - \hat{Z}_i}{2} \tag{7.10}$$

where $N(i)$ is the set of neighbours of vertex i in the graph. The parameters A and B are set to 3 and 2, respectively, unless stated otherwise.

Note that Eqn. 7.10 is for maximization. Since we use minimization versions for all problems, we define that as follows:

$$\hat{H}_{PD}^{\min} = (-1) * \hat{H}_{PD}^{\min} = = \sum_{i \in V} \left(\frac{1 - \hat{Z}_i}{2} \right) - \sum_{i \in V} \left[1 - \left(\left(\frac{1 + \hat{Z}_i}{2} \right) \prod_{j \in N(i)} \left(\frac{1 + \hat{Z}_j}{2} \right) \right) \right] \tag{7.11}$$

$$= \sum_{i \in V} \left[\left(\frac{1 + \hat{Z}_i}{2} \right) \prod_{j \in N(i)} \left(\frac{1 + \hat{Z}_j}{2} \right) \right] - \sum_{i \in V} \left(\frac{\hat{Z}_i}{2} \right) \tag{7.12}$$

Cost Hamiltonian for MAXPES

$$\hat{H}_{PES}^{\min} = \sum_{i \in E} \left[\left(\frac{1 + \hat{Z}_i}{2} \right) \prod_{j \in N_e(i)} \left(\frac{1 + \hat{Z}_j}{2} \right) \right] - \sum_{i \in E} \left(\frac{\hat{Z}_i}{2} \right) \tag{7.13}$$

We do not provide the cost Hamiltonian for M^3 as defined in Section 6.2.1, as the appropriate penalty parameter settings for this higher-order formulation require further investigation.

7.1.2. Mixer Hamiltonian

The mixer Hamiltonian plays a crucial role in QAOA by preventing the system from getting trapped in eigenstates of the cost Hamiltonian. Without the mixer, applying only the cost unitary operator $e^{-i\gamma \hat{H}_C}$ to an eigenstate $|\psi\rangle$ of $\hat{H}_C$ would leave the state unchanged. By choosing a mixer that anti-commutes with the cost Hamiltonian, we ensure the system can escape local optima and explore the full solution space.

For unconstrained optimization problems, the Pauli-X mixer is effective as it transitions between all computational basis states. We use a basic Pauli-X mixer Hamiltonian as follows:

$$\hat{H}_M = \sum_i \hat{X}_i$$

The mixer Hamiltonian is used to explore the parameter space during the QAOA optimization process. The corresponding unitary operator is given by:

$$U_M(\beta) = e^{-i\beta \hat{H}_M} = \prod_i e^{-i\beta \hat{X}_i}$$

where $\hat{X}_i$ is the Pauli-X operator acting on qubit i. The initial state used in QAOA is typically the uniform superposition state $|+\rangle^{\otimes n}$, which is the ground state of $\hat{H}_M$. The Pauli-X mixer is particularly well-suited for unconstrained problems for several reasons: it enables transitions between all computational basis states for full solution space exploration, is

hardware-efficient requiring only local single-qubit rotations, and does not need to restrict state evolution to a subspace of valid configurations when all solutions are feasible. The SCOOP formulations of constrained problems are unconstrained, therefore, the standard Pauli-X mixer is used. To enable a fair comparison with the penalty-term formulations of constrained optimization problems, we use the same mixer Hamiltonian for both profit and penalty-term formulations.

7.1.3. Initial Parameters

The initial parameters for the QAOA algorithm are set with the following strategy:

- γ parameters: p evenly spaced values from 0 to π
- β parameters: p evenly spaced values from 0 to π

The parameters are subsequently refined using classical optimization methods such as RMSProp to calculate optimal parameters quantum circuit.

7.1.4. QAOA Circuit

This section describes the QAOA circuit used to solve the profit relaxation of the constrained problems. The QAOA circuit consists of alternating layers of cost and mixer Hamiltonians, where each layer is parameterized by angles γ and β.

7.2. Evaluation on the Xanadu PennyLane and Argonne QTensor Simulator platforms

7.2.1. Xanadu PennyLane

Fig. 7.1 shows the class diagram of our PennyLane implementation as well as functionality used to calculate exact reference solutions. The architecture separates the core QAOA components from problem-specific implementations, allowing easy addition of new problem types. This design enables seamless integration of both QUBO and HUBO formulations within the same framework, facilitating comparative analysis between penalty-based and SCOOP approaches. Key components of the PennyLane implementation can be found in Appendix B.

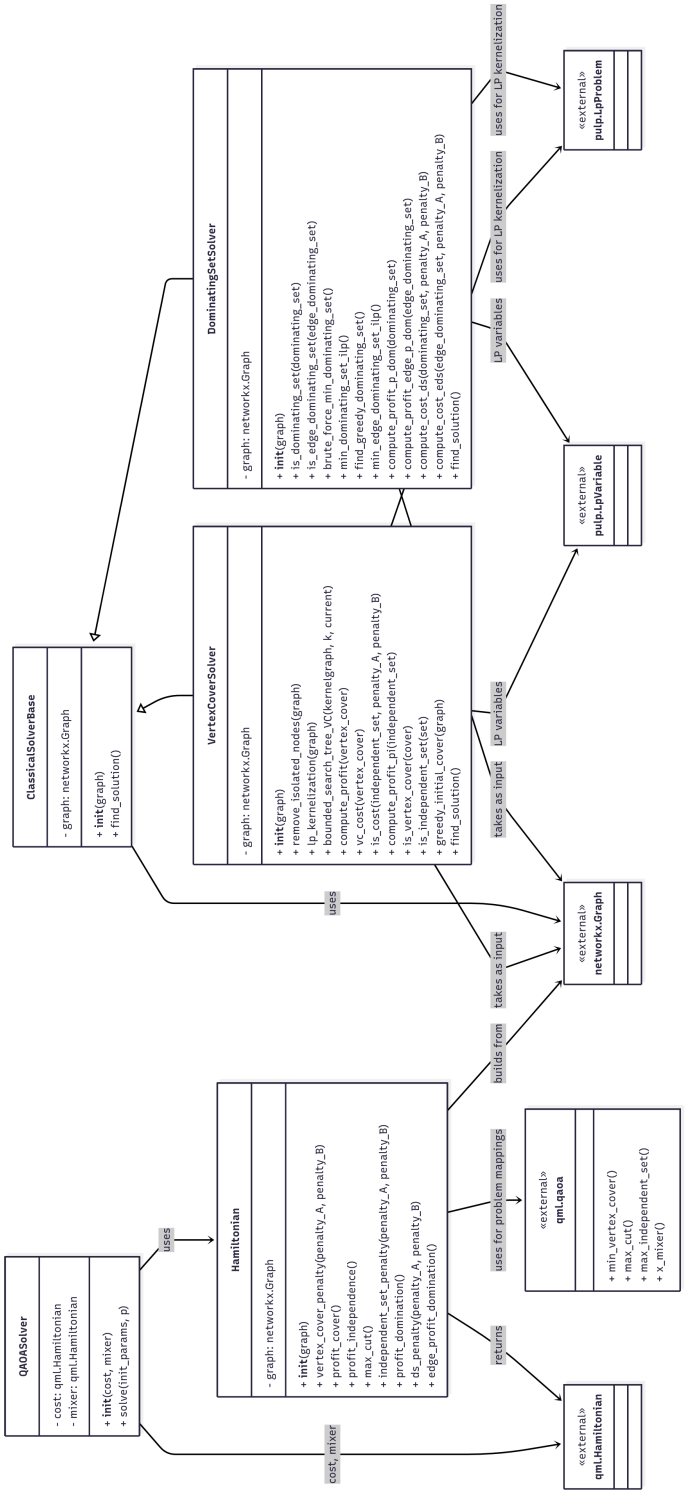


Figure 7.1.|: Experimental setup on Pennylane

7.2.2. Argonne QTensor Quantum Circuit Simulator

Argonne QTensor Simulator [Lykov et al., 2021, Lykov et al., 2022] is a highly efficient quantum circuit simulator library which is founded upon the tensor network contraction technique [Markov and Shi, 2008], offering a significant performance speedup compared to existing simulators (e.g., those that run a full amplitude-vector evolution).

QTensor serves as our primary simulation framework for evaluating SCOOP on larger problem instances (15-70 qubits) where full statevector simulation becomes infeasible. By leveraging tensor network techniques, QTensor enables efficient simulation of quantum circuits through strategic contraction of tensor networks. We extend QTensor's Circuit Composer framework to implement both constrained problems and their SCOOP variants, utilizing its QAOAComposer for circuit construction and QAOASimulator for execution. The framework's ability to handle larger qubit counts while maintaining reasonable computational resources is crucial for validating SCOOP's scalability beyond the limits of traditional quantum simulation approaches. Further details on the implementation of SCOOP problems on the QTensor simulator can be found in Appendix C

For smaller problem instances involving graphs with fewer than 14 nodes, we use PennyLane to perform full state vector simulation. For larger problem instances (up to 70 nodes), we use tensor network expectation value simulation with QTensor due to its memory efficiency and scalability. PennyLane code[1] and QTensor code used in this study will be made available on GitHub.[2] To validate our proposed approach, we employ the following methodology:

1. A set of random connected graphs of varying size and density are generated.

2. The given problem is solved using QAOA on the set of random graphs using our profit relaxation approach, as well as the penalty-term formulation of MinVC.

3. Results from our approach and results from the penalty-term QAOA are compared against classical exact solutions for varying graph sizes and densities.

7.2.2.1. Choice and density of graphs

For PennyLane, we used random connected Erdős-Rényi [Erd́os et al., 1960] graphs of varying densities (with edge probabilities of $0.1, 0.3, 0.5, 0.8$ and 3-regular graphs). For QTensor, we chose to work with sparse graphs (edge probability=0.1), and 3-regular graphs due to the large size of the graphs being simulated.

[1] https://github.com/RigiResearch/pf-opt
[2] https://github.com/danlkv/QTensor

7.2.2.2. Depth of circuits

For graphs with fewer that $n = 14$ nodes, we performed experiments with up to eight layers of QAOA. For larger graphs on QTensor we performed simulations with up to five layers.

7.2.2.3. Classical optimizer

We have used both Gradient Descent and a Gradient Descent based optimizer RMSProp (Root Mean Square Propagation) on PennyLane and QTensor. Both are similar optimizers based on gradients, with a fixed learning rate on Gradient Descent, and an adaptive learning rate on RMSProp. The results presented here are obtained using RMSProp, which offers more sophisticated parameter adaptation compared to standard gradient descent.

7.2.3. Contributions to QTensor Simulator

QTensor enables efficient classical simulation of quantum circuits using tensor network contractions and is well studied for the MaxCut problem with QAOA [Lykov et al., 2021]. The design principles of QTensor allow for ease of extensibility of the framework with minimal changes to the underlying QTensor architecture. The primary components of QTensor include *Composers* and *Simulators*. Composers allow for the creation of quantum circuits in a way that is backend-agnostic. These circuits can then be run using Simulators of popular libraries such as Qiskit and Cirq, as well as the QTensor tensor network simulator. We extend this framework to include the vertex cover and profit cover problems and their expectation value calculation on QTensor. Because our Hamiltonians contain four distinct terms, calculating an expectation value requires four calls to the simulator. To alleviate this, we consolidate the node contributions, for example:

$$\hat{H}_{\text{PC}} = \frac{1}{4} \sum_{(i,j)\in E} \left[\hat{Z}_i \hat{Z}_j\right] + \frac{1}{4} \sum_{i\in V} (\deg(i) - 2) \cdot \hat{Z}_i \tag{7.14}$$

$$\hat{H}_{\text{PI}} = \frac{1}{4} \sum_{(i,j)\in E} \left[\hat{Z}_i \hat{Z}_j\right] - \frac{1}{4} \sum_{i\in V} (\deg(i) - 2) \cdot \hat{Z}_i \tag{7.15}$$

7.2.4. Classical Post-processing

To convert the results for the unconstrained profit problem (i.e., solutions for P_U) to results of the constrained problem (P_C), we apply polynomial-time classical post-processing that runs in $\mathcal{O}(|E|)$ time. We apply Alg. 1 above based on Theorem 1 in Chapter 3 on solutions to MaxPC to obtain the corresponding solutions to MinVC. Similarly, Alg. 2 above based

on Theorem 2 is used to obtain solutions to MAXIS. To find solutions for MAXCL, we use the same Alg. 2 on the graph $G_C = (V, (V \times V) \setminus E)$ (the complement graph).

These polynomial-time algorithms guarantee that the profit of the post-processed result is at least as high as the profit of the input to the classical post-processing and ensure that the post-processed result is a feasible solution to the constrained optimization problem.

7.2.5. Classical Algorithms for Exact Reference Solutions

MinVC, MaxIS, MaxCl

To compare the results of our approach to optimum solutions for MINVC inputs, we used the implementation of an exact classical algorithm, as described below. The solutions obtained from this algorithm serve as benchmarks to evaluate the solution quality of our approach.

To obtain exact solutions for the MINVC problem, we employ a linear programming based problem kernel approach and use an implementation by Abu-Khzam of this reduction rule [Fomin et al., 2019, Abu-Khzam et al., 2005], combined with a basic bounded search-tree fixed-parameter algorithmic approach [Downey et al., 1999a, Downey and Fellows, 2013a]. Our (classical) algorithm is listed below as Alg. 12, and additional background on fixed-parameter tractability and classical algorithms is available in Chapter 2. Implementation details can be found in Appendix A.

Before computing the bounded search tree, we determine an upper bound of the size of the vertex cover using a greedy heuristic method, by repeatedly selecting vertices with the highest degree until a vertex cover is complete. Reference solutions for MAXIS and MAXCL are also obtained by running Alg. 12 with additional steps guided by the relationships among all three problems (cf. Section 5.3).

To find maximum independent sets and maximum cliques, we use Alg. 12 with some post-processing:

1. MAXIS: $IS = V \setminus \text{MinVC}(G)$
2. MAXCL: $MCl = V \setminus \text{MinVC}(G_C)$

MINDS and MAXPD

To find exact solutions for MINDS and MAXPD, we use the Algorithm described in Alg. 13. This algorithm is based on simple reduction rules and uses a recursive approach to find the minimum dominating set. The algorithm works by iteratively applying reduction rules to vertices with degree 1 or 2, and then applying an Integer Linear Programming (ILP) solver to find the minimum dominating set for the remaining graph.

Algorithm 12: MINVC(G):
Exact classical algorithm

$V_1, G' \leftarrow$ LPKernelization(G)
$V_2 \leftarrow$ GreedyVertexCover(G')
$k \leftarrow |V_2|$
$V_{\text{best}} \leftarrow V_1 \cup V_2$
$k_{\text{best}} = k - 1$
while $k_{\text{best}} \geq 0$ **do**
  $V_k \leftarrow$ BoundedSearchTree(G', k_{best})
  if $V_k = \emptyset$ **then**
    return V_{best}
  else
    $k_{\text{best}} \leftarrow k_{\text{best}} - 1$
    $V_{\text{best}} = V_1 \cup V_k$
  end if
end while
return V_{best}

Algorithm 13: MinDS(G): Exact Classical Algorithm

if $V = \emptyset$ **then**
  return $\emptyset$
end if
for $v \in V$ **do**
  if degree(v) $= 1$ **then**
    $u \leftarrow$ neighbor(v)
    $G' \leftarrow G[V \setminus \{v\}]$
    return $\{u\} \cup$ MinDS(G')
  end if
end for
for $v \in V$ **do**
  if degree(v) $= 2$ **then**
    $\{u_1, u_2\} \leftarrow$ neighbors(v)
    $D_1 \leftarrow \{u_1\} \cup$ MinDS($G[V \setminus \{v\}]$)
    $D_2 \leftarrow \{v\} \cup$ MinDS($G[V \setminus \{u_1, u_2\}]$)
    $D_3 \leftarrow \{u_2\} \cup$ MinDS($G[V \setminus \{v, u_1\}]$)
    return $\min\{D_1, D_2, D_3\}$ {Return smallest set}
  end if
end for
return ILP DominatingSet(G)

7.2.6. Evaluation of the SCOOP Framework

To evaluate the performance of the SCOOP framework, we compared the results obtained from our profit relaxation approach with those obtained from the penalty-term constrained formulations. We evaluated our approach across different problem sizes and graph densities, for different number of QAOA layers to provide a comprehensive assessment of our approach. To evaluate our hybrid approach to find small vertex covers via large profit covers, we compare the results for the constrained problem and its SCOOP twin on both PennyLane (less than 15 nodes) and QTensor (20-70 nodes). For smaller problem instances, we use a combination of probabilities, summed optimal probabilities, summed near-optimal probabilities, and mean approximation ratio to evaluate performance. For larger graphs, a full statevector simulation is not possible due to an exponential number of states to evaluate. In this case, we perform an expectation value analysis using the Argonne QTensor Tensor network simulator.

Both the constrained problem and its unconstrained SCOOP twin are solved on PennyLane as well as QTensor using the same initial parameters and classical optimizer with 100 or more iterations.

- **Probabilities**: In our analysis, we examine the individual probabilities of obtaining feasible solutions, comparing the constrained formulations and unconstrained SCOOP formulations using the standard QAOA implementation.

- **Summed optimal probability (maximum success probabililty)**: We calculate the summed probabilities of obtaining optimal solutions for both the constrained and unconstrained formulations. This metric provides insight into the likelihood of finding optimal solutions across multiple iterations.

- **Summed Near-Optimal Probabilities**: We also compute the summed probabilities of obtaining near-optimal solutions, which are defined as solutions that are within a certain threshold of the optimal solution. This metric helps us understand the performance of our approach in finding solutions that are close to optimal.

- **Expectation Values**: For larger graphs, since full statevector simulation is not possible, we use the QTensor tensor network simulator and compute the expectation values of the cost Hamiltonians for both the constrained and unconstrained formulations. This allows us to evaluate the performance of our approach in terms of the expected profit obtained from the solutions.

- **Approximation Ratio**: The approximation ratio is defined as the ratio of the expected value of the solution obtained by our approach to the expected value of the optimal solution. This metric provides a measure of how well our approach performs compared to the optimal solution. The approximation ratio is calculated as follows:

$$\text{Approximation Ratio} = \frac{\text{Expected Value of Solution}}{\text{Expected Value of Optimal Solution}} \tag{7.16}$$

7.2.7. A note on Approximation Ratios

In classical computing, we say that an algorithm ALG for a maximization problem is an α-approximation algorithm (or that its approximation factor is α) if the following inequality holds for every input instance x:

$$ALG(x) \leq OPT(x) \leq \alpha \cdot ALG(x)$$

For a minimization problem:

$$OPT(x) \leq ALG(x) \leq \alpha \cdot OPT(x)$$

Note that in both cases the approximation factor α is a number greater than or equal to 1. In contrast to classical approximation algorithms [Vazirani, 2001, Hromkovič, 2013], in the QAOA literature a different notational convention is adopted [Farhi et al., 2014]. All problems as minimization problems (the standard form for optimization), the approximation ratio is defined as:

$$\text{Approximation Ratio} = \frac{\text{Expected Value of Optimal Solution}}{\text{Expected Value of Solution}} \leq 1 \tag{7.17}$$

Under this convention, the approximation ratio is always bounded above by 1, with values closer to 1 indicating better algorithm performance. This aligns with the standard practice in quantum optimization literature and provides a consistent metric across different problem types. However, this notation can be misleading when applied to penalty-based encodings of constrained problems. In such cases, the approximation ratio may not accurately reflect the true performance of the algorithm, as discussed below.

Limitations of Approximation Ratio for Penalty-Based Encodings

While approximation ratio serves as a reliable metric for unconstrained problems encoded naturally as QUBOs or HUBOs (where cost values maintain proper solution hierarchy), its effectiveness breaks down for penalty-based encodings of constrained problems. As discussed in Chapter 3, penalty-based approaches, while capable of encoding optimal solutions correctly, often fail to maintain proper cost hierarchies for near-optimal solutions.

This misalignment between solution quality and cost values may create an artificial inflation of the approximation ratio. Consider the following scenario: suppose a QAOA execution returns a infeasible solution with 50% probability, and a higher-quality feasible solution with 25% probability. Due to suboptimal penalty parameter settings, the higher-quality solution might be assigned a worse cost value than the infeasible solution. Consequently, the approximation ratio calculation would favor the poorer solution, producing misleadingly optimistic performance metrics. This issue is further compounded when working

with larger graphs using expectation value simulators like QTensor. Unlike full statevector simulation, expectation value calculations do not provide information about individual solution probabilities, making it impossible to filter out infeasible solutions. Without the ability to distinguish between feasible and infeasible solutions in the expectation value, the approximation ratio becomes particularly unreliable as a performance metric, as it may incorporate contributions from infeasible solutions that artificially improve the ratio.

This phenomenon highlights a fundamental limitation: approximation ratio becomes an unreliable metric for evaluating algorithm performance on penalty-encoded problems, as it fails to capture the true hierarchy of solution quality. This limitation is particularly significant when the penalty parameters are not perfectly tuned and near-optimal solutions are of interest.

7.3. Chapter Summary

This chapter detailed our experimental methodology and evaluation metrics for assessing the SCOOP framework. We presented comprehensive setup details for both quadratic and higher-order problem formulations.

We described the QAOA implementation specifics, including cost Hamiltonians for various problem pairs, mixer Hamiltonian, and parameter initialization. The evaluation was conducted on two complementary platforms: Xanadu PennyLane for full statevector simulation of smaller instances (up to 14 nodes) and Argonne QTensor for tensor network simulation of larger instances (up to 70 nodes).

We described the following metrics to evaluate the quality of solutions produced by penalty-based methods and our approach using the SCOOP framework: individual probability distributions, summed probabilities of optimal and near-optimal solutions, and approximation ratios.

We also discussed important limitations of approximation ratios in penalty-based encodings. This is especially relevant when working with expectation value calculations on larger instances using tensor network simulation where individual solution probabilities cannot be determined.

The experimental methodology and metrics described in this chapter are used to evaluate our SCOOP framework in the following chapter, where we present detailed results across various problem classes and instance sizes.

Chapter 8

Experimental Evaluation and Discussion on Quantum Simulators

Contents

Chapter Contributions

C10: *Comprehensive experimental evaluation of QUBO-based SCOOP problems:*

- C10.1: *Full statevector simulation (≤ 14 nodes) on Xanadu PennyLane*
- C10.2: *Tensor network simulation (15-70 nodes) on Argonne QTensor*
- C10.3: *Comparison with penalty-based approaches across multiple metrics*

C11: Evaluation of HUBO-based SCOOP problems through full statevector simulation on small 3-regular graph instances, demonstrating feasibility of higher-order optimization problems

This chapter presents the results of our experimental evaluation of solving SCOOP problems using QAOA, focusing on QUBO and HUBO formulations presented in the previous

chapters. We utilize PennyLane's standard analytical simulator for smaller problem instances (≤ 14 nodes) and the Argonne QTensor Tensor network simulator for larger graphs (15-70 nodes). Our analysis includes a range of metrics to assess performance, including probabilities, summed probabilities, near-optimal solutions, and approximation ratios.

8.1. Experimental Results: QUBO Problems

8.1.1. Probabilities

In our analysis, we examine the individual probabilities of obtaining feasible solutions, comparing the profit and constrained formulations using the standard (i.e., vanilla) QAOA implementation.

Fig. 8.1 shows a probability distribution for MaxPC and MinVC with $p \in \{1, 2, 3\}$ layers for the five node graph shown in the inset of Fig. 8.1, run on PennyLane's standard analytical simulator (`default.qubit`). The classical optimizer used is the Root Mean Squared Propagation (RMSProp). The quantum-classical loop is run 200 times to obtain the results. The bar in the burgundy color in Fig. 8.1 shows the result of post-processing profit cover results to vertex cover using Alg. 1 in Chapter 3. As the number of layers increases, we can see a high probability (greater than 0.9) of obtaining the optimal result. Fig. 8.2 demonstrates analogous probability distributions for the dominating set problem, which requires a higher-order Hamiltonian formulation, using a representative five-node graph as our test instance.

8.1.2. Summed Probabilities

Summed probabilities (or the winning probabilities) refers to the summation of probabilities associated with obtaining optimal solutions in a given problem space.

$$SP_{\text{OPT}} = \sum_{k \in sol_{\text{opt}}} P(X = k) \tag{8.1}$$

We adopt the summed probability metric from [Saleem, 2020]. They introduced it to capture optimal and near-optimal solutions to a constrained optimization problem better than, say, an approximation ratio. In Equation 8.1, sol_{opt} is the set of all optimal solutions, and $P(X = i)$ is the probability of obtaining solution i. The vertex cover, independent set, and clique problems have multiple optimal solutions in both their constrained and profit formulations. Notably, the profit variants of these problems tend to yield a greater number of optimal solutions compared to their constrained counterparts. Solving the profit formulation leads to solutions sampled with a higher probability because every MinVC solution of size k corresponds to a MaxPC solution with profit $p = |E| - k$, and additionally,

certain infeasible vertex covers can achieve the same profit value while still being valid profit covers.

The increased density of optimal solutions for profit problems, coupled with the probabilistic nature of quantum computing outputs enhances the likelihood of the QAOA converging towards an optimal outcome. Fig. 8.3 shows the summed probabilities over eight layers for constrained and profit versions of vertex cover, independent set and clique for varying edge probabilities averaged over ten graphs.

8.1.3. Near-optimal analysis

Our analysis extends beyond focusing on the optimal solutions. In many practical scenarios, a range of near-optimal or sub-optimal solutions is desirable. For example, in financial applications, such as portfolio optimization, a solution with slightly higher risk might be preferred because it also comes with the potential for greater returns. Therefore, we examine the probability distribution across this wider spectrum of solutions.

Eqns. 8.2 and 8.3 represent summed probabilities of optimal solutions, second-, third-best near-optimal solutions.

$$SP_{\text{OPT}-1} = SP_{\text{OPT}} + \sum_{k \in sol_{\text{opt}-1}} P(X = k) \tag{8.2}$$

$$SP_{\text{OPT}-2} = SP_{\text{OPT}} + SP_{\text{OPT}-1} + \sum_{k \in sol_{\text{opt}-2}} P(X = k) \tag{8.3}$$

Here $sol_{\text{opt}-1}$ and $sol_{\text{opt}-2}$ refer to the sets of the second and the third best solutions, respectively.

8.1.3.1. Near-optimal analysis on Vertex Covers

Fig. 8.7 shows summed probabilities of optimal and near-optimal solutions for MinVC and MaxPC for varying edge densities averaged over ten graphs. The leftmost column shows the summed probability of obtaining optimal solutions, the middle column shows the summed probability of obtaining optimal solutions and second best solutions. The rightmost column shows the summed probability of obtaining optimal solutions, second best solutions, and third best solutions. In the case of profit problems, near-optimal solutions are those with the second and the third best profit. For example, in the case of MaxPC the graph in Fig. 3.3a has an optimal profit of 5. A second-best solution for this graph would be $PC = \{1, 3, 5\}$ with a profit of 4. A third best solution for this graph would be $PC = \{3, 5\}$ with a profit of 3.

For constrained problems, such as the MinVC, we choose the second-best solutions from the set of feasible solutions with a vertex cover size of $|VC|_{\text{OPT}} + 1$. Consider the same

graph from Fig. 3.3a, this has an optimal vertex cover of size 4. A second-best solution for this graph would be VC $= \{1, 2, 3, 4, 5\}$ with a size of 5. A third best solution for this graph would be $PC = \{0, 1, 2, 3, 4, 5\}$ with a size of 6. Note there are also examples of the second and the third best profit covers. Near-optimal solutions for vertex cover can only be obtained by adding vertices to the minimum vertex cover, however, near-optimal solutions to profit cover can be obtained by adding or removing vertices to the maximum profit cover.

Given that constrained problems have a penalty-term formulation, it is worth noting that near-optimal solutions that have the second lowest or the third lowest cost (as calculated using Eqn. 7.2) may not correspond to feasible solutions depending on the penalties chosen (cf. Section 3.2). Therefore, near-optimal solutions are chosen by summing the probabilities for all feasible solutions with size $|VC|_{\text{OPT}} + 1$ and $|VC|_{\text{OPT}} + 2$.

8.1.3.2. Near-optimal analysis on Independent Sets

Fig. 8.4 is analogous to Fig. 8.7 (which shows summed optimal and near-optimal probability results for vertex cover), but applied to the MAXIS and MAXPI problems for $n = 8$. Fig. 8.5 shows results of summed optimal and near-optimal solutions for $n = 14$. The results are presented across a spectrum of graph densities, ranging from sparse ($d = 0.1$) to dense ($d = 0.8$) edge probabilities, allowing us to evaluate the algorithm's performance under varying conditions.

8.1.3.3. Near-optimal analysis on 3-Regular Graphs

Fig. 8.6 shows the near-optimality analysis for 3-regular graphs for all six problems; MINVC and MAXPC in the first row, MAXIS and MAXPI in the middle row, and, MAXCL and PCL in the bottom row.

8.1.4. Expectation value simulation of large problems with QTensor

Fig. 8.8 shows the cost function evaluations for a random 3-regular graph for $n \in \{30, 40, 50, 70\}$. While expectation values indicate the closeness to the optimal solution, with profit cover, we have the advantage that the expectation value corresponds to the calculated profit, with the minimum expectation value equal to the maximum profit. For example, consider the results for layer 3 for MAXPC. The cost value is close to -15, suggesting a high likelihood of a profit cover with a profit of 15, and therefore, a high likelihood of a vertex cover of size at most $|E| - 15$.

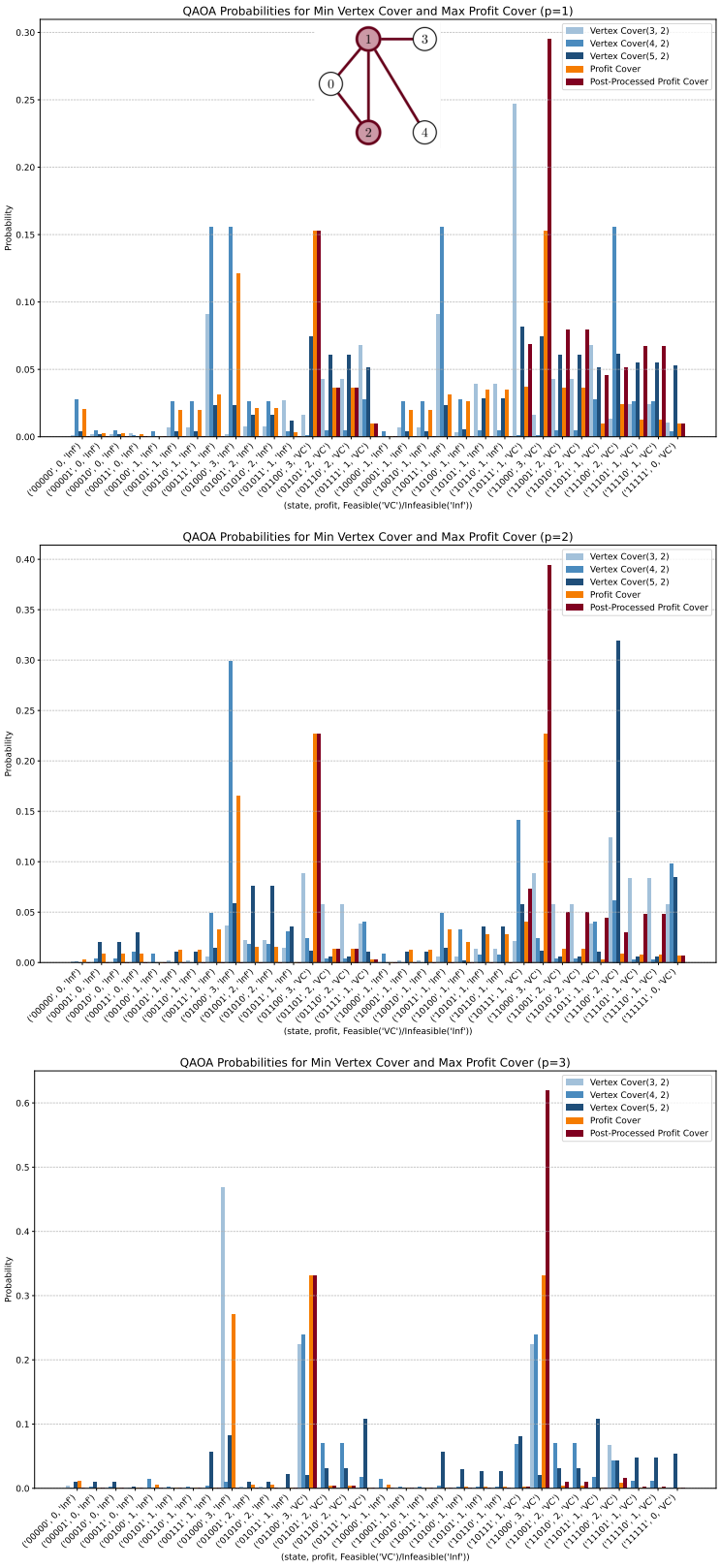


Figure 8.1.|: Probabilities for a sample graph (shown in the inset of topmost figure or graph) containing 5 vertices with $p \in \{1, 2, 3\}$ layers. MinVC is run for three different penalty parameters and is shown in blue. For example, Vertex Cover (3, 2) refers to MinVC penalties $A = 3, B = 2$. Results of MaxPC are shown in orange. Post-processed results of MaxPC are shown in burgundy.

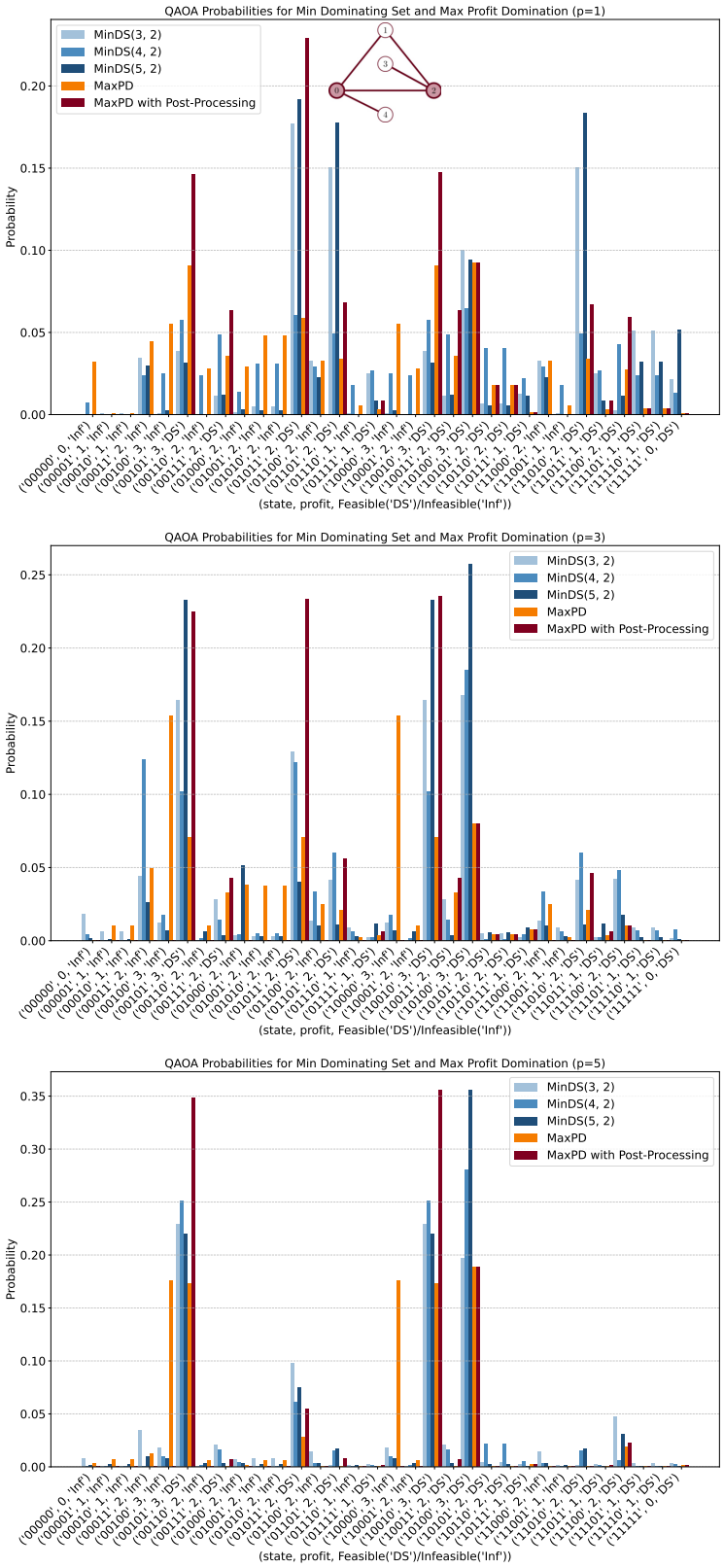


Figure 8.2.|: Probabilities for a sample graph (shown in inset of leftmost sub-figure) containing 5 vertices with $p \in \{1, 3, 5\}$ layers. MinDS is run for three different penalty parameters and is shown in blue. For example, Dominating Set(3, 2) refers to MinDS penalties $A = 3, B = 2$. Results of MaxPD are shown in orange. Post-processed results of MaxPD are shown in burgundy.

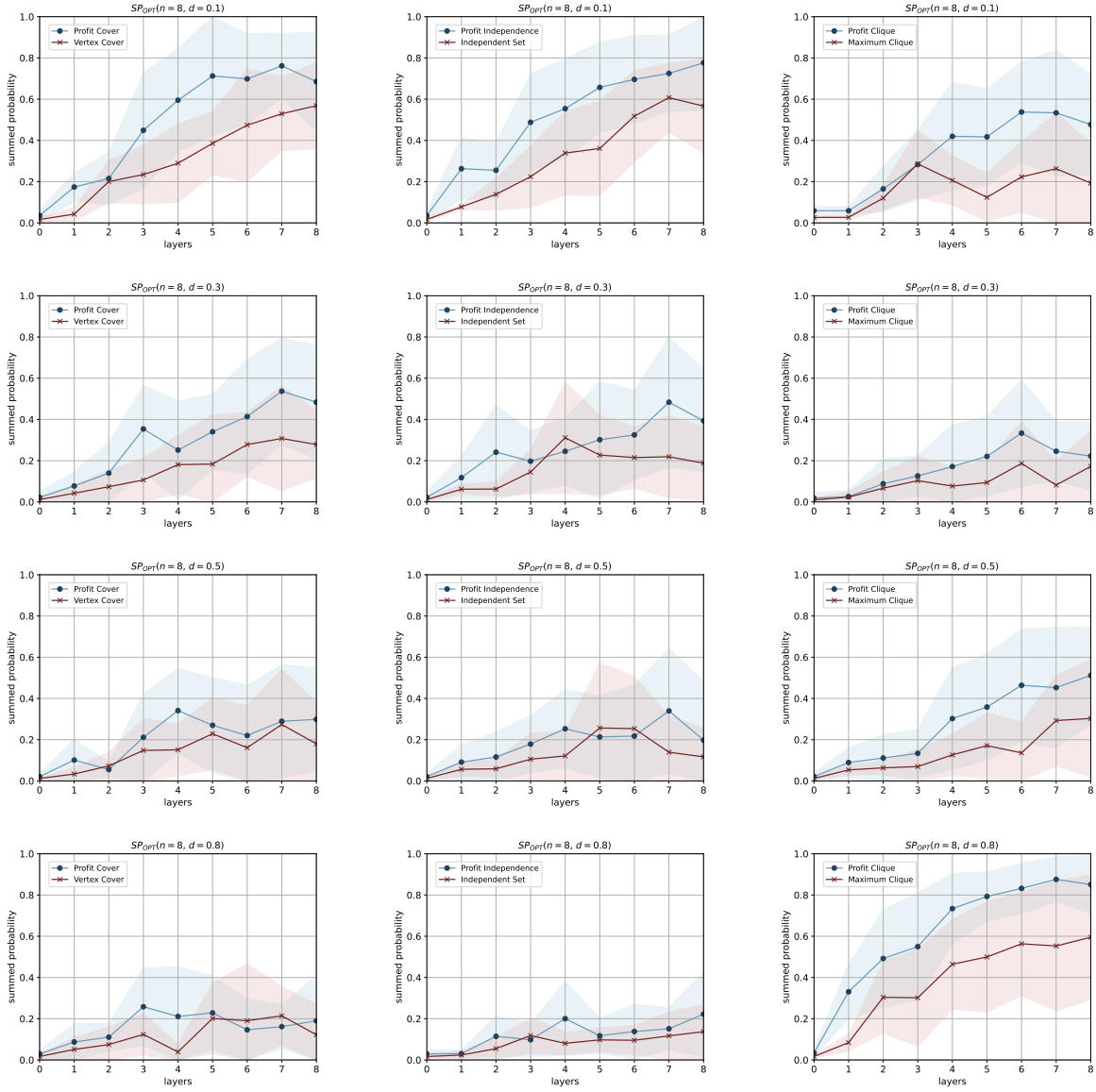


Figure 8.3.|: **Average summed probabilities of optimal solutions over ten graphs obtained over eight layers.** The figures in the leftmost column compare MinVC and MaxPC, The figures in the middle column show MaxIS and MaxPI, The figures in the rightmost column compare MaxCl and MaxPCl. The rows indicate different edge densities of $d \in \{0.1, 0.3, 0.5, 0.8\}$. Each figure features two line graphs that represent the comparison between the profit version (blue) and the constrained version (red). Data points correspond to the average value for each layer, with shaded regions indicating one standard deviation away from the mean. The blue line graphs (profit formulation) are consistently above the red line graphs (constrained formulation), indicating that the profit versions have a higher probability of obtaining optimal solutions.

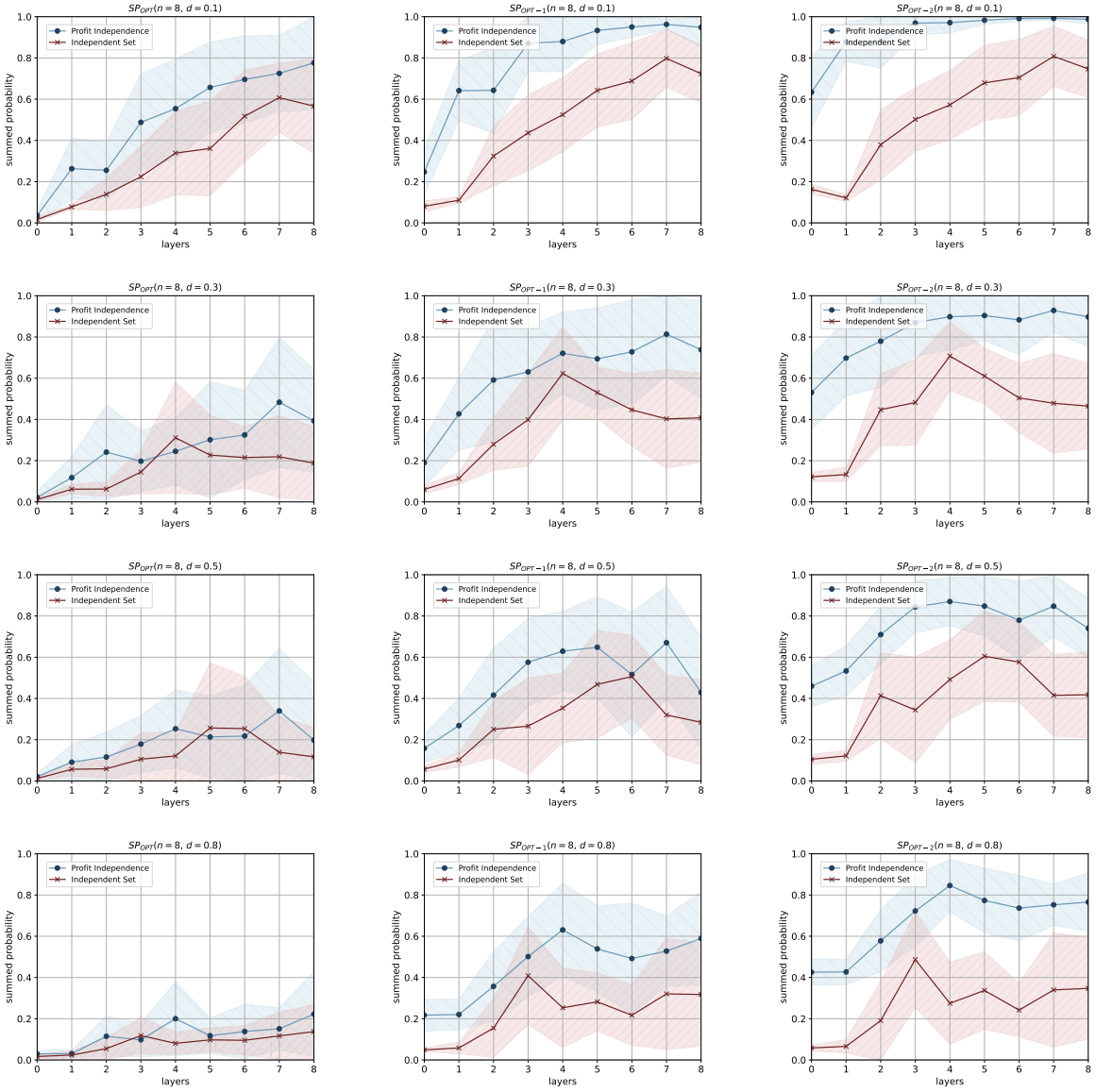


Figure 8.4.|: Probabilities of optimal and near-optimal solutions obtained over eight layers for MaxIS *and* MaxPI. The first column shows the summed probability of all the optimal solutions averaged over ten graphs with $n = 8$. The second column depicts the summed probability of obtaining the optimal solution and the second best solution. The third column indicates the summed probability of the optimal, second best and the third best solutions. Each row indicates a different edge density. Each figure features two line graphs that represent the comparison between the profit version (blue) and the constrained version (red). The results show that the profit formulation consistently has a higher probability of obtaining optimal and near-optimal solutions compared to the constrained formulation.

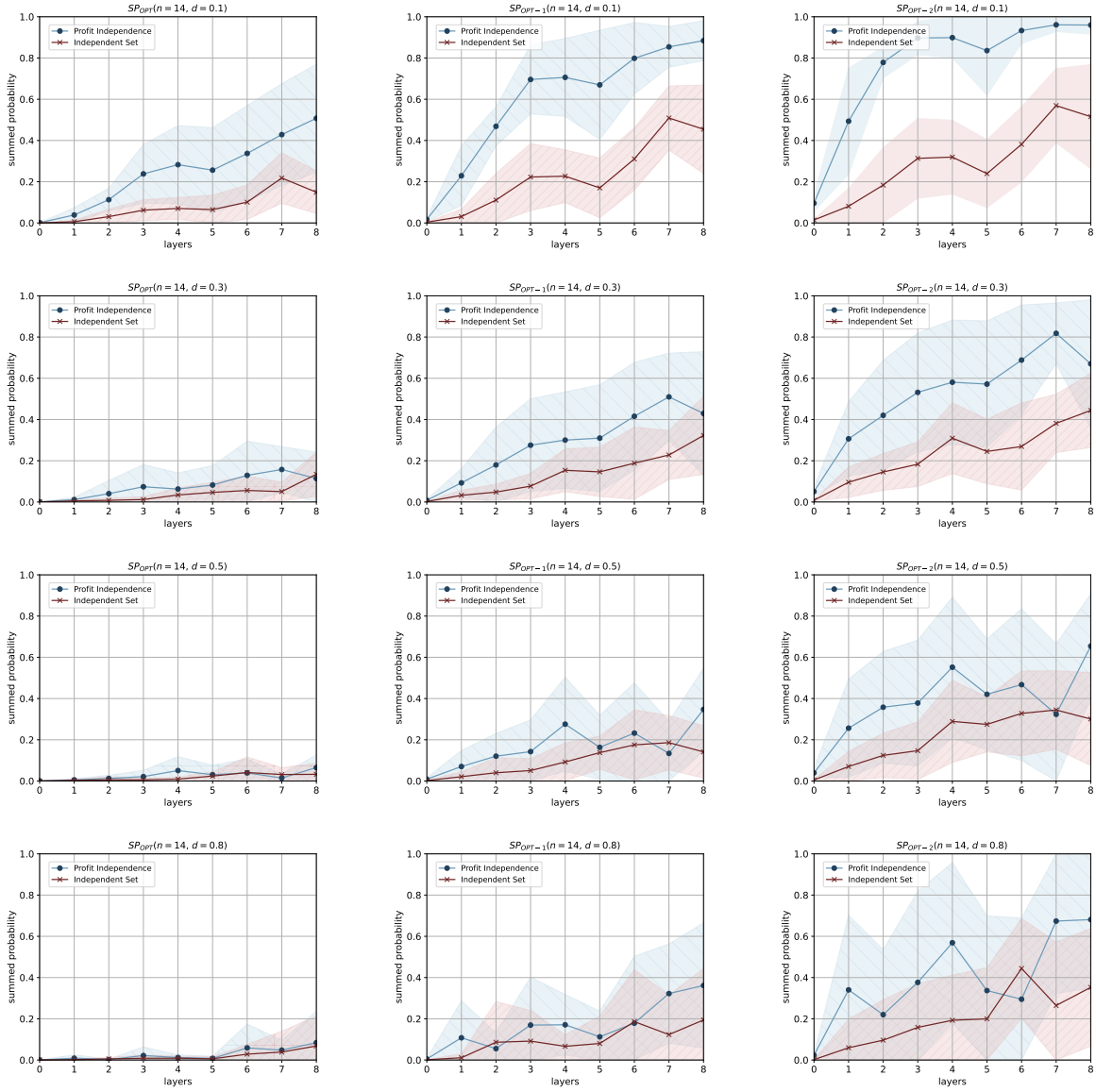


Figure 8.5.|: **Summed optimal and near-optimal probabilities obtained over eight layers averaged over ten graphs for MaxPI and MaxIS for** $n = 14$**.**The figures presented here can be interpreted in a manner similar to Fig. 8.7. Each row represents a different edge density of the graphs. The figures in the left-most column display the summed optimal probabilities, while those in the middle and right columns show the near-optimal probabilities. Each figure features two line graphs that represent the comparison between the profit version (blue) and the constrained version (red), and shows that QAOA on MaxPI results in higher probabilities of obtaining optimal and near-optimal solutions.

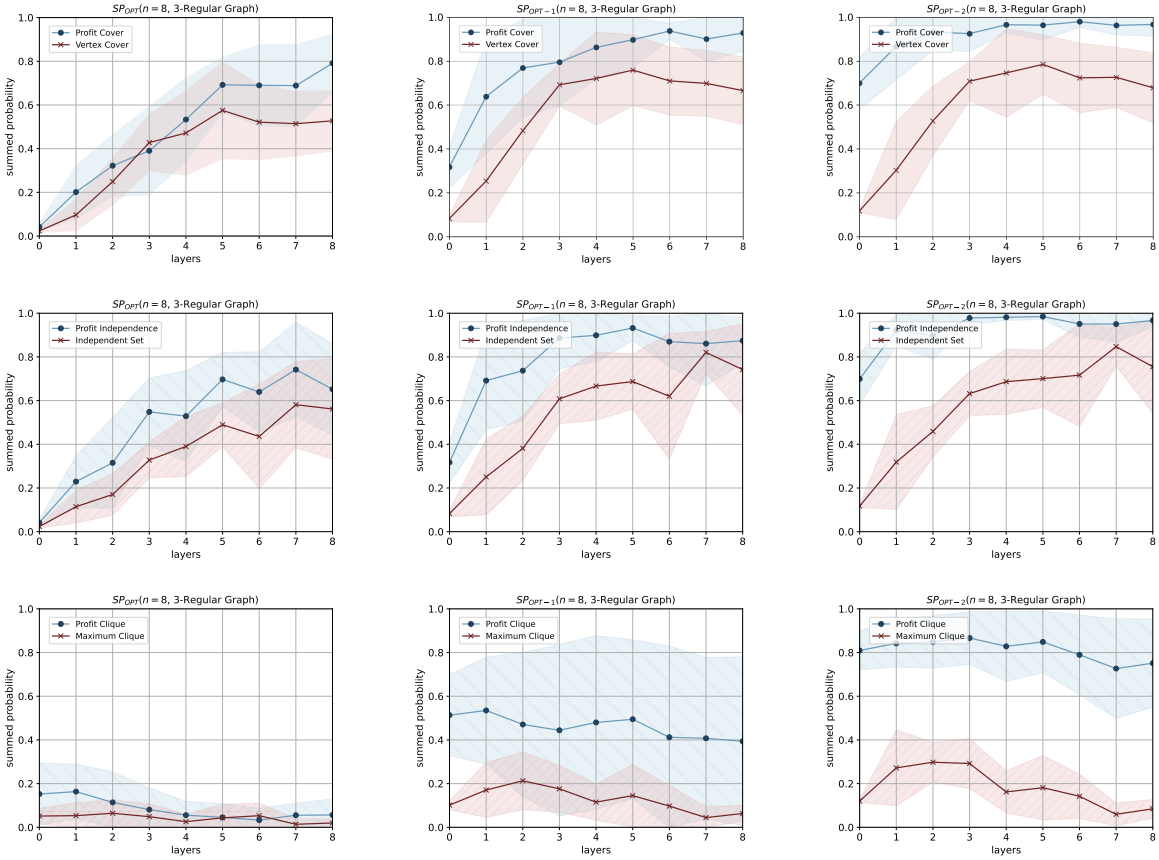


Figure 8.6.|: **Average summed probabilities of optimal solutions over ten 3-regular graphs obtained over eight layers.** The figures in the topmost row compare MINVC and MAXPC, The figures in the middle row compare MAXIS and MAXPC, the graphs or figures in the bottom row compare MAXCL and MAXPCL. The rows indicate the extent of near-optimality. Each figure features two line graphs that represent the comparison between the profit version (blue) and the constrained version (red). Data points correspond to the average value for each layer, with shaded regions indicating one standard deviation away from the mean. For three-regular graphs, we see consistent improvement in probabilities for profit formulations over their constrained versions.

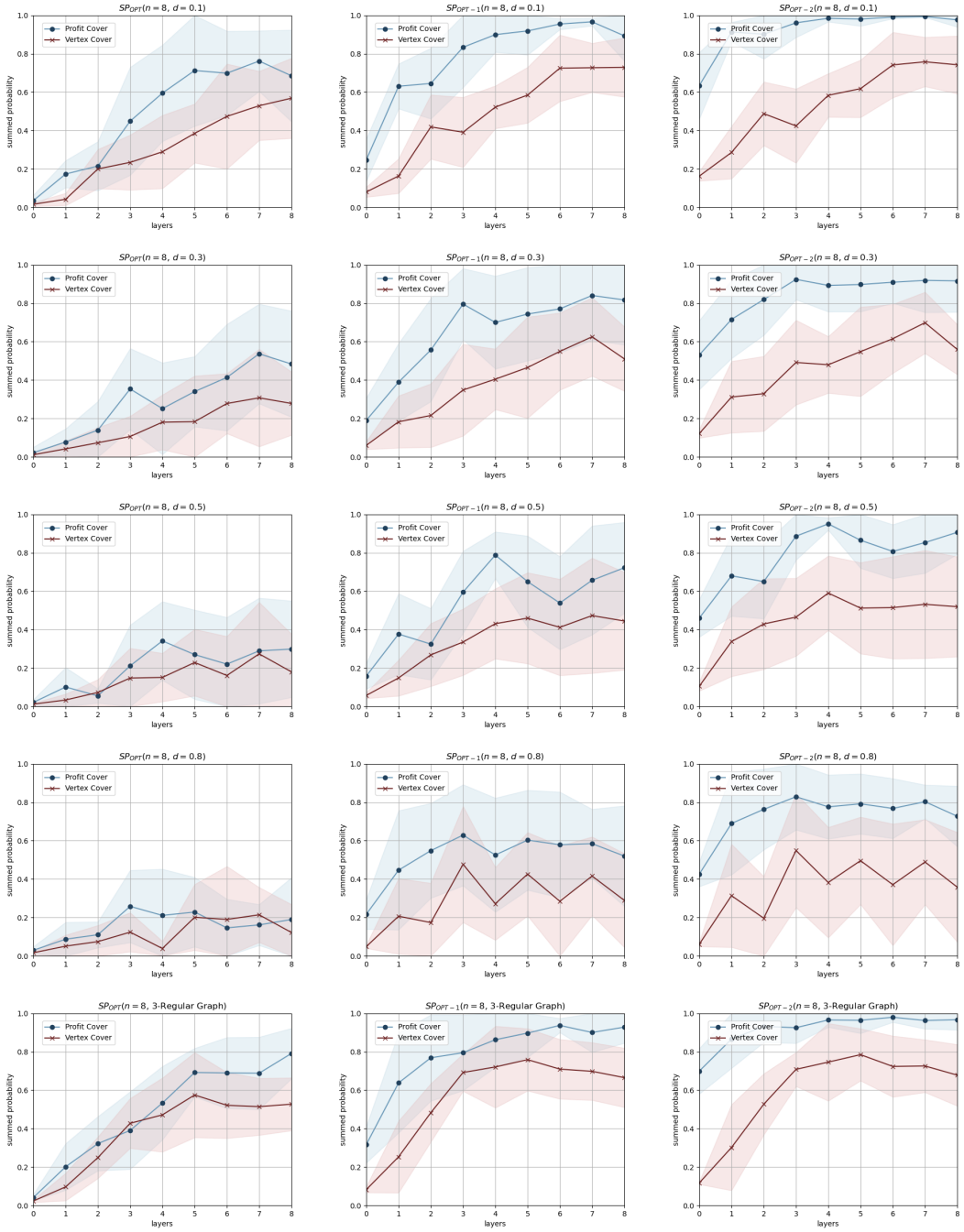


Figure 8.7.|: Probabilities of optimal and near-optimal solutions obtained over eight layers for MinVC *and* MaxPC. The figures in the first column show the summed probability of all the optimal solutions averaged over ten graphs with $n = 8$. The figures in the second column depict the summed probability of obtaining the optimal solution and the second best solution. The figures in the third column indicates the summed probability of the optimal, second best and the third best solutions. Each row indicates a different edge density with the last row showing the results for 3-regular graphs.

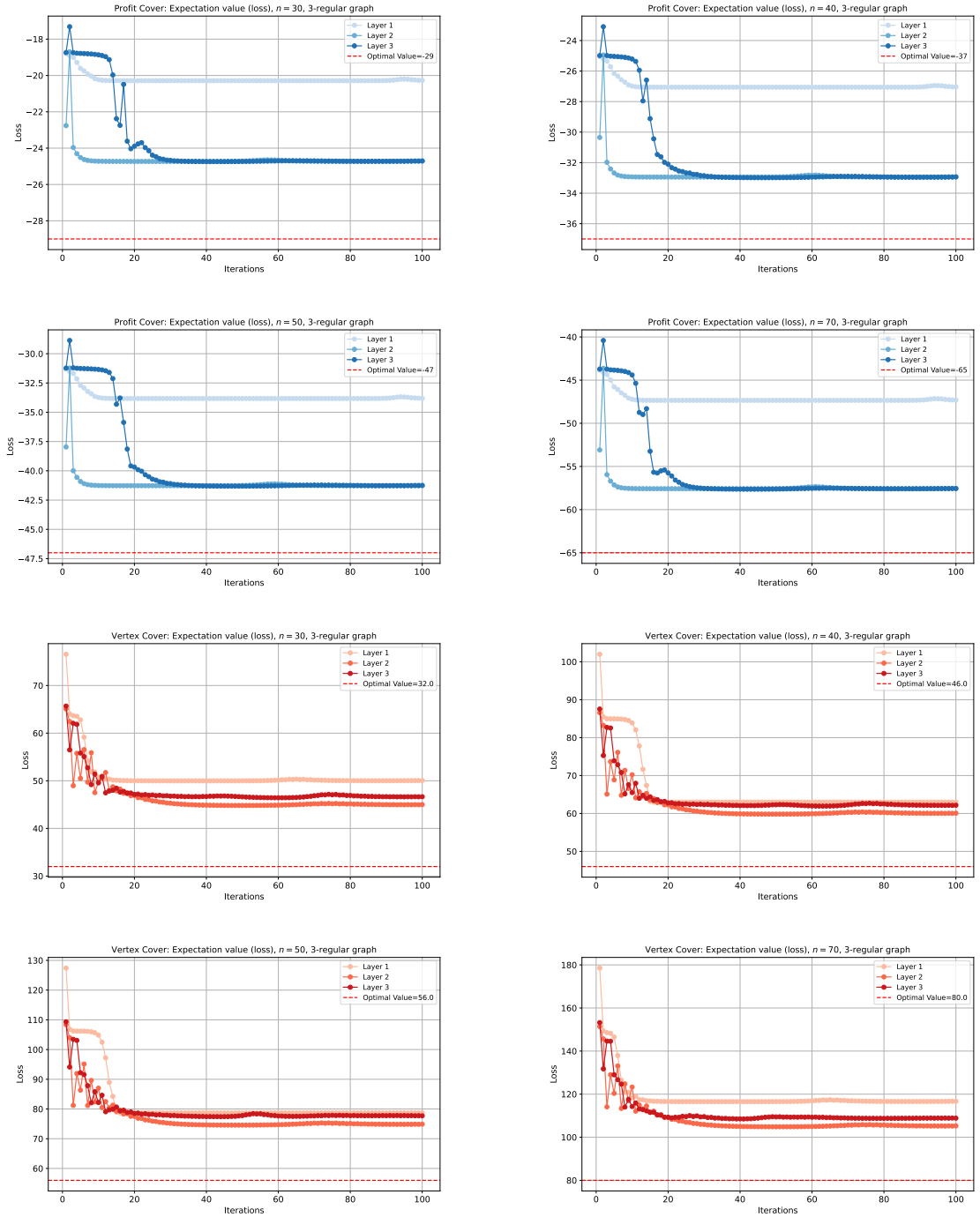


Figure 8.8.|: Cost optimization over $n \in \{30, 40, 50, 70\}$, for 3-regular graphs on QTensor for MaxPC and MinVC.

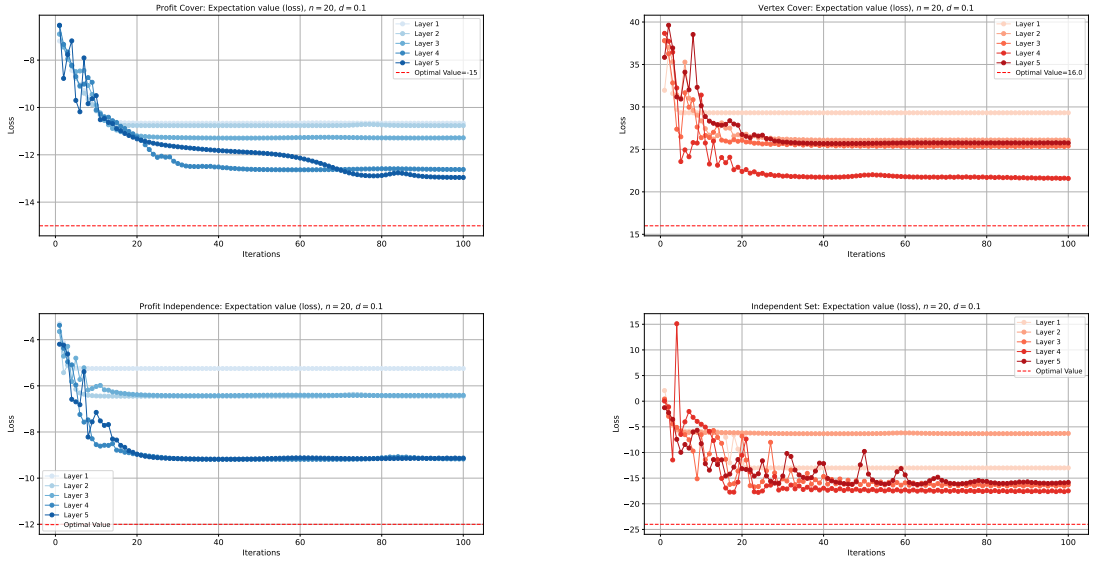


Figure 8.9.|: Cost optimization for $n = 20$ for a sparse (edge probability of 0.1) graph, on a graph over five layers. The figures in the top row present the expectation value minimized over 100 iterations for MaxPC and MinVC, while The figures in the bottom row show the results for MaxPI and MaxIS. Note that although MaxPC, MaxPI, and MaxIS are maximization problems, the results presented correspond to the minimization versions of these problems.

8.2. Approximation Ratios

Fig. 8.10 shows the approximation ratios for solving MAXPC for $n = 7$ and $n = 8$ over eight layers and ten graphs calculated using PennyLane. The approximation ratio is calculated using:

$$r = \frac{|\text{Expectation value obtained} + \Delta_{\text{PC}}|}{\text{Optimal Profit}} \tag{8.4}$$

We show the results starting with layer 0, which has no cost or mixer unitaries, and shows the expectation value of an equal superposition state added to the offset Δ_{PC} (note the addition of Δ_{PC} in Equation 8.4). Adding the offset provides the exact expectation value of the profit obtained.

Fig. 8.11 shows the average MAXPC performance of 10 3-regular graphs each for $n = \{8, 10, 12, 14\}$ on PennyLane and for $n = \{20, 30, 40, 70\}$ calculated on QTensor.

8.3. Discussion: QUBO Problems

Our findings provide valuable insights into formulating and solving constrained optimization problems with vanilla QAOA.

Solving the cost Hamiltonian for constrained problems with QAOA requires fine-tuning of penalty parameters in addition to optimizing the variational parameters, to be able to maneuver the cost function landscape successfully. For profit formulations, since every state vector represents a feasible solution, one only needs to optimize the circuit's variational parameters without the need to set appropriate penalties.

Fig. 8.1 shows one instance of probability distribution with $p \in \{1, 2, 3\}$ layers for the five node graph shown in the inset of the graph, run on PennyLane's standard simulator (`default.qubit`). Since there are three maximum profit covers, we see a higher probability for the states 01000, 01100, and 11000 than all other states. While 01100 and 11000 are minimum vertex covers, the state 01000 (i.e., VC $= \{1\}$) is not a vertex cover but can be converted into a minimum vertex cover (i.e., VC $= \{1, 0\}$ or VC $= \{1, 2\}$) of size 2 (Alg. 1). At $p = 3$, the vertex cover formulation demonstrates improved outcomes. Note that the summed probabilities for the profit covers (43.1%) of the largest size are higher than the summed probabilities for the minimum vertex covers (35.5%).

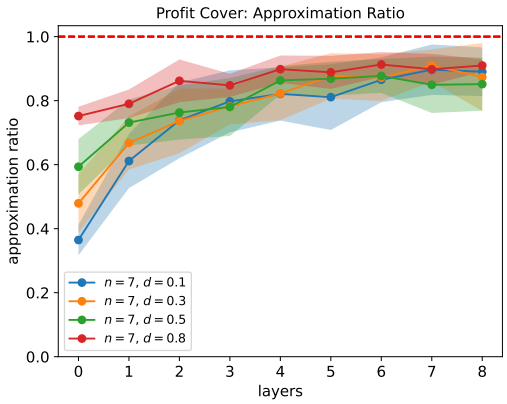
For the same problem instance, the data points in burgundy in Fig. 8.1 shows the results of post-processing profit cover using Alg. 1. Recall that every bit string for profit cover is feasible, whereas only a subset of the bit strings are feasible for vertex cover. Therefore, we run Alg. 1 on every bit string to convert each of those into feasible vertex covers and add up the probabilities.

A key finding from our experimental results is that profit formulations consistently achieve higher summed probabilities of obtaining optimal and near-optimal solutions compared to their constrained counterparts. This advantage in sampling efficiency demonstrates the effectiveness of our approach in leveraging the unconstrained nature of profit formulations. From Fig. 8.3, one can see that for optimal constrained problem solutions to catch up to the same summed probability value as profit cover, on average, it takes additional QAOA layers. Fig. 8.7 shows the summed probabilities for near-optimal solutions. The near-optimal analysis shows a significant increase in the probability values for the profit formulations as compared to the constrained formulation. Due to the unconstrained nature of the profit formulations, many infeasible solutions are also candidate near-optimal solutions as compared to the constrained formulation. Note that while solutions may be infeasible, for profit versions, they can be easily converted to a feasible solution with no change in profit using classical post-processing. While the same classical post-processing routine can be used for the constrained formulations, it does not hold the same value as the constrained formulations promote feasible solutions over infeasible solutions.

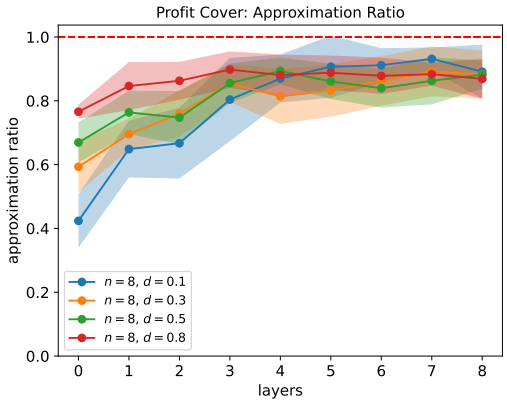
We observe a dip in the summed probabilities for both profit and constrained versions of vertex cover and independent set for dense graphs. A few dips in the summed probabilities over layers can be attributed to the classical optimizer becoming trapped at local minima. Further investigation into the cost function landscape of profit cover problems is warranted here.

We can see the same trend between results obtained in the summed probabilities, as depicted in Fig. 8.3, and the approximation ratios obtained (i.e., the increase up to layer 2 and the dip in solution quality between layers 3 and 5).

Fig. 3.2 and Fig. 8.1 show the effect of using different penalties for the constrained formulation of the MinVC problem. In both figures, it is clear that not one penalty value is better than another for finding optimal solutions using QAOA. The performance of penalties is also not consistent over layers; for example, in the top left corner sub-figure of Fig. 3.2, the summed optimal probability of QAOA with penalty parameters $A = 6, B = 2$ does not perform well on average in the initial layers, but improves after layer 5. Additionally, the sub-figure on the bottom left corner shows that QAOA with penalty parameters of $A = 6, B = 2$ does not perform well for graphs with edge density of $d = 0.5$. Fig. 8.8 shows the cost function evaluations of MaxPC and MinVC for a single random 3-regular graph with $n \in \{30, 40, 50, 70\}$. Fig 8.9 shows MaxPC, MinVC, MaxPI and MaxIS results for a single sparse graph (edge density, $d = 0.1$). We observe fluctuations in the early parts of the optimization for all four problems due to the adaptive learning rate of RMSProp. The red dotted line represents the exact solution to the corresponding problems. While QAOA tends to get stuck in local minima, it approaches the exact solution more closely in MaxPC compared to MinVC, due to the presence of more optimal and near-optimal solutions in MaxPC than MinVC. We further remark that in the case of constrained optimization (MinVC) while the expectation value indicates how close the current solution is to the optimal one in terms of the objective, it does not give information about whether the constraints are satisfied. However, this is not the case for the unconstrained MaxPC. For example, the top left graph of Fig. 8.8 shows that the expectation value achieved with



(a) Approximation ratios for graphs with $n = 7$



(b) Approximation ratios for graphs with $n = 8$

Figure 8.10.|: Approximation ratios computed for MaxPC on PennyLane. The rate of change of approximation ratio is higher for sparse graphs compared to denser graphs. Denser graphs have more near-optimal solutions compared to sparse graphs, therefore, start off with a high approximation ratios but have a smaller rate of change of the approximation ratio.

MaxPC is in between -24 and -26, indicating that the optimal profit is at least 24 for this particular graph. Given a total of $|E|$ edges for the graph, we can conclude that a vertex cover of size at most $|VC| \leq |E| - 24$ exists for this graph. We demonstrate the scalability of our approach by comparing the approximation ratio achieved on MaxPC for $p = 1, 2, 3$ layers, which shows that our approach can achieve an approximation ratio greater than 0.8 for graphs of varying sizes, ranging from $n = 8$ to $n = 70$.

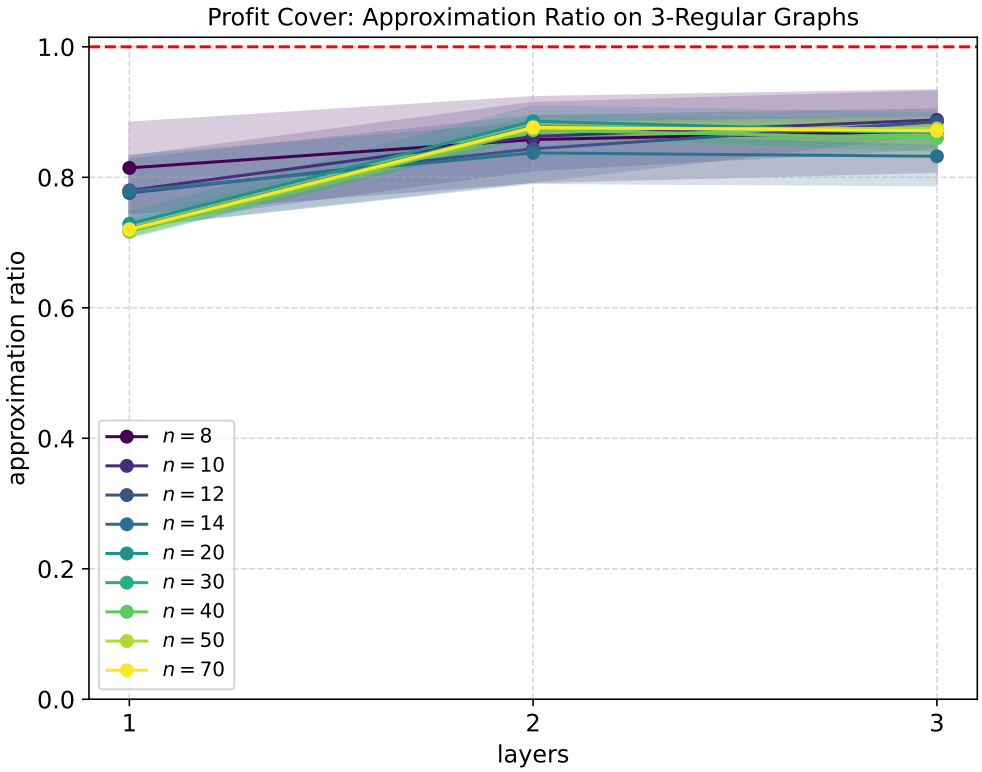


Figure 8.11.|: Approximation ratios computed for MaxPC on PennyLane for $n = \{8, 10, 12, 14\}$ and QTensor for $n = \{20, 30, 40, 50, 70\}$ showing scalability of our approach.

8.4. Experimental Results: HUBO Problems

8.4.1. Probabilities

In our analysis, we examine the individual probabilities of obtaining feasible solutions for the constrained problem, comparing Hamiltonians for both SCOOP twins using the standard QAOA implementation. Fig. 8.2 shows a probability distribution for MaxPD and MinDS with $p \in \{1, 3, 5\}$ layers for the five node graph shown in the inset of Fig. 8.2, run on PennyLane's standard analytical simulator (`default.qubit`). The classical optimizer used is the Root Mean Squared Propagation (RMSProp). The quantum-classical loop is run 400 times to obtain the results. The bar in the burgundy color in Fig. 8.2 shows the result of post-processing profit domination results to dominating set using Alg. 1. As the number of layers increases, we can see a high probability (greater than 0.8) of obtaining the optimal result.

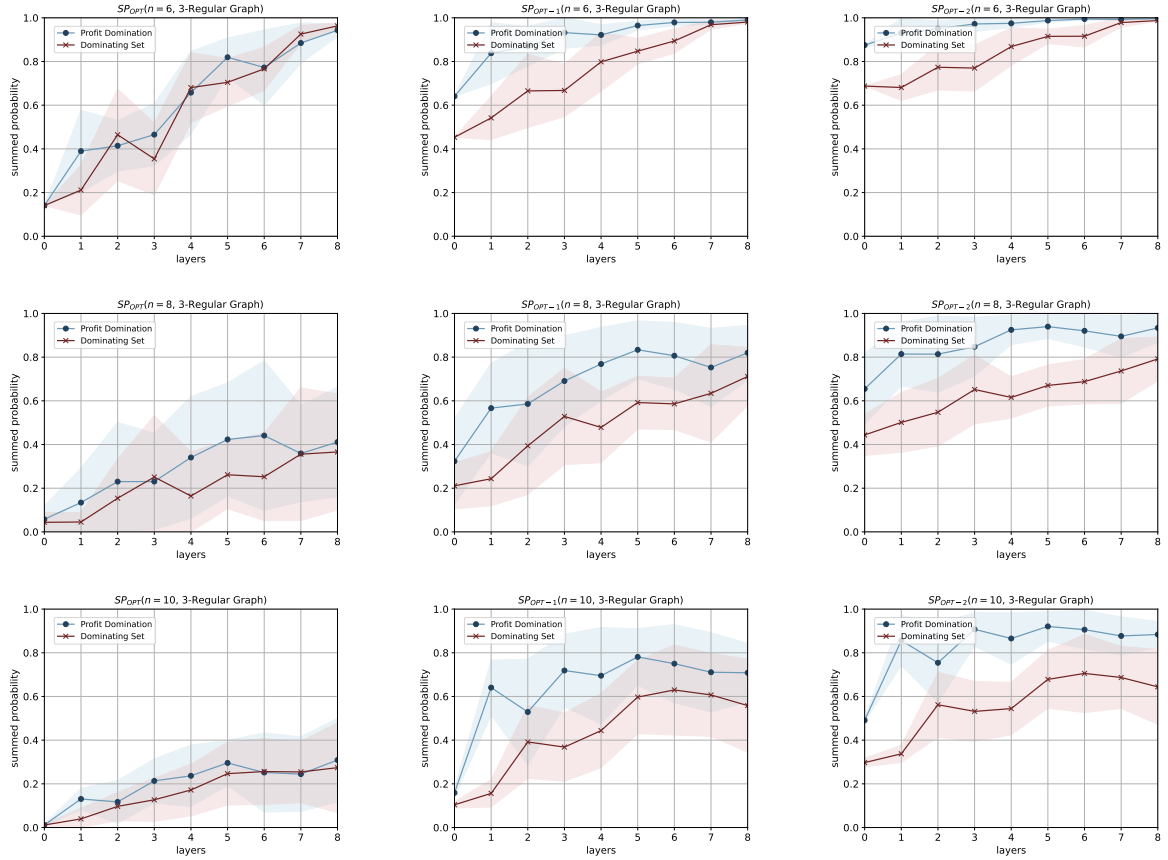


Figure 8.12.|: Probabilities of optimal & near-optimal solutions obtained over 8 layers for MINDS & MAXPD averaged over 10 3-regular graphs. The sub-figures in the first column show the summed probability of all the optimal solutions with $n \in \{6, 8, 10\}$. The sub-figures in the 2^{nd} column depict the summed probability of obtaining the optimal solution and the 2^{nd} best solution. The sub-figures in the 3^{rd} column indicates the summed probability of the optimal, 2^{nd} best and the 3^{rd} best solutions.

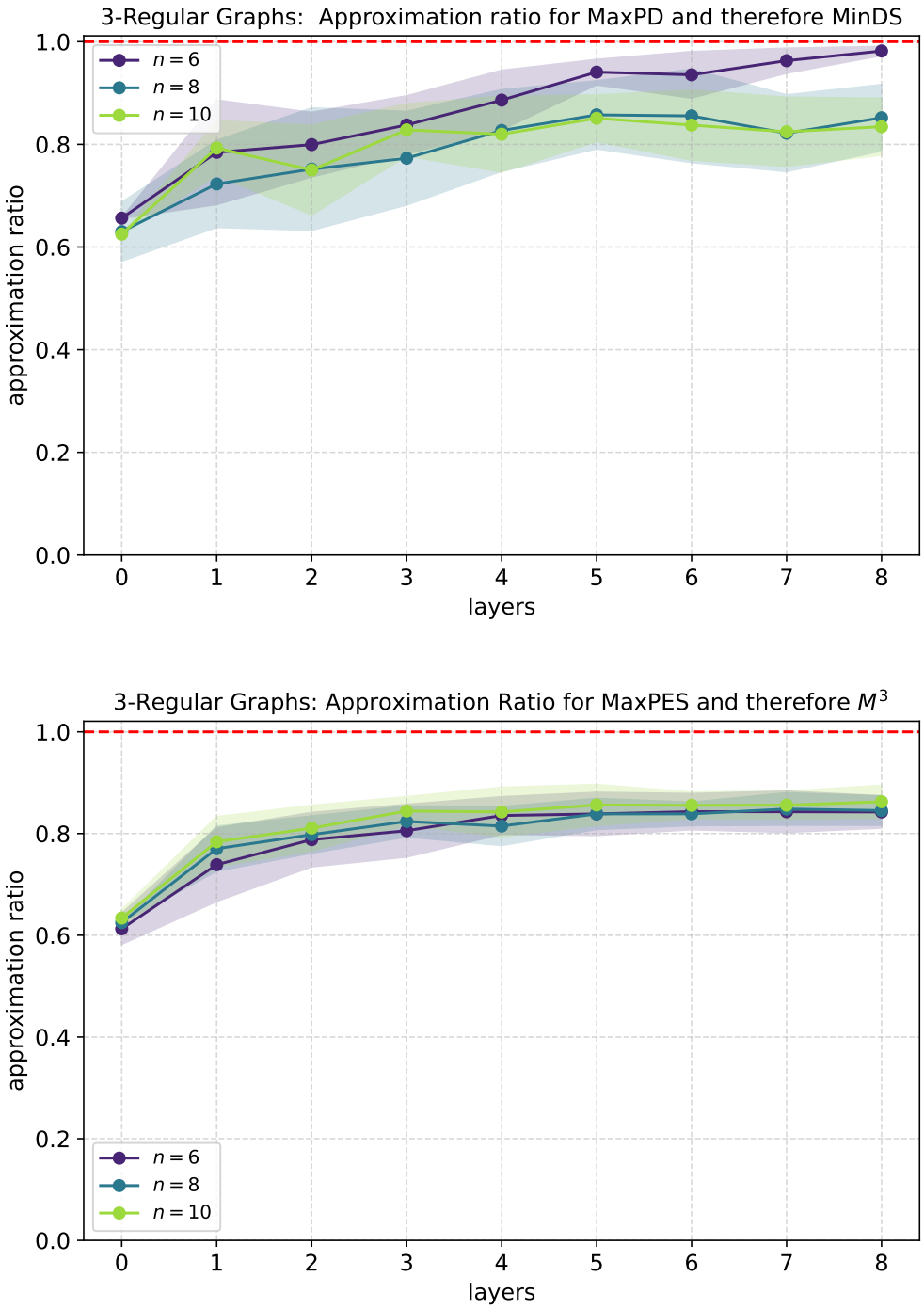


Figure 8.13.|: Approximation ratios computed for MaxPD and MaxPES on PennyLane for $n \in \{6, 8, 10\}$ averaged over 10 3-regular graphs. The profit of solutions to MinDS are guaranteed to be no worse than MaxPD (Alg. 1), and thus the approximation ratio is preserved for MinDS. Similarly, Algs. 7 and 8 imply that this approximation ratio also applies to M^3.

8.4.2. Summed Optimal and Near-Optimal Probabilities

Fig. 8.12 shows summed optimal and near-optimal probabilities for MaxPD and MinDS. For MinDS we present the results with penalties set to $A = 3, B = 2$. The results are averaged over 10 3-regular graphs each of sizes $n \in \{6, 8, 10\}$. For a 3-regular graph, when all three neighbors of a node interact to contribute jointly to the cost, it results in terms involving four variables (the node plus its three neighbors), leading to a polynomial with a maximum degree of 4. Thus, in this case, the QAOA operates on a cost function of degree 4, rather than the more typical degree 2 in QUBOs.

8.4.3. Approximation Ratios

Fig. 8.13 shows the approximation ratio obtained for MaxPD and MaxPES for $n \in \{6, 8, 10\}$ on 3-regular graphs. The derived approximation ratio applies not only to the SCOOP variants but also to the constrained versions of the problem. Specifically, Theorem 6 and Theorem 7 guarantee that the approximation ratio remains valid under the imposed constraints of MinDS and M^3.

8.5. Discussion: HUBO Problems

From Figs. 8.12 and 8.13, we observe that our method achieves an average probability of $10\% - 40\%$ of sampling the best solution at $p = 1$ for 3-regular graphs with up to 10 nodes. Our approach yields over 60% probability of sampling near-optimal solutions at $p = 1$ for $n \in \{6, 8, 10\}$—a probability that only increases with additional layers or broader solution sets. Due to the direct relationship with its constrained SCOOP twin MinDS, post-processing these solutions will only make the probability better (but not worse, see Theorem 6).

In the past, for MinDS, comprehensive experiments have been done only on quantum annealing hardware by Dinneen et al. in 2017 [Dinneen and Hua, 2017a]. M^3, described in Lucas (2014) [Lucas, 2014], has not yet been tested on QA or gate-based hardware.

Chapter 9

Utility-scale experiments of SCOOP on IBM Quantum Hardware

Contents

Chapter Contributions

C12: Evaluation of SCOOP framework on IBM quantum hardware using Qiskit 2.0, analyzing hardware considerations and experimental results on IBM *Fez* processor using MinVC and MaxPC problems as test cases.

In this chapter, we present our results from utility-scale experiments of the SCOOP framework on IBM Quantum hardware. We introduce the experimental setup, including the hardware used, the optimization techniques applied, and the specific combinatorial optimization problems selected for evaluation: MinVC and its SCOOP twin MaxPC. The experiments are conducted on IBM's superconducting quantum hardware using the Heron R2 processor, which is a state-of-the-art quantum processor provided by IBM Quantum [Gambetta, 2022]. We investigate circuit optimization techniques available in Qiskit and present the results obtained from running constrained combinatorial optimization problems and their SCOOP counterparts on IBM quantum hardware instances. Our analysis includes circuit metrics such as two-qubit gate counts and circuit depths, and discusses their implications for solution quality and hardware efficiency.

9.1. Quantum Software: Qiskit 2.0

Qiskit 2.0[1] represents a significant evolution in quantum software development, focusing on performance and scalability. This version introduces several key improvements that are useful in terms of performance and scalability. The transition to Qiskit 2.0 brings substantial architectural enhancements through a new C-language API for interfacing with quantum observables and extensive Rust-based implementations for core components. Rust enables significant performance improvements, including 2x speedup in circuit construction and 20% faster transpilation using Benchpress [Nation et al., 2025]. Rust also facilitates a new C-language API, critical for high-performance computing integration and hardware-level optimizations. Qiskit 2.0 adopts Semantic Versioning (<major>.<minor>.<patch>), with an 18-month support cycle. These improvements, particularly the enhanced circuit optimization capabilities and error mitigation features, are crucial for large scale experimental evaluation of the SCOOP framework on IBM's quantum hardware. The new architecture enables more efficient handling of quantum circuits, which is essential for scaling to larger problem instances.

9.2. IBM Quantum Hardware

The era of quantum utility has emerged, where quantum computers are becoming practical tools for scientific discovery [Kim et al., 2023,Alexeev et al., 2021]. Recent achievements by IBM demonstrate that noisy quantum computers can produce accurate expectation values using up to 127 qubits and 2880 gates, pushing beyond the limits of classical simulation capabilities [Kim et al., 2023]. With the release of Qiskit 2.0, the latest version of Qiskit, improvements in circuit execution and performance enable researchers to work with larger and more complex quantum circuits. In this section, we provide an overview of the IBM Quantum hardware used in our experiments, focusing on the Heron R2 architecture.

9.2.1. Overview of IBM Quantum Processors

IBM Quantum processors utilize superconducting qubits, which are known for their relatively high coherence times and gate fidelities. The architecture is built to support various quantum algorithms, with a focus on scalability and integration with classical computing resources. Over the years, IBM has released several generations of quantum processors, each with improvements in qubit connectivity, gate fidelity, and overall performance, evolving from the 5-qubit *Canary* series to the ultra-dense 1,121-qubit *Condor*. Key milestones include the *Falcon* series, which introduced scalable connectivity and multiplexed readout, and the *Hummingbird* and *Eagle* processors, which pushed coherence and qubit counts further. Recent innovations have centered around tunable couplers and improved control fidelity, exemplified by *Egret* and *Osprey*. The latest step in this progression, which is also

[1] 2

used in our experiments, is the *Heron* series, with Heron R2 offering 156 qubits arranged in a heavy-hexagonal lattice. It features improved coherence, fast gate execution, and TLS (two-level system) noise mitigation, making it IBM's most performant system to date. In this work, we utilize **Heron R2** as it is the most advanced processor currently accessible via the IBM Quantum cloud platform.

9.2.2. Heron R2 Architecture

In this section, we provide a short overview of the Heron R2 processor architecture, highlighting its key features and innovations that make it suitable for quantum computing applications. The Heron R2 processor represents a significant milestone in IBM's quantum computing architecture,[3] offering substantial improvements over previous generations. Compared to its predecessor, the R2 revision features an increased qubit count from 133 to 166, implemented in a heavy-hexagonal lattice architecture. Key advancements include tunable couplers that significantly reduce crosstalk (a limitation in the earlier Eagle processor), enhanced coherence times, and faster gate execution speeds. A notable innovation is the integration of two-level system (TLS) noise mitigation techniques, which substantially improve quantum operation fidelity. These improvements have resulted in significantly lower median two-qubit error rates, making Heron R2 IBM's most capable quantum processor to date.

The Heron R2 processor implements a heavy-hexagonal (heavy hex) lattice architecture, a qubit connectivity pattern designed to balance connectivity requirements with fabrication constraints. In this topology, qubits are arranged in a hexagonal grid where each qubit connects to either two or three nearest neighbors. This asymmetric connectivity pattern creates a lattice of hexagons where some vertices (qubits) have additional connections, hence the term "heavy hex."

The heavy-hex topology represents a strategic balance between connectivity and error reduction, developed through collaborative design between hardware, theoretical, and application teams [Carroll et al., 2024]. While it features slightly reduced qubit connectivity compared to earlier architectures, this design choice significantly improves system performance by:

- minimizing frequency collisions between adjacent qubits, which can lead to unwanted interactions and errors,
- reducing spectator errors (unwanted interactions with inactive qubits) that can degrade the fidelity of quantum operations,
- maintaining sufficient connectivity for error correction codes, which are essential for reliable quantum computation,
- providing a scalable architecture for larger quantum systems

[3] https://newsroom.ibm.com/2023-12-04-IBM-Debuts-Next-Generation-Quantum-Processor-IBM-Quantum-System-Two,-Extends-Roadmap-to-Advance-Era-of-Quantum-Utility

This design approach prioritizes reliable quantum operations over maximum connectivity, resulting in better overall performance for practical quantum applications. The experiments were conducted on IBM *Fez* (version 1.3.1), a processor with Heron R2 architecture which provides a native basis gate set consisting of CZ, ID, RX, RZ, RZZ, SX, and X gates. At the time of performing the experiments, IBM *Fez* had lowest two-qubit error rate of 1.31×10^{-3} measured through isolated randomized benchmarking, and an average layered two-qubit error rate of 5.11×10^{-3} for a 100-qubit chain. The median CZ gate error rate was 2.737×10^{-3}, indicating high-fidelity two-qubit operations. The system's performance is measured using the hardware-aware Circuit Layer Operations Per Second or the CLOPS[4] metric, which accounts for both the execution speed and the physical constraints of the quantum processor. At the time of running experiments, IBM *Fez* had speeds of 195,000 CLOPS.

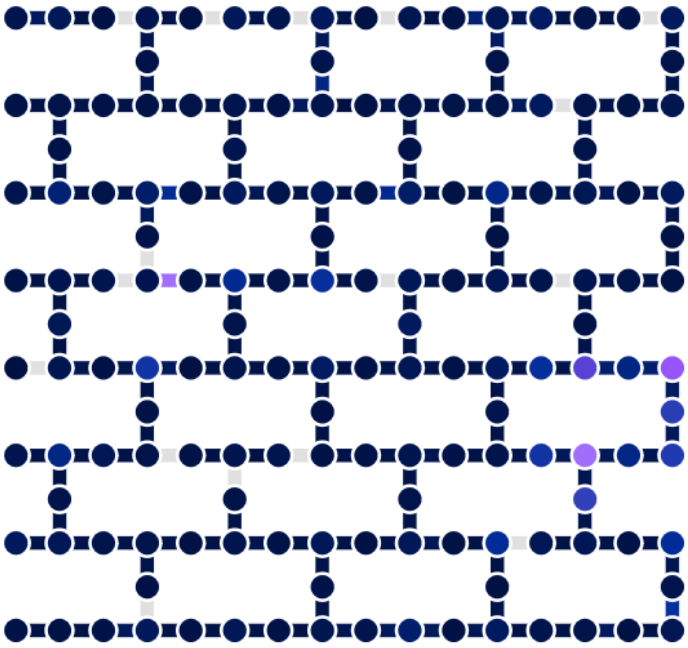


Figure 9.1.|: Heavy-hexagonal lattice architecture in Heron R2 processor IBM *Fez*, showing qubits (vertices) and their connectivity (edges). Some qubits connect to two neighbors while others connect to three, creating an asymmetric but optimal pattern for quantum error correction. Image source: IBM Quantum.

Fig. 9.1 illustrates the heavy-hexagonal lattice architecture in the Heron R2 processor IBM *Fez*, highlighting the qubits (vertices) and their connectivity (edges).

9.3. Quantum Circuit Optimization Techniques

Qiskit provides a comprehensive suite of optimization techniques to enhance quantum circuits for execution on IBM Quantum hardware. These techniques are crucial for reducing

[4] https://www.ibm.com/quantum/blog/circuit-layer-operations-per-second

circuit depth, minimizing gate counts, and improving overall performance. In this section, we discuss the key optimization levels available in Qiskit, focusing on their application to the SCOOP framework and the Heron R2 processor.

9.3.1. Optimization Levels in Qiskit

Qiskit provides different optimization levels for quantum circuit transpilation, ranging from basic transformations to sophisticated optimization techniques. We briefly describe these levels to provide context for our experimental results:

- **Level 0 (No Optimization):** Performs only necessary transformations to make circuits executable on the target hardware, such as mapping to the device topology and converting to the supported basis gate set.
- **Level 1 (Light):** Applies basic optimization passes by simple adjacent gate collapsing with minimal computational overhead.
- **Level 2 (Medium):** Introduces commutation analysis and noise-adaptive qubit mapping to identify gates that can be combined or simplified
- **Level 3 (High):** Employs heavy optimization through noise-adaptive qubit mapping, gate cancellation using commutativity rules, and unitary synthesis to maximize circuit efficiency. This level provides the most thorough optimization but requires significantly longer compilation times.

While higher optimization levels generally produce better results, they require longer compilation times and may not always yield significant improvements for every circuit.

9.3.2. Fractional Gate Decomposition

Native gate sets are the basic building blocks of quantum circuits, consisting of single-qubit and two-qubit gates that can be directly executed on quantum hardware. However, Non-native gate sets often lead to deep circuits due to decomposition with high gate counts, which can negatively impact the performance of quantum algorithms on quantum hardware. Circuit depth reduction becomes crucial when executing large-scale experiments involving hundreds of qubits and thousands of gates on current noisy quantum hardware. Fractional gates, introduced by IBM represent an important optimization technique that extends the instruction set architecture (ISA) with two additional operations, $RX(\theta)$ and $RZZ(\theta)$ enabling more efficient circuit implementations with reduced depths. This optimization is particularly relevant for QAOA as we scale to larger problem instances on IBM's quantum processors.

Given a quantum state $|\psi\rangle$, an $RX(\theta)$ gate is decomposed as follows:

$$RX(\theta)|\psi\rangle = RZ(\frac{5\pi}{2})\sqrt{X}RZ(\theta+\pi)\sqrt{X}RZ(\frac{5\pi}{2})|\psi\rangle$$

This decomposition significantly increases circuit depth, transforming a single gate into a sequence of five gates. While only the $\sqrt{X}$ gates contribute to the gate error due to IBM's implementation of RX gates, the doubling of effective gate depth becomes particularly significant when scaling to circuits with hundreds of qubits and thousands of operations.

Similarly, the $RZZ(\theta)$ gate is decomposed using four $\sqrt{X}$ gates, two controlled-Z gates and seven RZ gates. Fractional gates implement RZZ and RX gates natively, allowing for more efficient circuit implementations. Fig. 9.2 shows such a circuit with and without fractional gate decomposition. Given that QAOA circuits typically use RZZ as well as RX gates, the fractional gate decomposition is particularly useful. At the time of writing this dissertation RZZ gates can only be implemented for angles $\left[0, \frac{\pi}{2}\right]$, and therefore, we are limited too using these angles in our QAOA formulations as well.

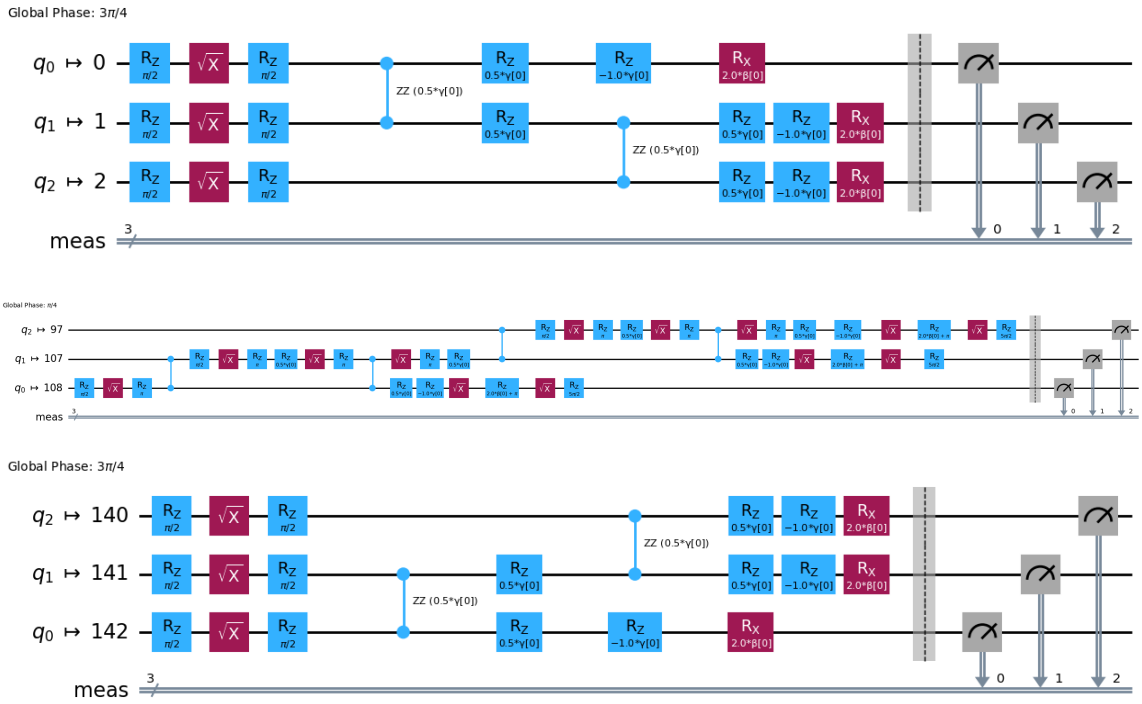


Figure 9.2.|: The first circuit shows a toy example with an optimization level of 0, which does not apply any optimization techniques. The second circuit shows the same toy example with an optimization level of 3, but without fractional gate decomposition. The third circuit shows the same toy example with an optimization level of 3, but with fractional gate decomposition applied. The fractional gate decomposition significantly reduces the circuit depth and gate count, demonstrating its effectiveness in optimizing quantum circuits for execution on IBM Quantum hardware.

9.4. Experimental Setup

In our experimental evaluation, we implemented the Quantum Approximate Optimization Algorithm (QAOA) in Qiskit to solve MINVC and MAXPC. Single 3-regular graphs of size 10, 50 and 100 were generated for testing. We configured the QAOA circuit with a single repetition (reps=1) and used error mitigation techniques including XY4 dynamical decoupling sequences [Ezzell et al., 2023]. The optimization was performed using the COBYLA method and a tolerance of $1e-2$. Different initial parameters were used: for example, we used initial parameters $\gamma = \frac{\pi}{2}$ and $\beta = \frac{\pi}{4}$, or $\gamma = \frac{\pi}{4}$ and $\frac{\pi}{8}$. Multiple parameter initializations were necessary for penalty-based approaches since the presence of penalty terms required either scaling down the Hamiltonian or reducing the angles to remain within the native RZZ gate constraints. The SCOOP framework avoided this complication entirely since its penalty-free formulation allowed direct use of the full range of available angles.

The quantum execution utilized two key Qiskit primitives: the *Estimator*, which computes expectation values of observables, and the *Sampler*, which returns measurement samples from quantum circuits. We used the estimator with 1000 shots during optimization to evaluate the cost function, followed by the sampler with 10000 shots for final state measurement to ensure statistical significance. The experiments were executed on the IBM `Fez` processor, which uses the Heron R2 architecture and experiments were conducted using Qiskit 2.0, which provides improved performance and circuit execution capabilities. Implementation details for circuit construction and execution using Qiskit 2.0, including key code snippets and configurations, are provided in Appendix D.

9.5. Results

In this section, we present the results of our experiments evaluating the SCOOP framework on IBM Quantum hardware.

Small-Scale Validation on 10 Qubits

Fig 9.3 shows the highest sampled instance (of 10000 runs) of the small-scale experiments on a 10-qubit instance of MAXPC run using QAOA with one layer. The circuit was optimized using Qiskit's optimization level 3 and initial parameters chosen were $\gamma = \frac{\pi}{2}$ and $\beta = \pi/2$. Running the classical post processing (Alg. 1) on the sampled result, we obtain a minimum vertex cover of size 7. The profit obtained from the sampled (and the post-processed) result is 8, which is a maximum profit cover.

Two-Qubit Gate Count and Circuit Depth

Figs. 9.4, 9.5 and 9.6 show the optimized circuit diagram (optimization level 3) for the 100 qubit instances of MAXPC. The circuit is divided into three parts due to its size. For graph

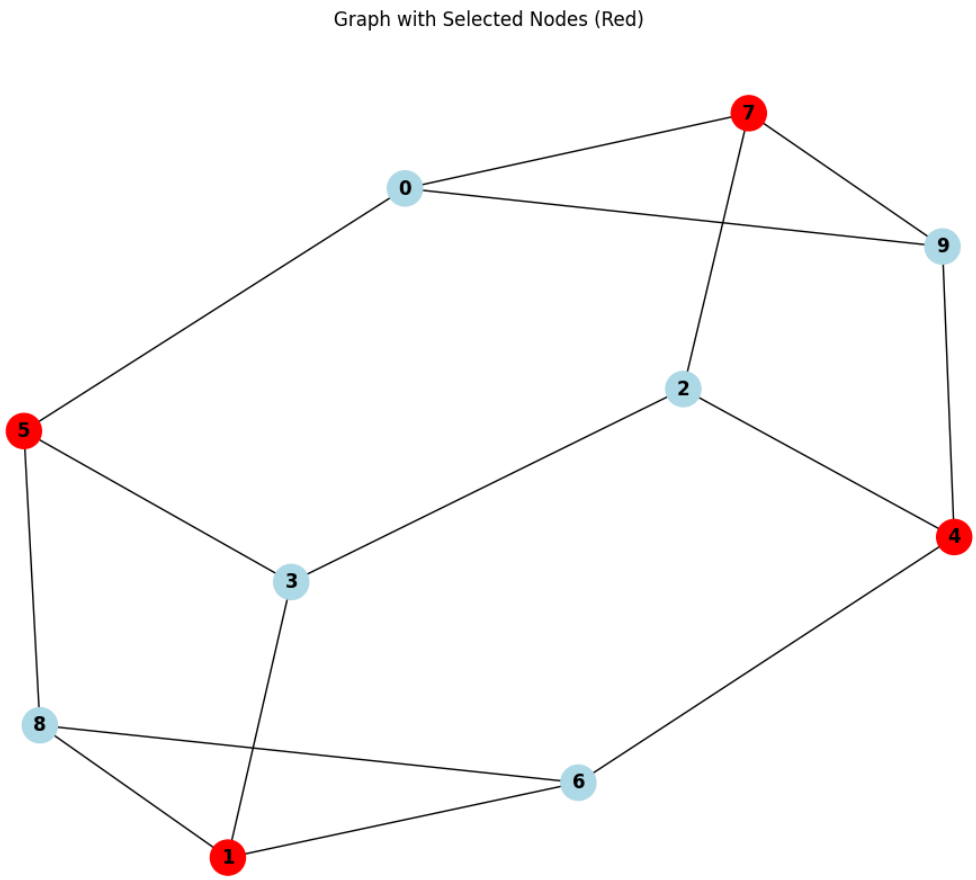


Figure 9.3.|: Highest sampled instance (of 10000 runs) of the small-scale experiments on a 10-qubit instance of MaxPC run using QAOA with one layer. Vertices in red represent the selected subset PC. This is a maximum profit cover with a profit of 8. We can post-process this result using Alg. 1 to obtain a minimum vertex cover of size 7.

instances with 100 nodes, the circuits were optimized using Qiskit's optimization level 3, which applies advanced optimization techniques to reduce circuit depth and gate counts. For the MaxPC circuit with 100 qubits, we observed a two-qubit gate depth of 165 and total gate depth of 456 for a single QAOA layer, increasing to 353 and 810, respectively, for two layers. For a 100 qubit MinVC circuit, we observed a two-qubit gate depth of 234 and total gate depth of 490 for a single QAOA layer. These measurements of two-qubit gate depth are particularly relevant for assessing circuit complexity on near-term devices, where two-qubit operations are typically the main source of errors.

Expectation Values and Solution Quality

Fig. 9.7 shows the expectation values obtained from the 100 qubit MaxPC instance with one layer (top) and two layers (bottom). Recall that the corresponding Hamiltonian $\hat{H}_{\text{PC}}$

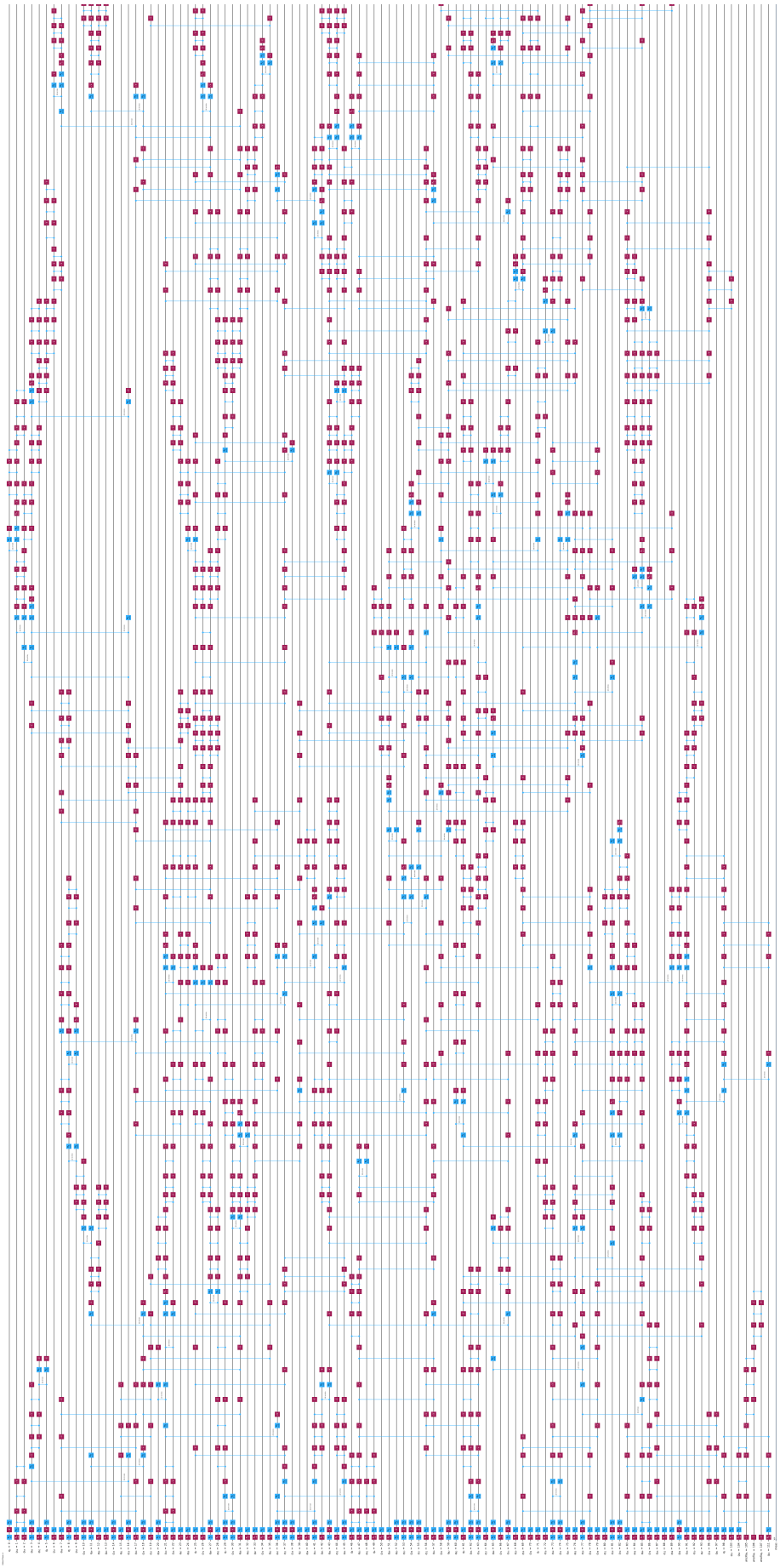


Figure 9.4.|: Optimized circuit diagram (optimization level 3) for the 100 qubit instance of MAXPC. Circuit continues on the next page.

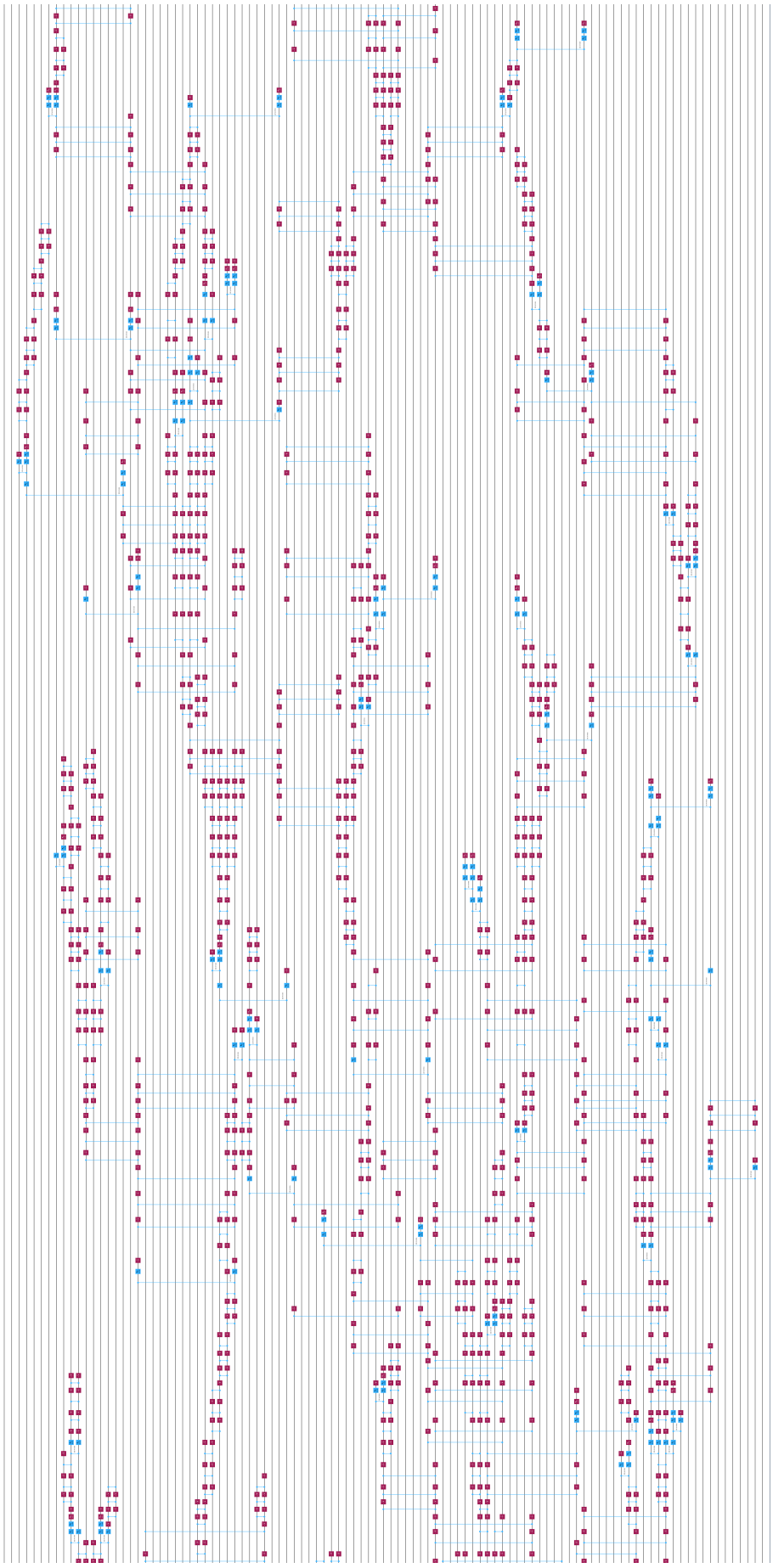


Figure 9.5.|: Optimized circuit diagram (optimization level 3) for the 100 qubit instances of MaxPC. Circuit continues on the next page.

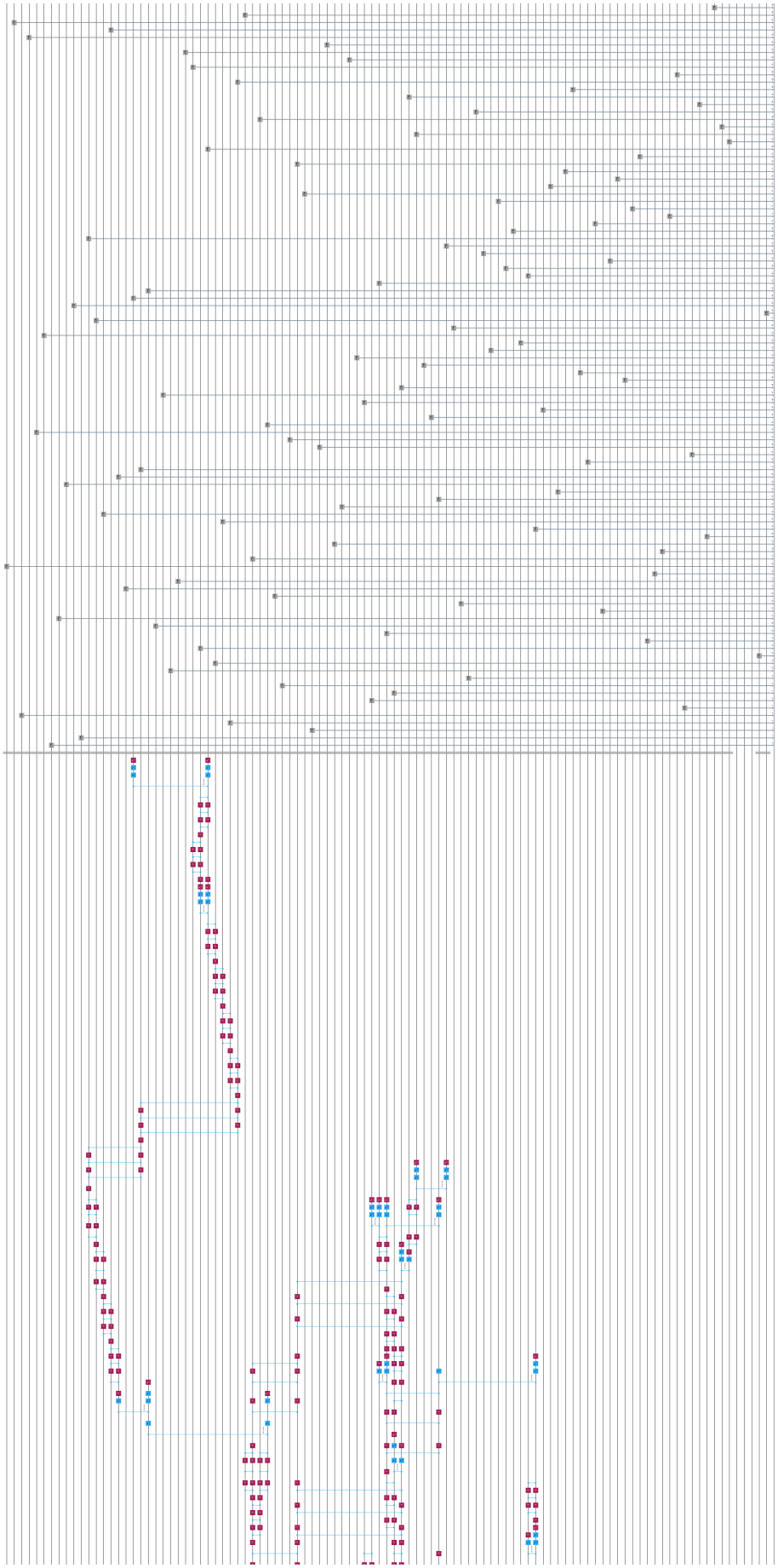


Figure 9.6.|: Optimized circuit diagram (optimization level 3) for the 100 qubit instance of MAXPC.

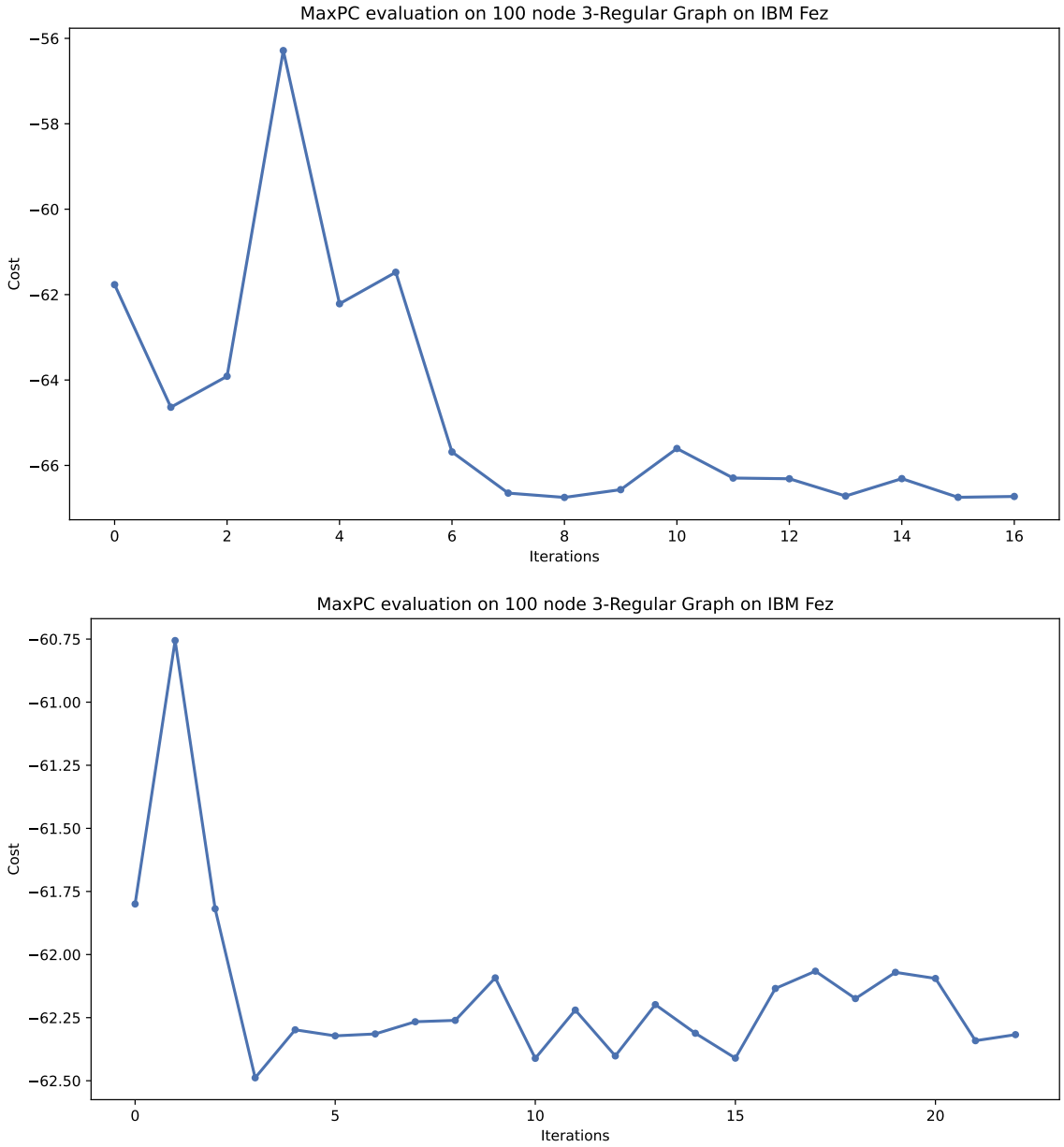


Figure 9.7.|: Expectation values for the 100 qubit MaxPC instance with one layer (top) and two layers (bottom). While MaxPC is a maximization, we use the standard minimization convention. The results indicate a better performance with one layer, as evidenced by the lower expectation values obtained than the two layer. This is due to the fact that the two-layer circuit has more gate depth and introduces additional noise, which can degrade the performance of the algorithm. The results demonstrate the importance of circuit depth optimization in achieving better solution quality on quantum hardware.

for MAXPC is defined as follows:

$$\hat{H}_{\text{PC}}^{\min} = \frac{1}{4}\sum_{uv\in E}(\hat{Z}_u\hat{Z}_v + \hat{Z}_u + \hat{Z}_v) - \frac{1}{2}\sum_{v\in V}\hat{Z}_v + \Delta_{\text{PC}} \tag{9.1}$$

where $\Delta_{\text{PC}} = \frac{|V|}{2} - \frac{3|E|}{4}$. While Δ_{PC} appears in the final expectation value calculations shown in the graphs, we exclude this constant offset during optimization as it does not influence the parameter optimization process. For the one layer circuit, the QAOA algorithm achieved an expectation value of $\text{Cost}_{PC} = -4.73$. To calculate the profit obtained, we add the constant offset Δ_{PC}. For the chosen instance $|V| = 100$ and $|E| = 150$, we have $\Delta_{\text{PC}} = 50 - 112.5 = -62$. Therefore, the profit obtained is:

$$\text{p}_{\text{PC}} = \text{Cost}_{\text{PC}} + \Delta_{\text{PC}} = -4.73 - 62 = -66.73$$

.

Note that this is negative due to minimization, but can be interpreted as a profit of at least 66 for the MAXPC instance. Given that the number of edges is 150, a vertex cover of size $150 - 66 = 84$ can be obtained. A minimum vertex cover would be smaller or equal to this size. Note that the highest sampled instance (of 10000 runs) may not correspond to this value, as the sampling process may yield different results and the expectation value obtained is a weighted average.

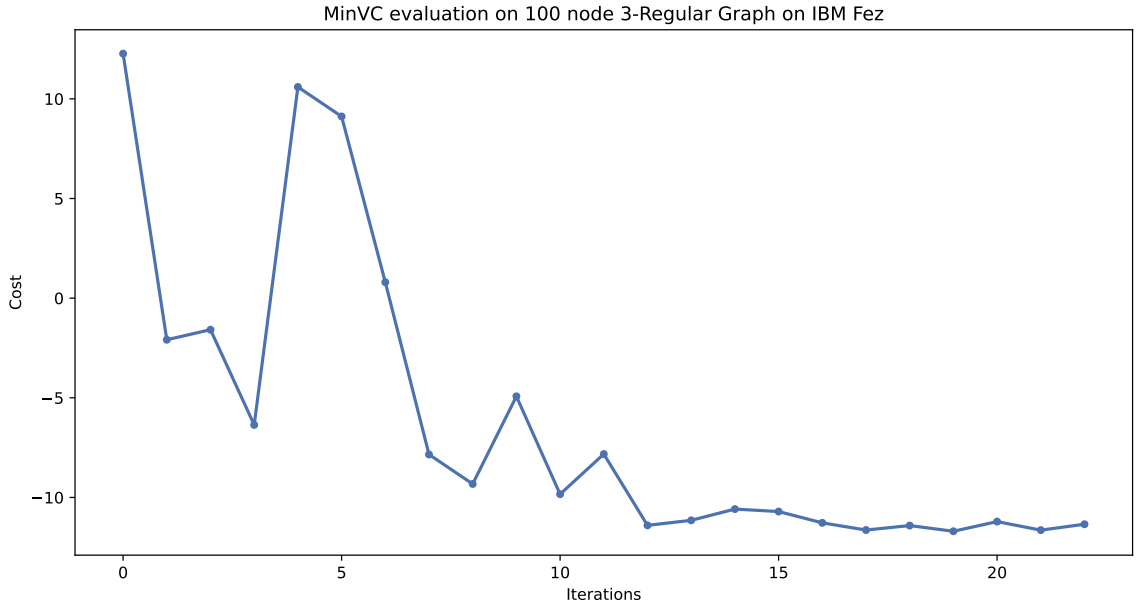


Figure 9.8.|: Expectation value for the 100 qubit MINVC instance with one layer. The initial paramters chosen were $\gamma = \frac{\pi}{4}$ and $\beta = \frac{\pi}{8}$. The two-layer circuit could not converge with the chosen initial parameters due to insufficient parameter space exploration as the angles were too small.

Fig. 9.8 shows the expectation value obtained from the 100 qubit MINVC instance with one layer. The initial parameters chosen were $\gamma = \frac{\pi}{4}$ and $\beta = \frac{\pi}{8}$ with a Hamiltonian that

was scaled down to fit within the native RZZ gate constraints. Recall the Hamiltonian $\hat{H}_{\text{VC}}$ for MINVC is defined as follows:

$$\hat{H}_{\text{VC}}^{\text{min}} = \frac{A}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v + \hat{Z}_u + \hat{Z}_v) - \frac{B}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{VC}} \tag{9.2}$$

For the experiments we used $0.5 * \hat{H}_{\text{VC}}^{\text{min}}$ with $A = 3$ and $B = 2$. The final expectation value obtained was $\text{Cost}_{\text{VC}} = -11.34$. Note that adding the offset value for the vertex cover Hamiltonian does not give us any meaningful insight into the solution quality as in the case of MAXPC.

We use the *Sampler* primitive to obtain the final state measurement results. Fig. 9.9 shows the final state measurement results for the 100 qubit instance for MINVC and MAXPC with one layer. Nodes in red represent the solution subset for both problems. The highest sampled instance shown in Fig 9.9 shows a MINVC solution of size 58 with 22 edges remaining to be covered indicating an infeasible solution. For the MAXPC instance, we obtained a profit cover selecting 54 vertices. Applying post-processing (Alg. 1) yielded a vertex cover of size 74. For comparison, we also applied the same post-processing to the infeasible MINVC solution which produced a larger vertex cover of size 79, further validating our SCOOP approach.

To validate our hardware results, we performed QTensor simulations on the identical problem instance (Figure 9.10). Note that the final visualization includes the offset calculations of Δ_{PC} and Δ_{VC}. From the simulations, we observe that the MAXPC instance converges to a value between -77 and -78 with a single layer, indicating a profit of at least 77 for the 100 qubit instance. With the simulation, we can calculate that there is a vertex cover of size $150 - 77 = 73$. This solution is 11 vertices smaller than the one obtained on IBM's quantum hardware, highlighting the impact of hardware noise and errors on solution quality compared to ideal simulation conditions.

Further analysis of the results by comparing it to the exact reference solutions is still ongoing, we have shown the preliminary results here to demonstrate the feasibility of running SCOOP on IBM Quantum hardware.

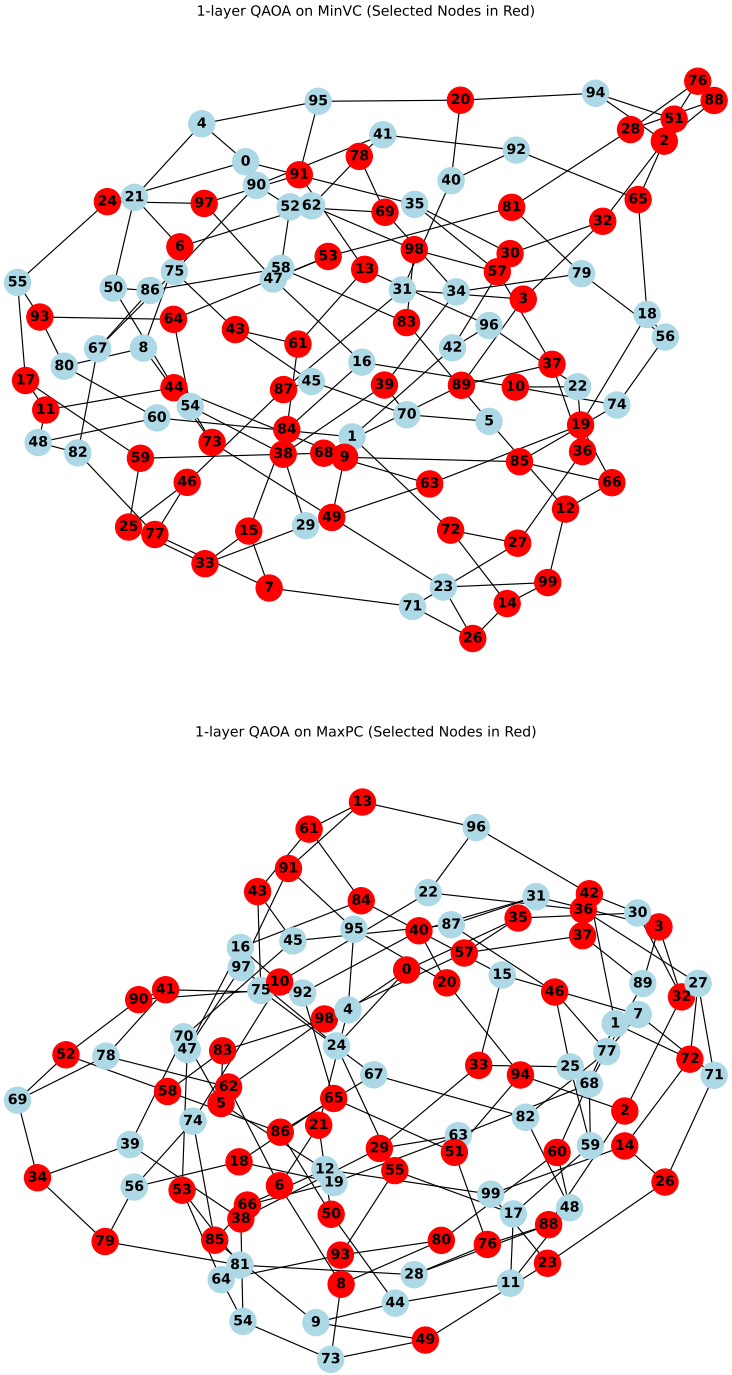


Figure 9.9.|: Final state measurement results for the 100 qubit instance for MinVC and MaxPC with one layer. Red vertices indicate the selected solution subset. The top figure shows the MinVC solution and the bottom figure shows the MaxPC solution, where the nodes in red represent the selected vertices in the profit cover.

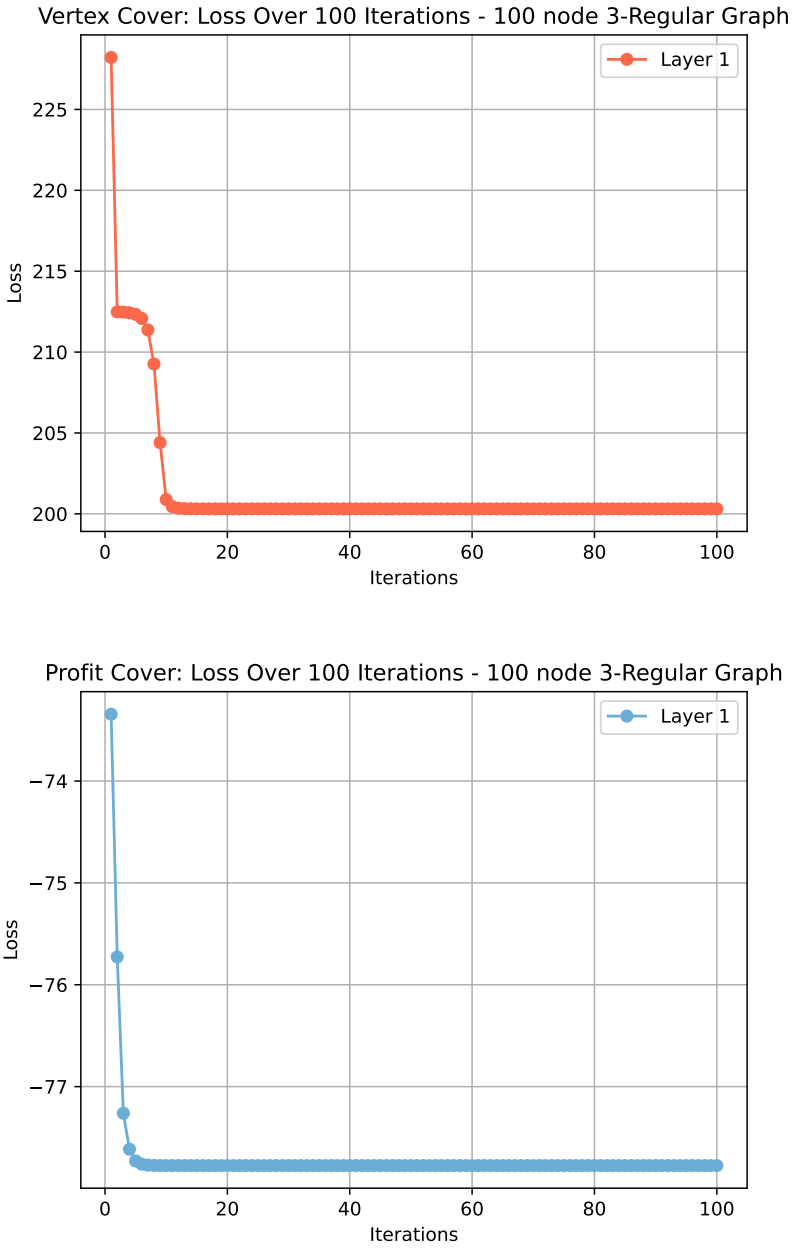


Figure 9.10.|: QTensor simulation results for the 100 qubit instance for MinVC and MaxPC with one layer. The top figure shows the QTensor simulation results for the MinVC instance and the bottom figure shows the QTensor simulation results for the MaxPC instance. The results show that QTensor can simulate the same instance with a significantly lower circuit depth and gate count compared to the IBM Quantum hardware execution.

9.6. Chapter Summary

In this chapter, we presented the results of our utility-scale experiments of the SCOOP framework on IBM Quantum hardware. We introduced the Heron R2 processor, highlighting its heavy-hexagonal lattice architecture and key features that make it suitable for quantum computing applications. We discussed the optimization techniques available in Qiskit, including fractional gate decomposition, which significantly reduces circuit depth and gate counts. We provided preliminary results of a utility-scale experiment using the SCOOP framework to solve MINVC and MAXPC on IBM Quantum hardware.

Part IV

Summary and Outlook

Chapter 10

Conclusions and Future Work

Contents

In this chapter, we summarize the contributions of this dissertation, discuss their significance, and outline future work opportunities.

10.1. Dissertation Summary

In this dissertation, we addressed the challenges of solving constrained combinatorial optimization problems using hybrid quantum-classical computing approaches. We identified the limitations of existing penalty-based methods, which often lead to sub-optimal solutions and feasibility issues. To overcome these challenges, we introduced the SCOOP framework, which transforms constrained combinatorial optimization problems into unconstrained counterparts, allowing for more effective solution exploration. Since quantum and quantum-classical methods return probabilistic results, it is crucial to ensure that high quality feasible solutions have a high probability of being sampled. The SCOOP framework operates on the concept of SCOOP twins, which are pairs of problems that share identical input domains but differ in their objective functions and solution spaces. The SCOOP framework ensures that the unconstrained twin problem is designed to capture the essence of the original constrained problem while allowing for effective exploration of the solution space. This transformation enables the use of quantum optimization techniques, such as the Quantum Approximate Optimization Algorithm (QAOA), to find high-quality solutions without being hindered by the constraints of the original problem.

10.1.1. Challenges Addressed

In this dissertation, we focused on the following key challenges related to the development of effective quantum encodings for *constrained* combinatorial optimization problems, particularly in the context of the Quantum Approximate Optimization Algorithm (QAOA) and the challenge of working within the limitations of current quantum hardware.

CH1: *Encoding of constrained combinatorial optimization problems for QAOA*. The investigation reaffirmed a core challenge in QAOA-based optimization: mapping constrained problems onto quantum circuits requires a delicate balance between simplicity and feasibility. Ineffective penalty-based encodings often yield infeasible results, whereas solution feasibility preserving encodings typically involve complex multi-qubit operations that increase circuit depth and computational demands.

CH2: *Solution Distribution*. This dissertation highlighted the importance of encoding strategies that account not only for the optimal solution but also for the broader solution landscape. Given the probabilistic nature of quantum outputs, encoding schemes must preserve the structure of near-optimal solutions to enhance practical utility under real-world quantum hardware limitations.

CH3: *Circuit Depth vs. Feasibility*. In this dissertation, we investigated the trade-offs between circuit depth and solution feasibility in the context of encoding using QUBO and HUBO strategies. Qubit-intensive QUBO formulations offer shallow circuits but come with an increased probability of infeasible solutions, whereas compact higher-order representations preserve the problem constraints better at the cost of deeper circuits. Considering hardware quality limitations and problem characteristics, the challenge lies in finding a balance between circuit depth and solution feasibility that maximizes the probability of obtaining high-quality feasible solutions.

CH4: *Scalability*. The complexity of tuning penalty-based encodings for constrained problems presents a significant bottleneck for scalability. In particular, managing the balance between the problem's penalty parameters and QAOA parameters remains a non-trivial task that complicates the training of variational circuits. We addressed this challenge by proposing SCOOP, an alternative encoding framework that reduces or eliminates the dependency on penalty parameters, thereby enhancing scalability and simplifying the optimization process.

CH5: *Trade-offs with Qubit Decoherence*. Due to the impact of qubit decoherence, deeper circuits face increased quantum noise and error rates, while shallower circuits traded accuracy for better hardware reliability. For constrained combinatorial optimization problems, this challenge is particularly pronounced, as the depth of the quantum circuit directly affects the probability of sampling high-quality feasible solutions. With the SCOOP framework, solutions that are infeasible with penalty-based approaches can still be explored effectively, as the SCOOP twins

are naturally unconstrained. Since this improves sampling probabilities, it also mitigates the impact of qubit decoherence by reducing circuit depth.

CH6: *Hybrid Workflow Overhead*. The integration of quantum and classical resources in hybrid workflows introduces communication overhead, particularly in iterative optimization procedures. One way to address this challenge to use classical tensor network simulators to calculate optimal parameters for the QAOA circuit for smaller problems and then use these parameters to run the QAOA circuit on quantum hardware for larger problems [Galda et al., 2021]. This approach aims to minimize the communication overhead by leveraging classical resources for initial parameter tuning. This was shown for MAXCUT which is an unconstrained problem. We anticipate that the SCOOP framework can be used to extend this approach to constrained combinatorial optimization problems, as SCOOP twins are unconstrained by design. This would allow for efficient parameter tuning on classical simulators, followed by execution on quantum hardware, thereby reducing the communication overhead in hybrid workflows.

The above challenges were important considerations throughout the research work for this dissertation. Out of these challenges, many research questions germinated. The next section summarizes the most important research questions together with their research findings and contributions.

10.1.2. Contributions

RQ1: What characteristics of existing penalty-based encoding approaches for constrained combinatorial optimization problems impact their ability to maintain solution feasibility and optimization quality across different problem classes?

C1: *Analysis of penalty-based QUBO formulations revealing that: (1) optimal penalty settings are instance-specific rather than problem-specific, (2) penalties that support finding optimal solutions may not necessarily support finding near-optimal solutions, and (3) strict enforcement of feasibility requires prohibitively large penalty values that impact solution quality and scalability.* This contribution highlights the limitations of penalty-based approaches to constrained combinatorial optimization problems, particularly in the context of QAOA, and sets the stage for exploring alternative encoding strategies that are better suited for obtaining high-quality solutions that satisfy feasibility constraints.

RQ2: How can we leverage the inherent limitations of penalty-based approaches for constrained combinatorial optimization problems to develop novel encoding strategies that enhance solution feasibility?

RQ2.1: How can we effectively capture both optimal and near-optimal solutions while accounting for the probabilistic nature of quantum systems and current hardware constraints?

RQ2.2: How can we leverage classical pre- and post-processing to maximize the probability of obtaining high-quality feasible solutions while minimizing QAOA circuit depth under current hardware noise and decoherence constraints?

C2: *Introduction of a penalty-free encoding strategy and its cost Hamiltonian for the constrained optimization problem* MINIMUM VERTEX COVER *using the* MAXIMUM PROFIT COVER *problem.* For the particular case of MINIMUM VERTEX COVER, we demonstrated that the penalty-free encoding effectively captures both optimal and near-optimal solutions, effectively addressing the challenges of solution feasibility and quality. This contribution establishes a foundation for further exploration of penalty-free encodings in other constrained combinatorial optimization problems.

RQ3: How can we develop a targeted encoding framework for constrained combinatorial optimization problems that reduces or eliminates penalty coefficient dependencies while maintaining solution quality?

C3: *Development of the SCOOP framework: A systematic approach for transforming constrained optimization problems into their unconstrained counterparts ensuring that the resulting problem pairs are **s**olution-enhanceable, **c**onstrained-unconstrained, **o**bjective-function-compatible **o**ptimization **p**roblem twins.* SCOOP characterizes the constraints in a constrained optimization problem and systematically transforms the problem into an unconstrained twin that can be solved using QAOA. The SCOOP framework is designed to reduce or eliminate the dependency on penalty parameters while ensuring that the resulting unconstrained twin captures the essence of the original constrained problem using classical post-processing, allowing for effective solution exploration. A comprehensive collection of SCOOP twins developed in this dissertation, spanning both NP-hard and polynomially solvable problems, is summarized in Table 10.1.

RQ4: Using such a framework, how can we identify and effectively formulate problems that can be encoded as QUBOs?

C4 - C5: *Application of SCOOP and QUBO formulations of SCOOP twins to the problems:* MAXIMUM INDEPENDENT SET*,* MAXIMUM CLIQUE*,* MAXIMUM SET PACKING*, and application to* MAX3SAT *— a naturally higher-order problem that can be encoded as a penalty-free QUBO.* We systematically identified problems that have similar kinds of constraint structures and applied the SCOOP framework to derive their unconstrained formulations. While MAX3SAT is a naturally higher-order problem, we demonstrated that it can be effectively encoded as a penalty-free QUBO using the SCOOP framework due to its relationship with MAXIMUM IS and therefore, MAXPI.

RQ5: Using such a framework, which problems can be effectively encoded as HUBOs and to what extent can we optimize the trade-off between QUBO and higher-order encodings to balance circuit depth, qubit requirements, and solution feasibility for different problem classes and hardware constraints?

C6-C8: *Derivation and formulation of HUBO-based SCOOP twins for both NP-hard problems (*MINIMUM DOMINATING SET*,* MINIMUM MAXIMAL MATCHING*,* MINIMUM EDGE

DOMINATING SET, MINIMUM SET COVER*) and classically tractable problems (*MINIMUM EDGE COVER, MAXIMUM MATCHING*), demonstrating SCOOP's versatility across complexity classes.* These contributions demonstrate SCOOP's versatility by creating novel unconstrained optimization problems that naturally have higher-order interactions. Rather than relying on penalty-based approaches or quadratization (which also introduces penalty terms), we introduce new unconstrained problems that preserve the inherent structure of higher-order constrained problems. This extension of SCOOP beyond quadratic formulations, while providing formulations across both NP-hard and polynomial-time solvable problems, represents a significant impact in encoding constrained combinatorial optimization problems for QAOA.

RQ6: How can we experimentally evaluate the effectiveness of the SCOOP framework, and what insights can we gain about the performance of QUBO and HUBO encodings in practical applications?

RQ6.1: How do these encodings compare to traditional penalty-based approaches in terms of solution feasibility and quality?

RQ6.2: How can we leverage quantum simulators to minimize the communication overhead in hybrid quantum-classical workflows while maintaining the effectiveness of iterative optimization procedures?

RQ6.3: How can SCOOP strengthen quantum computing benchmarking efforts for combinatorial optimization, and how can it be used evaluate the utility-scale performance of quantum algorithms on current hardware platforms?

C9 - C12: *Development, implementation, and evaluation of software for the SCOOP framework on simulators as well as quantum hardware including: QUBO/HUBO encodings in PennyLane, Qiskit and QTensor; test datasets with random graphs of varying sizes (n=5 to n=70) and densities (d = [0.1, 0.3, 0.5, 0.8]) as well as 3-regular graphs for benchmarking; extensions to QTensor's architecture for constrained optimization problems; and classical post-processing routines for solution feasibility.* This dissertation makes significant contributions to quantum computing benchmarking efforts by expanding beyond the traditional focus on unconstrained problems like MAXCUT. We provide a comprehensive suite of ten classically well-studied optimization problems, both constrained and unconstrained, that can serve as benchmarking targets for quantum algorithms. Our evaluation includes full statevector simulations for problems up to 14 nodes, tensor network simulations for larger instances (15-70 nodes), and real hardware experiments on IBM quantum processors. This multi-faceted evaluation approach, spanning different problem sizes and hardware platforms, establishes a more diverse and robust benchmarking framework for assessing quantum optimization algorithms. By including both NP-hard and polynomially solvable problems, our benchmark suite enables a more nuanced understanding of quantum algorithm performance across different complexity classes.

Beyond addressing our research questions, this dissertation makes the contribution of introducing novel unconstrained optimization problems such as MAXIMUM PROFIT SET

Table 10.1.: **SCOOP twins** This table summarizes the collection of SCOOP twins derived in this dissertation. The classical post-processing steps are provided to ensure that the solutions obtained from the unconstrained problems can be transformed back to feasible solutions for the original constrained problems.

CCOP	SCOOP Twin	Constraint Avoided	Classical Post-processing
MinVC	MaxPC (QUBO)	Edge coverage	Add vertices [Alg. 1]
MaxIS	MaxPI(QUBO)	Vertex independence	Remove conflicted vertices [Alg. 2]
MaxCl	MaxPCl (QUBO)	Complete subgraph	Remove conflicted vertices in complement graph [Alg. 2]
MaxSP	MaxPSP (QUBO)	Set disjointedness	Remove sets that intersect[Alg. 4]
Max3SAT	MaxPI (QUBO)	Constraints due to Quadratization	Remove conflicted vertices (corresponding to literals)
MinDS	MaxPD (HUBO)	Vertex domination	Add vertices [Alg. 1]
M^3	MaxPES (HUBO)	Maximality, matching	Add edges [Alg. 7], Ensure matching [Alg. 8]
MinEDS	MaxPES (HUBO)	Edge domination	Add edges [Alg. 7]
MinIEDS	MaxPES (HUBO)	Edge domination, edge independence	Add edges [Alg. 7], Ensure independence of edges [Alg. 8]
MinSC	MaxPSC (HUBO)	Set cover	Add elements [Alg. 10]
MinEC	MaxPEC (HUBO)	Edge cover	Add edges [Alg. 11]
MaxM	MaxPEC (HUBO)	Matching	Add edges [Alg. 11]

PACKING, MAXIMUM PROFIT SET COVER, MAXIMUM PROFITABLE EDGE SET and MAXIMUM PROFIT EDGE COVER, which are mathematically interesting independent of their quantum computing applications. The development of these problems, along with new relationships between constrained and unconstrained formulations, provides fresh perspectives on problem complexity and solution spaces that could benefit classical algorithmic approaches, opening up new directions for theoretical research.

10.2. Future Work

The contributions of this dissertation open up several avenues for future research. The SCOOP framework provides a solid foundation for further exploration of constrained combinatorial optimization problems. In this section, we outline some key areas for future work.

Applicability and limitations to the SCOOP Framework

The SCOOP framework can be applied to a wider range of constrained combinatorial optimization problems. In this dissertation, we focused on different kinds of covering and packing problems, as well as problems related to matchings. One avenue for future work is to explore the applicability of the SCOOP framework to these problems that have not been addressed in this dissertation, such as scheduling problems, routing problems, and other NP-hard problems. This would involve developing new SCOOP twins and evaluating their performance on quantum hardware. Such problems often have unique constraints including problems with linear constraints or problems with cycle or path constraints. It would be interesting to explore how the SCOOP framework can be extended to handle these constraints effectively.

QAOA Implementation Improvements

Several aspects of the QAOA implementation could be explored to enhance SCOOP's performance. While this dissertation used the standard X mixer Hamiltonian, alternative mixer Hamiltonians may better suit specific problem classes or hardware constraints, particularly for problems where certain constraints cannot be fully unconstrained. Additionally, exploring optimizers beyond RMSProp and COBYLA, such as basin-hopping [Iwamatsu and Okabe, 2004] or evolutionary algorithms, could improve convergence and solution quality. This investigation would involve analyzing solution landscapes across different problem instances to identify optimal optimizer configurations for various SCOOP problems.

Investigating more problems that do not fit the SCOOP framework

While the SCOOP framework provides a powerful tool for transforming constrained combinatorial optimization problems, there are problems that may not fit into the SCOOP framework. One of the problems considered in this dissertation, the MAX3SAT problem, is an example of a problem that does not fit the SCOOP framework as it has an unconstrained higher-order formulation. However, it does have constraints when it is quadratized. It is still possible to encode it as a QUBO problem by using its relationship with the MAXIMUM INDEPENDENT SET problem. Since MAXIMUM INDEPENDENT SET is a SCOOP problem, we can use the SCOOP framework to encode it as a QUBO problem. However, this approach does not directly follow the guidelines of the SCOOP framework but we have used a SCOOP twin to encode it as a QUBO problem. This highlights the need for further investigation into problems that do not fit the SCOOP framework and how they can be effectively encoded for quantum optimization.

Integration with QPLEX Framework

The SCOOP problems developed in this dissertation present an opportunity for integration with QPLEX, a quantum hardware-agnostic Python library for solving optimization problems [Giraldo et al., 2023]. QPLEX provides a unified interface for executing both classical and quantum optimization algorithms across different hardware platforms. Adding our SCOOP problems to QPLEX would extend its problem library with novel unconstrained formulations and enable benchmarking across different quantum platforms, including both gate-based and annealing systems. This integration would make SCOOP problems more accessible to the research community through a standardized implementation, while allowing direct comparisons between penalty-based and SCOOP approaches using QPLEX's unified interface.

Implications of the SCOOP framework for Distributed Quantum Computing

For a constrained combinatorial optimization problem, different penalty parameters can lead to different solutions. Large problems that require distributed quantum computing may have different penalty parameters for different parts of the problem (e.g., by dividing the problem or by circuit cutting [Tang et al., 2021]). This can lead to challenges in penalty setting as there would be multiple penalty parameters to set. The SCOOP framework does not require penalty parameters, which can simplify the process of setting up distributed quantum computing workflows for constrained optimization problems. The SCOOP framework can be used to encode the problem as a QUBO or HUBO problem, which can then be distributed across multiple quantum devices without the need for penalty parameters. This can lead to more effective distributed quantum computing workflows for constrained combinatorial optimization problems.

10.3. Closing Reflections

This dissertation set out to address a fundamental challenge in quantum optimization: developing effective encoding strategies to solve constrained combinatorial optimization problems on quantum computers, using hybrid quantum-classical algorithms. Through the development of the SCOOP framework, we have shown how constrained optimization problems can be systematically transformed into their unconstrained counterparts while preserving solution quality and feasibility, eliminating the need for penalty parameters and enhancing the probability of finding high-quality solutions.

The SCOOP framework represents a significant step toward making quantum optimization more practical for real-world applications. Our approach spans both NP-hard and polynomially solvable problems, demonstrating the framework's versatility across different complexity classes.

This dissertation aligns with the practicality of near-term quantum computing: quantum processors are specialized hardware that work alongside classical computers. Our SCOOP framework embraces this hybrid approach by strategically allocating computationally intensive tasks to quantum hardware while leveraging classical resources for pre- and post-processing. The SCOOP framework provides a foundation for future research in quantum optimization, offering a systematic approach that acknowledges both the potential and limitations of near-term quantum devices.

Acronyms

QUBO	Quadratic Unconstrained Binary Optimization
HUBO	Higher-order Unconstrained Binary Optimization
QAOA	Quantum Approximate Optimization Algorithm
MinVC	Minimum Vertex Cover
MaxPC	Maximum Profit Cover
CPU	Central Processing Unit
QPU	Quantum Processing Unit
GPU	Graphics Processing Unit
HQC	Hybrid Quantum-Classical
VQE	Variational Quantum Eigensolver
QA	Quantum Annealing
QSVM	Quantum Support Vector Machine
QPE	Quantum Phase Estimation
VQA	Variational Quantum Algorithm
QHPC	Quantum High-Performance Computing
MaxIS	Maximum Independent Set
MaxPI	Maximum Profit Independence
MaxCl	Maximum Clique
MaxPCl	Maximum Profit Clique
MinDS	Minimum Dominating Set
MaxPD	Maximum Profit Domination
MinEDS	Minimum Edge Dominating Set
MaxPED	Maximum Profit Edge Domination
MaxPES	Maximum Profitable Edge Set
Max3SAT	Maximum 3-Satisfiability
MaxSP	Maximum Set Packing
MaxPSP	Maximum Profitable Set Packing
M^3	Minimum Maximal Matching
MM	Maximum Matching
MinEC	Minimum Edge Cover
MaxPEC	Maximum Profit Edge Cover

Bibliography

[D-W, 2025a] (2025a). D-Wave Leap^2. dwavesys.com/take-leap.

[D-W, 2025b] (2025b). D-Wave Quantum. dwavesys.com/.

[Goo, 2025a] (2025a). Google Cirq. https://quantumai.google/cirq.

[Goo, 2025b] (2025b). Google Quantum Computing. research.google/research-areas/quantum-computing/.

[hon, 2025] (2025). Honeywell Quantum Solutions. www.honeywell.com/us/en/company/quantum.

[IBM, 2025a] (2025a). IBM Quantum. www.ibm.com/quantum-computing.

[Mic, 2025] (2025). Microsoft Quantum. https://quantum.microsoft.com/.

[Qis, 2025] (2025). Qiskit: An open-source framework for quantum computing. https://www.ibm.com/quantum/qiskit.

[Rig, 2025a] (2025a). Rigetti Forest SDK. qcs.rigetti.com/sdk-downloads.

[Rig, 2025b] (2025b). Rigetti Think Quantum. www.rigetti.com.

[Xan, 2025] (2025). Xanadu Quantum Cloud. xanadu.ai/.

[Aaronson, 2013] Aaronson, S. (2013). *Quantum computing since Democritus*. Cambridge Univ. Press.

[Aasen et al., 2025] Aasen, D., Aghaee, M., Alam, Z., Andrzejczuk, M., Antipov, A., Astafev, M., Avilovas, L., Barzegar, A., Bauer, B., Becker, J., et al. (2025). Roadmap to fault tolerant quantum computation using topological qubit arrays. *arXiv preprint arXiv:2502.12252*.

[Abbas et al., 2024] Abbas, A., Ambainis, A., Augustino, B., Bärtschi, A., Buhrman, H., Coffrin, C., Cortiana, G., Dunjko, V., Egger, D. J., Elmegreen, B. G., et al. (2024). Challenges and opportunities in quantum optimization. *Nature Reviews Physics*, pages 1–18.

[Abu-Khzam et al., 2005] Abu-Khzam, F. N., Langston, M. A., and Suters, W. H. (2005). Fast, effective vertex cover kernelization: a tale of two algorithms. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, page 16.

[Aharonov et al., 2001] Aharonov, D., Ambainis, A., Kempe, J., and Vazirani, U. (2001). Quantum walks on graphs. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 50–59.

[Alexeev et al., 2021] Alexeev, Y., Bacon, D., Brown, K. R., Calderbank, R., Carr, L. D., Chong, F. T., DeMarco, B., Englund, D., Farhi, E., Fefferman, B., Gorshkov, A. V., Houck, A., Kim, J., Kimmel, S., Lange, M., Lloyd, S., Lukin, M. D., Maslov, D., Maunz, P., Monroe, C., Preskill, J., Roetteler, M., Savage, M. J., and Thompson, J. (2021). Quantum computer systems for scientific discovery. *PRX Quantum*, 2:017001.

[Ambainis, 2007] Ambainis, A. (2007). Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239.

[Ambainis et al., 2019] Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prūsis, K., and Vihrovs, J. (2019). Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1783–1793. ACM.

[Angara et al., 2025a] Angara, P. P., Lykov, D., Stege, U., Alexeev, Y., and Müller, H. (2025a). The art of avoiding constraints: A penalty-free approach to constrained combinatorial optimization with QAOA. *arXiv preprint arXiv:2503.10077*.

[Angara et al., 2025b] Angara, P. P., Martins, E., Stege, U., and Müller, H. (2025b). Scoop: A quantum-computing framework for constrained combinatorial optimization. *arXiv preprint arXiv:2504.10897*.

[Angara et al., 2020] Angara, P. P., Stege, U., Müller, H. A., and Bozzo-Rey, M. (2020). Hybrid quantum-classical problem solving in the nisq era. In *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, CASCON 2020, page 247–252. IBM Corp.

[Anschuetz et al., 2018] Anschuetz, E. R., Olson, J. P., Aspuru-Guzik, A., and Cao, Y. (2018). Variational quantum factoring.

[Arfken et al., 2013] Arfken, G. B., Weber, H. J., and Harris, F. E. (2013). Chapter 22 - calculus of variations. In Arfken, G. B., Weber, H. J., and Harris, F. E., editors, *Mathematical Methods for Physicists (Seventh Edition)*, pages 1081–1124. Academic Press, Boston, seventh edition edition.

[Ausiello et al., 2012] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. (2012). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media.

[Bärtschi and Eidenbenz, 2019] Bärtschi, A. and Eidenbenz, S. (2019). Deterministic preparation of dicke states. In *International Symposium on Fundamentals of Computation Theory*, pages 126–139. Springer.

[Bärtschi and Eidenbenz, 2020] Bärtschi, A. and Eidenbenz, S. (2020). Grover mixers for QAOA: Shifting complexity from mixer design to state preparation. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 72–82.

[Bazgan et al., 2005] Bazgan, C., Escoffier, B., and Paschos, V. T. (2005). Completeness in standard and differential approximation classes: Poly-(D) APX-and (D) PTAS-completeness. *Theoretical Computer Science*, 339(2-3):272–292.

[Bellman, 1962] Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63.

[Bergholm et al., 2018] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M. S., Alonso-Linaje, G., AkashNarayanan, B., Asadi, A., et al. (2018). Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*.

[Berman and Fürer, 1994] Berman, P. and Fürer, M. (1994). Approximating maximum independent set in bounded degree graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 365–371.

[Bittel and Kliesch, 2021] Bittel, L. and Kliesch, M. (2021). Training variational quantum algorithms is NP-hard. *Physical Review Letters*, 127(12):120502.

[Bonet-Monroig et al., 2023] Bonet-Monroig, X., Wang, H., Vermetten, D., Senjean, B., Moussa, C., Bäck, T., Dunjko, V., and O'Brien, T. E. (2023). Performance comparison of optimization methods on variational quantum algorithms. *Physical Review A*, 107(3):032407.

[Briegel et al., 2009] Briegel, H. J., Browne, D. E., Dür, W., Raussendorf, R., and Van den Nest, M. (2009). Measurement-based quantum computation. *Nature Physics*, 5(1):19–26.

[Brooks, 2019] Brooks, M. (2019). Beyond quantum supremacy: the hunt for useful quantum computers. *Nature*, 574(7776):19–22.

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

[Campbell and Dahl, 2022] Campbell, C. and Dahl, E. (2022). QAOA of the highest order. In *Proceedings 19th IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 141–146. IEEE.

[Carroll et al., 2024] Carroll, M. S., Wootton, J. R., and Cross, A. W. (2024). Subsystem surface and compass code sensitivities to non-identical infidelity distributions on heavy-hex lattice. *arXiv preprint arXiv:2402.08203*.

[Cerezo et al., 2021] Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., et al. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

[Chalupnik et al., 2022] Chalupnik, M., Melo, H., Alexeev, Y., and Galda, A. (2022). Augmenting QAOA ansatz with multiparameter problem-independent layer. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 97–103.

[Chancellor et al., 2016] Chancellor, N., Zohren, S., Warburton, P. A., Benjamin, S. C., and Roberts, S. (2016). A direct mapping of max k-sat and high order parity checks to a chimera graph. *Scientific reports*, 6(1):37107.

[Chandarana et al., 2022] Chandarana, P., Hegade, N. N., Paul, K., Albarrán-Arriagada, F., Solano, E., Del Campo, A., and Chen, X. (2022). Digitized-counterdiabatic quantum approximate optimization algorithm. *Physical Review Research*, 4(1):013141.

[Choi, 2010] Choi, V. (2010). Adiabatic quantum algorithms for the np-complete maximum-weight independent set, exact cover and 3sat problems. *arXiv preprint arXiv:1004.2226*.

[Chor et al., 2004] Chor, B., Fellows, M., and Juedes, D. (2004). Linear kernels in linear time, or how to save k colors in o (n 2) steps. In *International Workshop on Graph-theoretic Concepts in Computer Science*, pages 257–269. Springer.

[Cook et al., 2020] Cook, J., Eidenbenz, S., and Bärtschi, A. (2020). The quantum alternating operator ansatz on maximum k-vertex cover. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 83–92.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158.

[Córcoles et al., 2019] Córcoles, A. D., Kandala, A., Javadi-Abhari, A., McClure, D. T., Cross, A. W., Temme, K., Nation, P. D., Steffen, M., and Gambetta, J. M. (2019). Challenges and opportunities of near-term quantum computing systems. *arXiv preprint arXiv:1910.02894*.

[Cowtan et al., 2019] Cowtan, A., Dilkes, S., Duncan, R., Simmons, W., and Sivarajah, S. (2019). Phase gadget synthesis for shallow circuits. *arXiv preprint arXiv:1906.01734*.

[Cygan et al., 2015] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

[de Berg et al., 2023] de Berg, M., Sadhukhan, A., and Spieksma, F. (2023). Stable approximation algorithms for dominating set and independent set. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*.

[Dietzfelbinger, 2008] Dietzfelbinger, M. (2008). Sanjoy dasgupta, christos papadimitriou, umesh vazirani, algorithms, mcgraw hill, boston (2007), p. x+ 320, paperback $33.75, isbn : 978 - 007352340 - 8 jonkleinberg, vatardos, algorithmdesign, pearson/addisonwesley, boston(2006), p.xxiii+ 838, hardcover$103, isbn: 978-032129535-4.

[Dinneen and Hua, 2017a] Dinneen, M. J. and Hua, R. (2017a). Formulating graph covering problems for adiabatic quantum computers. In *Proceedings Australasian Computer Science Week Multiconference (ASW 2017)*. ACM.

[Dinneen and Hua, 2017b] Dinneen, M. J. and Hua, R. (2017b). Formulating graph covering problems for adiabatic quantum computers. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–10.

[Downey and Fellows, 2013a] Downey, R. and Fellows, M. (2013a). *Fundamentals of Parameterized Complexity*. Springer.

[Downey et al., 1998] Downey, R., Fellows, M., and Stege, U. (1998). Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Ser. in Discr. Mathem. and TCS*, 49.

[Downey et al., 1999a] Downey, R., Fellows, M., and Stege, U. (1999a). Parameterized complexity: A framework for systematically confronting computational intractability. *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, 49:49–99.

[Downey and Fellows, 1995] Downey, R. G. and Fellows, M. R. (1995). Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on computing*, 24(4):873–921.

[Downey and Fellows, 2013b] Downey, R. G. and Fellows, M. R. (2013b). *Fundamentals of Parameterized Complexity*, volume 4. Springer.

[Downey et al., 1999b] Downey, R. G., Fellows, M. R., and Stege, U. (1999b). Computational tractability: The view from mars. *Bulletin of the EATCS*, 69:73–97.

[Downey et al., 1999c] Downey, R. G., Fellows, M. R., and Stege, U. (1999c). Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49, pages 49–99.

[Dürr and Høyer, 1996] Dürr, C. and Høyer, P. (1996). A quantum algorithm for finding the minimum. *CoRR*, 9607014.

[Edmonds, 1965] Edmonds, J. (1965). Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56.

[Egger et al., 2021] Egger, D. J., Mareček, J., and Woerner, S. (2021). Warm-starting quantum optimization. *Quantum*, 5:479.

[Erd́os et al., 1960] Erd́os, P., Rényi, A., et al. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5(1):17–60.

[Ezzell et al., 2023] Ezzell, N., Pokharel, B., Tewala, L., Quiroz, G., and Lidar, D. A. (2023). Dynamical decoupling for superconducting qubits: A performance survey. *Physical Review Applied*, 20(6):064027.

[Farhi et al., 2020] Farhi, E., Gamarnik, D., and Gutmann, S. (2020). The quantum approximate optimization algorithm needs to see the whole graph: A typical case. *arXiv preprint arXiv:2004.09002*.

[Farhi et al., 2014] Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*.

[Farhi and Harrow, 2016] Farhi, E. and Harrow, A. W. (2016). Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint arXiv:1602.07674*.

[Fernau and Stege, 2019] Fernau, H. and Stege, U. (2019). Profit parameterizations of dominating set. In *Proceedings 13th International Conference Algorithmic Aspects in Information and Management (AAIM 2019), Beijing, China*, pages 108–120. Springer.

[Feynman, 1982] Feynman, R. P. (1982). Simulating physics with computers. *Int. J. Theor. Phys.*, 21(6–7):467–488.

[Fomin et al., 2019] Fomin, F., Lokshtanov, D., Saurabh, S., and Zehavi, M. (2019). *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press.

[Fomin and Kratsch, 2010] Fomin, F. V. and Kratsch, D. (2010). *Exact Exponential Algorithms*. Springer.

[Gabbassov et al., 2023] Gabbassov, E., Rosenberg, G., and Scherer, A. (2023). Lagrangian duality in quantum optimization: Overcoming qubo limitations for constrained problems. *arXiv preprint arXiv:2310.04542*.

[Gabor et al., 2019] Gabor, T., Zielinski, S., Feld, S., Roch, C., Seidel, C., Neukart, F., Galter, I., Mauerer, W., and Linnhoff-Popien, C. (2019). Assessing solution quality of 3sat on a quantum annealing platform. In *Quantum Technology and Optimization Problems: First International Workshop, QTOP 2019, Munich, Germany, March 18, 2019, Proceedings 1*, pages 23–35. Springer.

[Galda et al., 2021] Galda, A., Liu, X., Lykov, D., Alexeev, Y., and Safro, I. (2021). Transferability of optimal qaoa parameters between random graphs. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 171–180. IEEE.

[Gambetta, 2022] Gambetta, J. (2022). Quantum-centric supercomputing: The next wave of computing. *IBM Research Blog*.

[Gambetta, 2021] Gambetta, J. (research.ibm.com/blog/ibm-quantum-roadmap. 2021.). IBM's roadmap for scaling quantum technology.

[Gambetta et al., 2021] Gambetta, J., Faro, I., and Wehden, K. (research.ibm.com/blog/quantum-development-roadmap. 2021.). IBM's roadmap for building an open quantum software ecosystem.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, volume 174. freeman San Francisco.

[Ge and Dunjko, 2020] Ge, Y. and Dunjko, V. (2020). A hybrid algorithm framework for quantum computers with application to finding hamiltonian cycles. *J. Math. Phys.*, 61.

[Giraldo et al., 2023] Giraldo, J., Ossorio, J., Villegas, N. M., Tamura, G., and Stege, U. (2023). QPLEX: Realizing the integration of quantum computing into combinatorial optimization software. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 1044–1049. IEEE.

[Glos et al., 2022] Glos, A., Krawiec, A., and Zimborás, Z. (2022). Space-efficient binary optimization for variational quantum computing. *npj | Quantum Information*, 8(1):39.

[Glover et al., 2022] Glover, F., Kochenberger, G., Hennig, R., and Du, Y. (2022). Quantum bridge analytics I: a tutorial on formulating and using QUBO models. *Annals of Operations Research*, 314(1):141–183.

[Golden et al., 2023] Golden, J., Bärtschi, A., O'Malley, D., and Eidenbenz, S. (2023). Numerical evidence for exponential speed-up of QAOA over unstructured search for approximate constrained optimization. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 496–505.

[Grover, 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC 1996)*, pages 212–219.

[Grover and Radhakrishnan, 2005] Grover, L. K. and Radhakrishnan, J. (2005). Is partial quantum search of a database any easier? In *Proceedings of the 17th Annnual ACM Symposium Parallel Algorithms and Architectures*, pages 186–194.

[Hadfield et al., 2019] Hadfield, S., Wang, Z., O'gorman, B., Rieffel, E. G., Venturelli, D., and Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34.

[Harris and Narayanaswamy, 2024] Harris, D. G. and Narayanaswamy, N. (2024). A faster algorithm for vertex cover parameterized by solution size. In *41st International Symposium on Theoretical Aspects of Computer Science*.

[Harrow et al., 2009] Harrow, A. W., Hassidim, A., and Lloyd, S. (2009). Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103(150502).

[He et al., 2017] He, Y., Luo, M.-X., Zhang, E., Wang, H.-K., and Wang, X.-F. (2017). Decompositions of n-qubit Toffoli gates with linear circuit complexity. *International Journal of Theoretical Physics*, 56:2350–2361.

[Held and Karp, 1961] Held, M. and Karp, R. M. (1961). A dynamic programming approach to sequencing problems. In *Proceedings of the 16th ACM National Meeting*, pages 201–204.

[Herrman et al., 2022] Herrman, R., Lotshaw, P. C., Ostrowski, J., Humble, T. S., and Siopsis, G. (2022). Multi-angle quantum approximate optimization algorithm. *Scientific Reports*, 12(1):6781.

[Herrman et al., 2021] Herrman, R., Treffert, L., Ostrowski, J., Lotshaw, P. C., Humble, T. S., and Siopsis, G. (2021). Impact of graph structures for QAOA on MaxCut. *Quantum Information Processing*, 20(9):289.

[Hromkovič, 2013] Hromkovič, J. (2013). *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer Science & Business Media.

[Huembeli and Dauphin, 2021] Huembeli, P. and Dauphin, A. (2021). Characterizing the loss landscape of variational quantum circuits. *Quantum Science and Technology*, 6(2):025011.

[Håstad, 1999] Håstad, J. (1999). Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142.

[IBM, 2025b] IBM (2025b). IBM Expands Quantum Data Center in Poughkeepsie, New York to Advance Algorithm Discovery Globally. https://newsroom.ibm.com/2024-09-26-ibm-expands-quantum-data-center-in-poughkeepsie,-new-york-to-advance-algorithm-discovery-globally. [Accessed 14-04-2025].

[IBM Quantum, 2023] IBM Quantum (2023). What is quantum utility? https://www.ibm.com/quantum/blog/what-is-quantum-utlity.

[Ibrahim et al., 2022] Ibrahim, C., Lykov, D., He, Z., Alexeev, Y., and Safro, I. (2022). Constructing optimal contraction trees for tensor network quantum circuit simulation. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE.

[Iwamatsu and Okabe, 2004] Iwamatsu, M. and Okabe, Y. (2004). Basin hopping with occasional jumping. *Chemical Physics Letters*, 399(4-6):396–400.

[Johnson et al., 2011] Johnson, M. W., Amin, M. H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., et al. (2011). Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198.

[Kameda and Munro, 1974] Kameda, T. and Munro, I. (1974). A $o(|v|.|e|)$ algorithm for maximum matching of graphs. *Computing*, 12(1):91–98.

[Kandala et al., 2017a] Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J. M., and Gambetta, J. M. (2017a). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246.

[Kandala et al., 2017b] Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J. M., and Gambetta, J. M. (2017b). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246.

[Kantorovich and Akilov, 2014] Kantorovich, L. V. and Akilov, G. P. (2014). *Functional analysis*. Elsevier.

[Karakostas, 2009] Karakostas, G. (2009). A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4):1–8.

[Karalekas et al., 2020] Karalekas, P. J., Tezak, N. A., Peterson, E. C., Ryan, C. A., da Silva, M. P., and Smith, R. S. (2020). A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Sci. Techn.*, 5(2):024003.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer.

[Kim et al., 2023] Kim, Y., Eddins, A., Anand, S., Wei, K. X., Van Den Berg, E., Rosenblatt, S., Nayfeh, H., Wu, Y., Zaletel, M., Temme, K., et al. (2023). Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618(7965):500–505.

[Koch et al., 2025] Koch, T., Neira, D. E. B., Chen, Y., Cortiana, G., Egger, D. J., Heese, R., Hegade, N. N., Cadavid, A. G., Huang, R., Itoko, T., et al. (2025). Quantum optimization benchmark library–the intractable decathlon. *arXiv preprint arXiv:2504.03832*.

[Kulshrestha and Safro, 2022] Kulshrestha, A. and Safro, I. (2022). Beinit: Avoiding barren plateaus in variational quantum algorithms. In *2022 IEEE international conference on quantum computing and engineering (QCE)*, pages 197–203. IEEE.

[Lanes et al., 2025] Lanes, O., Beji, M., Corcoles, A. D., Dalyac, C., Gambetta, J. M., Henriet, L., Javadi-Abhari, A., Kandala, A., Mezzacapo, A., Porter, C., et al. (2025). A framework for quantum advantage. *arXiv preprint arXiv:2506.20658*.

[Lee et al., 2019] Lee, Y., Joo, J., and Lee, S. (2019). Hybrid quantum linear equation algorithm and its experimental test on IBM quantum experience. *Nature Scient. Reports*, 9.

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

[Lloyd and Braunstein, 1999] Lloyd, S. and Braunstein, S. L. (1999). Quantum computation over continuous variables. *Physical Review Letters*, 82(8):1784.

[Lucas, 2014] Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2.

[Lykov et al., 2021] Lykov, D., Chen, A., Chen, H., Keipert, K., Zhang, Z., Gibbs, T., and Alexeev, Y. (2021). Performance evaluation and acceleration of the QTensor quantum circuit simulator on GPUs. In *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*, pages 27–34.

[Lykov et al., 2022] Lykov, D., Schutski, R., Galda, A., Vinokur, V., and Alexeev, Y. (2022). Tensor network quantum simulator with step-dependent parallelization. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE22)*, pages 582–593.

[Malan and Engelbrecht, 2013] Malan, K. M. and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163.

[Mandal et al., 2020] Mandal, A., Roy, A., Upadhyay, S., and Ushijima-Mwesigwa, H. (2020). Compressed quadratization of higher order binary optimization problems. In *Proceedings 17th ACM International Conference on Computing Frontiers*, pages 126–131.

[Markov and Shi, 2008] Markov, I. L. and Shi, Y. (2008). Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981.

[Martinis, 2004] Martinis, J. M. (2004). Superconducting qubits and the physics of josephson junctions. In *Les Houches*, volume 79, pages 487–520. Elsevier.

[McArdle et al., 2020] McArdle, S., Endo, S., Aspuru-Guzik, A., Benjamin, S., and Yuan, X. (2020). Quantum computational chemistry. *Rev. Mod. Phys.*, 92(1):015003.

[Micali and Vazirani, 1980] Micali, S. and Vazirani, V. V. (1980). An $o(\sqrt{|V|.|E|})$ algoithm for finding maximum matching in general graphs. In *21st Annual symposium on foundations of computer science (Sfcs 1980)*, pages 17–27. IEEE.

[Microsoft Azure, 2025] Microsoft Azure (2025). Microsoft Quantum Development Kit. azure.microsoft.com/en-ca/solutions/quantum-computing/.

[Mirkarimi et al., 2024] Mirkarimi, P., Shukla, I., Hoyle, D. C., Williams, R., and Chancellor, N. (2024). Quantum optimization with linear ising penalty functions for customer data science. *Physical Review Research*, 6(4):043241.

[Mount, 2021] Mount, E. (blog.google/technology/research/2021-year-review-google-quantum-ai/. 2021.). 2021 Year in Review: Google Quantum AI.

[Nation et al., 2025] Nation, P. D., Saki, A. A., Brandhofer, S., Bello, L., Garion, S., Treinish, M., and Javadi-Abhari, A. (2025). Benchmarking the performance of quantum computing software for quantum circuit creation, manipulation and compilation. *Nature Computational Science*, pages 1–9.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

[Nemhauser and Trotter, 1975] Nemhauser, G. L. and Trotter, L. E. (1975). Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248.

[Nielsen and Chuang, 2011] Nielsen, M. A. and Chuang, I. L. (2011). *Quantum Computation and Quantum Information*. Cambridge University Press.

[Nvidia, 2024] Nvidia (2024). NVIDIA CUDA-Q Introduces more capabilities for quantum accelerated supercomputing. https://developer.nvidia.com/blog/cuda-quantum-introduces-more-capabilities-for-quantum-accelerated-supercomputing/.

[Orús, 2014] Orús, R. (2014). A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of physics*, 349:117–158.

[Papadimitriou and Yannakakis, 1988] Papadimitriou, C. and Yannakakis, M. (1988). Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234.

[Papadimitriou, 1997] Papadimitriou, C. H. (1997). NP-completeness: A retrospective. In *International Colloquium on Automata, Languages, and Programming*, pages 2–6. Springer.

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

[Peierls, 1936] Peierls, R. (1936). On Ising's model of ferromagnetism. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 32, pages 477–481. Cambridge University Press.

[Pellow-Jarman et al., 2021] Pellow-Jarman, A., Sinayskiy, I., Pillay, A., and Petruccione, F. (2021). A comparison of various classical optimizers for a variational quantum linear solver. *Quantum Information Processing*, 20.

[Pelofske et al., 2023a] Pelofske, E., Bärtschi, A., and Eidenbenz, S. (2023a). Quantum annealing vs. QAOA: 127 qubit higher-order Ising problems on NISQ computers. In *International Conference on High Performance Computing*, pages 240–258. Springer.

[Pelofske et al., 2019] Pelofske, E., Hahn, G., and Djidjev, H. (2019). Solving large minimum vertex cover problems on a quantum annealer. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 76–84.

[Pelofske et al., 2023b] Pelofske, E., Hahn, G., and Djidjev, H. N. (2023b). Solving larger maximum clique problems using parallel quantum annealing. *Quantum Information Processing*, 22(5):219.

[Peruzzo et al., 2014] Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O'Brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213.

[Powell, 2007] Powell, M. J. (2007). A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5):170–174.

[Preskill, 2018] Preskill, J. (2018). Quantum computing in the nisq era and beyond. *Quantum*, 2:79.

[Rattew et al., 2019] Rattew, A. G., Hu, S., Pistoia, M., Chen, R., and Wood, S. (2019). A domain-agnostic, noise-resistant, hardware-efficient evolutionary variational quantum eigensolver. *arXiv preprint arXiv:1910.09694*.

[Robson, 2001] Robson, J. M. (2001). Finding a maximum independent set in time $O(2^{n/4})$. Technical report, Technical Report 1251-01, LaBRI, Université Bordeaux I.

[Roch et al., 2023] Roch, C., Ratke, D., Nüßlein, J., Gabor, T., and Feld, S. (2023). The effect of penalty factors of constrained Hamiltonians on the Eigenspectrum in quantum annealing. *ACM Transactions on Quantum Computing*, 4(2):1–18.

[Saleem, 2020] Saleem, Z. H. (2020). Max-independent set and the quantum alternating operator ansatz. *International Journal of Quantum Information*, 18(04):2050011.

[Saleem et al., 2023] Saleem, Z. H., Tomesh, T., Tariq, B., and Suchara, M. (2023). Approaches to constrained quantum approximate optimization. *Springer Nature Computer Science*, 4(2):183.

[Schrijver, 2005] Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68.

[Scott, 2004] Scott, A. E. J. (2004). Classical and parameterized complexity of cliques and games. Master's Thesis.

[Shaydulin et al., 2023] Shaydulin, R. et al. (2023). Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem. *arXiv preprint arXiv:2308.02342*.

[Shaydulin and Pistoia, 2023] Shaydulin, R. and Pistoia, M. (2023). QAOA with $n \cdot p \geq 200$. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 1074–1077.

[Shor, 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium Foundations of Computer Science*, FOCS, pages 124–134.

[Smith et al., 1997] Smith, A. E., Coit, D. W., Baeck, T., Fogel, D., and Michalewicz, Z. (1997). Penalty functions. *Handbook of Evolutionary Computation*, 97(1):C5.

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

[Stege, 1999] Stege, U. (1999). *Resolving conflicts in problems from computational biology*. PhD thesis, ETH Zurich.

[Stege et al., 2002a] Stege, U., van Rooij, I., Hertel, A., and Hertel, P. (2002a). An $o(pn + 1.151^p)$-algorithm for p-profit cover and its practical implications for vertex cover. In *Algorithms and Computation: 13th International Symposium (ISAAC 2002)*, pages 249–261. Springer.

[Stege et al., 2002b] Stege, U., van Rooij, I., Hertel, A., and Hertel, P. (2002b). An $O(pn + 1.151^p)$-algorithm for p-profit cover and its practical implications for vertex cover. In Bose, P. and Morin, P., editors, *Algorithms and Computation*, pages 249–261. Springer.

[Streif and Leib, 2020] Streif, M. and Leib, M. (2020). Training the quantum approximate optimization algorithm without access to a quantum processing unit. *Quantum Science and Technology*, 5(3):034008.

[Sung et al., 2020] Sung, K. J., Yao, J., Harrigan, M. P., Rubin, N. C., Jiang, Z., Lin, L., Babbush, R., and McClean, J. R. (2020). Using models to improve optimizers for variational quantum algorithms. *Quantum Science and Technology*, 5(4):044008.

[Suzuki, 1993] Suzuki, M. (1993). Improved trotter-like formula. *Physics Letters A*, 180(3):232–234.

[Tang et al., 2021] Tang, W., Tomesh, T., Suchara, M., Larson, J., and Martonosi, M. (2021). Cutqc: using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*, pages 473–486.

[Tomesh et al., 2023] Tomesh, T., Saleem, Z. H., Perlin, M. A., Gokhale, P., Suchara, M., and Martonosi, M. (2023). Divide and conquer for combinatorial optimization and distributed quantum computation. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 1–12.

[Van Rooij, 2003] Van Rooij, I. (2003). *Tractable cognition: Complexity theory in cognitive psychology*. PhD thesis.

[Vazirani, 2001] Vazirani, V. V. (2001). *Approximation Algorithms*, volume 1. Springer.

[Wang et al., 2025] Wang, B.-Y., Cui, X., Zeng, Q., Zhan, Y., Yung, M.-H., and Shi, Y. (2025). Speedup of high-order unconstrained binary optimization using quantum $${{\mathbb{z}}}_{2}$$lattice gauge theory. *Communications Physics*, 8(1):150.

[Wang and Tian, 2011] Wang, X. and Tian, J. (2011). Dynamic programming for NP-hard problems. *Procedia Engineering*, 15:3396–3400.

[Wilson et al., 2022] Wilson, E., Mueller, F., and Pakin, S. (2022). Combining hard and soft constraints in quantum constraint-satisfaction systems. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE.

[Wurtz et al., 2022] Wurtz, J., Lopes, P. L., Gorgulla, C., Gemelke, N., Keesling, A., and Wang, S. (2022). Industry applications of neutral-atom quantum computing solving independent set problems. *arXiv preprint arXiv:2205.08500*.

[Xanadu, 2024] Xanadu (2024). PennyLane QAOA cost Hamiltonians. https://docs.pennylane.ai/en/stable/_modules/pennylane/qaoa/cost.html#min_vertex_cover.

[Xanadu, 2025] Xanadu (2025). Strawberry Fields. strawberryfields.ai/.

[Xiao and Nagamochi, 2017] Xiao, M. and Nagamochi, H. (2017). Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146.

[Yannakakis and Gavril, 1980] Yannakakis, M. and Gavril, F. (1980). Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372.

[Zielinski et al., 2024] Zielinski, S., Nüßlein, J., Kölle, M., Gabor, T., Linnhoff-Popien, C., and Feld, S. (2024). Solving max-3sat using qubo approximation. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pages 681–691.

[Zuckerman, 2006] Zuckerman, D. (2006). Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 681–690.

Part V

Appendices

Appendix A

Classical Pre-processing techniques and Exact Reference Solutions for Vertex Cover

We discuss a few classical pre-processing techniques for the problem MINIMUM VERTEX COVER and its parameterized decision version k-VERTEX COVER. These techniques have been used in the algorithms to compute exact reference solutions for the problem MINIMUM VERTEX COVER and have also served to compute the exact reference solutions for the problems MAXIMUM INDEPENDENT SET and MAXIMUM CLIQUE.

A.1. MINIMUM VERTEX COVER and k-VERTEX COVER

The optimization problem that asks to find the MINIMUM VERTEX COVER is a vertex cover of the smallest size and the parameterized decision version asks whether there exists a vertex cover of size at most k. Since k-Vertex Cover is fixed-parameter tractable, we can leverage powerful preprocessing techniques to reduce the problem size while preserving solution properties. This reduction in problem size is particularly valuable as it allows us to handle larger instances by first applying polynomial-time kernelization before using more computationally intensive solution methods.

A.2. Reduction rules for k-VERTEX COVER

The problem MINIMUM VERTEX COVER is NP-hard [Garey and Johnson, 1979] and the parameterized decision version k-VERTEX COVER is fixed-parameter tractable [Downey et al., 1999b]. Using a bounded search tree, the problem can be solved in time $O(2^k n)$, where n is the number of vertices in the graph. However, this running time is not practical for graphs with a large value of k. Therefore, we can use pre-processing techniques to reduce the size of the input graph before applying the algorithm.

We classify some classical pre-processing rules from the literature (also called reduction rules) to be *general* or to be *specific* to the parameter k (i.e., the size of the vertex cover

to be determined). The following are a few reduction rules for the k-VERTEX COVER problem [Downey et al., 1999b]. The kinds of vertices used in the reduction rules are illustrated in Fig. A.1.

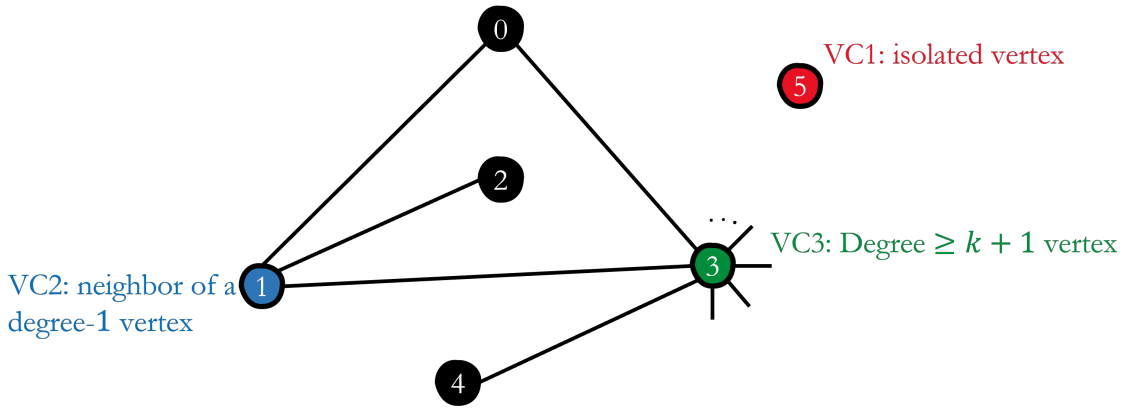


Figure A.1.|: The different kinds of vertices used in the reduction rules **VC1**, **VC2** and **VC3**

- **VC1:***(General) Removal of all isolated vertices.* This reduction rule relates to isolated vertices. Vertices with no adjacent edges can be removed from the graph thereby reducing the input size.
- **VC2:***(General) Neighbouring vertices of degree-1 vertices.* We can add the neighbouring vertices of degree-1 vertices into the vertex cover and remove this edge, and its incident edges from the graph.
- **VC3:***(Specific) Vertices with a degree at least* $k+1$. We can add a vertex of degree $k+1$ to the vertex cover. By not adding this vertex, we would need to add all of the $k+1$ neighbours to the vertex cover which is not possible since we are looking for a vertex cover of size at most k.

Note that above rules can be implemented in the time $O(nk)$.

Kernelization based on global properties of the vertex cover

The first three rules explore *local* substructures, one could also exploit kernelizations based on *global* properties [Nemhauser and Trotter, 1975] leading to kernelizations based on matching. Another example of using arbitrarily large substructures is *crown reductions* which is a generalization of **VC2**. A *crown* consists of an independent set I (no two vertices in I are connected by an edge) and another set H containing vertices adjacent to vertices in I as illustrated in Fig. A.2. For $I \cup H$ to be a crown, there has to exist a size-$|H|$ maximum matching (i.e., every vertex of H is matched) in the graph induced by the edges between H and I. If such a crown exists, then we need at least $|H|$ vertices to cover all edges in the crown. Since I is an independent set, there is a minimum size vertex cover for the graph that contains all the vertices of H and none of I. Therefore, any given crown can be deleted reducing the parameter k by $|H|$. Chor et al. [Chor et al., 2004] give a polynomial-time kernelization with a running time of $O(k(n+m))$, where n is the number of vertices and m

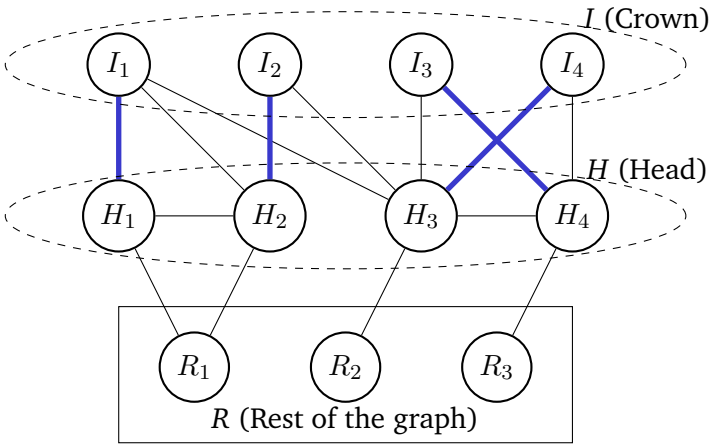


Figure A.2.|: A graph G with a crown $I \cup H$.

is the number of edges for finding crowns by computing maximum matchings. This yields another general reduction rule.

- **VC4:** *General* If G has a crown structure $I \cup H$, then delete H and reduce k by $|H|$.

A.3. Linear Programming Kernelization for k-VERTEX COVER

The optimization version of Vertex Cover can be formulated as an integer program [Abu-Khzam et al., 2005]:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{u \in V} x_u \\ \text{Subject to:} \quad & x_u + x_v \geq 1 \quad \forall \{u, v\} \in E \\ & x_u \in \{0, 1\} \quad \forall u \in V \end{aligned}$$

where $x_u = 1$ indicates vertex u is in the cover. Relaxing this to a linear program by replacing $x_u \in \{0, 1\}$ with $x_u \geq 0$ provides a lower bound on the optimal vertex cover size [Nemhauser and Trotter, 1975].

Given the LP solution, we partition vertices into three sets:

$$\begin{aligned} P &= \{u \in V \mid x_u > 0.5\} \\ Q &= \{u \in V \mid x_u = 0.5\} \\ R &= \{u \in V \mid x_u < 0.5\} \end{aligned}$$

A modification [Abu-Khzam et al., 2005] of the Nemhauser-Trotter theorem [Nemhauser and Trotter, 1975] guarantees that there exists an optimal vertex cover

that contains all vertices in P and no vertices in R. This insight enables a problem reduction by allowing us to fix certain vertices as either included or excluded from the final solution.

A.4. Exact Reference Solutions for MINIMUM VERTEX COVER

The exact reference solutions use the pre-processing techniques described above to reduce the size of the input graph before applying a branching algorithm. The `VertexCoverSolver` class implements the complete solution pipeline for solving the MINIMUM VERTEX COVER problem. The class is initialized with an input graph:

Listing A.1: Vertex Cover Solver Initialization

```
class VertexCoverSolver:
    def __init__(self, graph):
        self.graph = graph
```

General Pre-processing

The following code snippets demonstrate the general pre-processing techniques that can be applied to the input graph.

Listing A.2: General Pre-processing for MINVC

```
def remove_isolated_nodes(self, graph):
    """
    Removes all isolated nodes from the graph.
    """
    isolated_nodes = [node for node in graph.nodes() if graph.degree(node) == 0]
    graph.remove_nodes_from(isolated_nodes)
    return graph
def apply_pendant_rule(self, graph):
    """
    Handles pendant vertices (degree-1 vertices)by adding their neighbors to the
    """
    solution_vertices = []
    modified_graph = graph.copy()

    # Get all degree-1 vertices
    pendant_vertices = [v for v in graph.nodes() if graph.degree(v) == 1]

    for v in pendant_vertices:
        neighbor = list(graph.neighbors(v))[0]
        if neighbor not in solution_vertices:
```

```
            solution_vertices.append(neighbor)
        modified_graph.remove_nodes_from([v, neighbor])

    return modified_graph, solution_vertices
```

LP Kernelization

The following snippet is used to apply the LP Kernelization technique to the input graph. The function `lp_kernelization` takes a graph as input and returns the reduced graph along with the vertex cover size.

Listing A.3: LP Kernelization for MINVC

```
def lp_kernelization(self, graph):
    """
    Credits: Faisal Abu-Khzam
    Implement the LP kernelization technique.
    Returns the kernel graph and the vertices in the cover so far.
    """
    # Create a Pulp ILP problem
    prob = pl.LpProblem("VertexCover", pl.LpMinimize)
    x = pl.LpVariable.dicts("x", graph.nodes(), lowBound =0 ,cat='Continuous'
    prob += pl.lpSum(x)

    for (u,v) in self.graph.edges():
        prob += x[u] + x[v] >= 1

    solver = pl.getSolver('PULP_CBC_CMD', threads = 1)
    start = time.time()
    prob.solve(solver)
    end = time.time()
    execution_time = end - start

    kernelgraph = graph.copy()
    solutionpart = []

    for v in prob.variables():
        index = re.findall(r'\d+', v.name)
        s = [str(i) for i in index]
        res = int("".join(s))
        if v.varValue!=0.5:
                kernelgraph.remove_node(res)
        if v.varValue>0.5:
            solutionpart.append(res)
```

```
    return kernelgraph, solutionpart, execution_time
```

Bounded Search Tree

The following code snippet implements the bounded search tree algorithm for solving the MINIMUM VERTEX COVER problem.

Listing A.4: Bounded Search Tree for MINVC

```
def bounded_search_tree_VC(self, kernelgraph, k, current):
        # Base cases
        if not kernelgraph.edges():
            return True, current
        elif k == 0:
            return False, []

        u = max(kernelgraph.nodes(), key=lambda x: kernelgraph.degree(x))
        neighbors = list(kernelgraph.neighbors(u))
        # Recurse on two cases: (1) Remove u from the
        # graph, (2) Remove v from the graph
        graph_minus_u = kernelgraph.copy()
        graph_minus_u.remove_node(u)
        current_u = current.copy()
        current_u.append(u)
        result_minus_u = self.bounded_search_tree_VC(graph_minus_u, k - 1, curre

        graph_minus_v = kernelgraph.copy()
        graph_minus_v.remove_nodes_from(neighbors)
        current_v = current.copy()

        if(k-len(neighbors)>=0):
            current_v.extend(neighbors)
            result_minus_v = self.bounded_search_tree_VC(graph_minus_v, k - len(
        else:
            result_minus_v = False, []
        if result_minus_u[0]:
            return result_minus_u
        elif result_minus_v[0]:
            return result_minus_v
        else:
            return False, []
```

Cost Calculation

We compute the cost of the vertex cover based on the penalties set in the graph. For profit covers, there is no penalty, and profit is calculated as the number of edges covered subtracted by the number of vertices in the cover. Note that we multiply the profit by -1 when we convert it into a cost (as the QAOA solver minimizes the cost function and we use this value for comparison).

Listing A.5: Cost Calculation for MINVC

```
def compute_profit(self, cover):
    gain=0
    for edge in self.graph.edges():
        if (edge[0] in cover or edge[1] in cover):
            gain +=1

    loss = len(cover)
    profit = gain - loss
    if (profit<0):
        profit=0
    return profit
```

Appendix B

Impementation Details: PennyLane

B.1. Hamiltonian Construction

The PennyLane library provides a convenient way to construct Hamiltonians using the `qml.Hamiltonian` class. This class allows us to define Hamiltonians in terms of Pauli operators and their coefficients, which is essential for implementing quantum algorithms like QAOA. A `Hamiltonian` class implements various quantum Hamiltonians for graph optimization problems using PennyLane's quantum computing framework. It provides a unified interface for constructing both cost and mixer Hamiltonians for QAOA. The following listing shows the implementation of the Hamiltonian class for the problem MAXPC. Other QUBO formulations can be implemented similarly by defining the appropriate cost Hamiltonians.

Listing B.1: Hamiltonian Construction in PennyLane

```
class Hamiltonian:
    def __init__(self, graph):
        self.graph = graph

    def profit_cover_cost_Hamiltonian(self):
        obs = []
        coeffs = []

        for edge in self.graph.edges():
            coeffs.extend([0.25, 0.25, 0.25])
            obs.extend([qml.PauliZ(edge[0]), qml.PauliZ(edge[1]),
                        qml.PauliZ(edge[0]) @ qml.PauliZ(edge[1])])
        hamiltonian_edge = qml.Hamiltonian(coeffs, obs)

        obs = []
        coeffs = []
        for vertex in self.graph.nodes():
            coeffs.extend([-0.5])
            obs.extend([qml.PauliZ(vertex)])
```

```
        hamiltonian_vertex = qml.Hamiltonian(coeffs, obs)
        profit_cover = hamiltonian_edge + hamiltonian_vertex
        return profit_cover
```

For SCOOP problems with higher-order terms, the Hamiltonian construction is similar.

Listing B.2: Hamiltonian Construction for MaxPD

```
def profit_domination_cost_Hamiltonian(self):
    gain = qml.Hamiltonian([0], [qml.Identity(0)])
    for i in self.graph.nodes():
        hamiltonian_i = qml.Hamiltonian([1], [qml.Identity(i)])
                        - qml.Hamiltonian([0.5, -0.5],
                        [qml.Identity(i), qml.PauliZ(i)])
        for j in self.graph.neighbors(i):
            obs = []
            coeffs = []
            coeffs.extend([0.5, -0.5])
            obs.extend([qml.Identity(j),
                    qml.PauliZ(j)])
            hamiltonian_n = qml.Hamiltonian([1], [qml.Identity(j)])
                            - qml.Hamiltonian(coeffs, obs)
            hamiltonian_i = hamiltonian_i@hamiltonian_n

        gain += (qml.Hamiltonian([1], [qml.Identity(i)]) - hamiltonian_i)

    obs = []
    coeffs = []
    for i in self.graph.nodes():
        coeffs.extend([0.5, -0.5])
        obs.extend([qml.Identity(i),
                    qml.PauliZ(i)])

    loss = qml.Hamiltonian(coeffs, obs)
    profit = gain - loss
    cost_hamiltonian = (-1)*profit
    return cost_hamiltonian
```

For constructing the Hamiltonians for constrained problems, penalties values are input arguments as well:

Listing B.3: Hamiltonian Construction for MinVC

```
def vertex_cover_penalty(self, penalty_A=3, penalty_B=2):
    obs = []
```

```
    coeffs = []

    for edge in self.graph.edges():
        coeffs.extend([penalty_A/4,penalty_A/4, penalty_A/4])
        obs.extend([qml.PauliZ(edge[0]), qml.PauliZ(edge[1]),
                    qml.PauliZ(edge[0]) @ qml.PauliZ(edge[1])])
    hamiltonian_edge = qml.Hamiltonian(coeffs, obs)

    obs = []
    coeffs = []
    for vertex in self.graph.nodes():
        coeffs.extend([-1*penalty_B/2])
        obs.extend([qml.PauliZ(vertex)])

    hamiltonian_vertex = qml.Hamiltonian(coeffs, obs)
    cost_hamiltonian = hamiltonian_edge + hamiltonian_vertex
    return cost_hamiltonian
```

B.2. QAOA Implementation

The QAOASolver class implements the complete QAOA workflow, handling both circuit construction and optimization. It takes cost and mixer Hamiltonians as inputs and manages:

- Circuit Construction: Implements p-layer QAOA with alternating cost and mixer operations
- Parameter Optimization: Uses RMSProp optimizer over 400 iterations
- Measurement: Returns both state probabilities and expectation values
- Initial State Preparation: Applies Hadamard gates to create superposition

The solver automatically handles quantum device configuration and provides methods for both optimization and final state measurement. We implement the QAOA algorithm using PennyLane's `qaoa` function, which allows us to specify the cost Hamiltonian and the mixer Hamiltonian. PennyLane provides methods for layering quantum operations, which is essential for implementing the QAOA algorithm. For QAOA, PennyLane provides QAOA cost layer and QAOA mixer layer functions to construct the cost and mixer layers of the QAOA circuit. These functions allow us to define the quantum operations that correspond to the cost and mixer Hamiltonians, respectively. The `qml.layer` function is used to apply these layers in a structured manner, enabling the construction of multi-layer QAOA circuits. The cost layer applies the unitary operation $U_C = e^{-i\gamma H_C}$ where H_C is the cost Hamiltonian and γ is a variational parameter that determines the strength of the evolution under the cost Hamiltonian. Similarly, the mixer layer applies the unitary operation $U_M = e^{-i\beta H_M}$

where H_M is the mixer Hamiltonian and β is another variational parameter. The QAOA circuit is constructed by alternating these layers for a specified number of repetitions p, which determines the depth of the circuit.

The following code snippet demonstrates how to set up and run the QAOA algorithm for a given graph.

Listing B.4: QAOA Implementation in PennyLane

```
import networkx as nx
import pennylane as qml
from pennylane import numpy as np
from pennylane import qaoa
from .hamiltonian import Hamiltonian
dev = qml.device("default.qubit", wires=16)

class QAOASolver:
    def __init__(self, cost, mixer):
        self.cost = cost
        self.mixer = mixer

    def solve(self, init_params, p=1):

        def qaoa_layer(gamma, alpha):
            qaoa.cost_layer(gamma, self.cost)
            qaoa.mixer_layer(alpha, self.mixer)

        def circuit(params, **kwargs):
            wires = self.cost.wires
            for w in wires:
                qml.Hadamard(w)

            qml.layer(qaoa_layer, p, params[0], params[1])

        @qml.qnode(dev)
        def cost_function(params):
            circuit(params)
            return qml.expval(self.cost)

        optimizer = qml.RMSPropOptimizer()
        steps = 400
        params = init_params
        for _ in range(steps):
            params = optimizer.step(cost_function, params)

        @qml.qnode(dev)
```

```
        def penalty_probability_circuit(gamma, alpha):
            wires = self.cost.wires

            circuit([gamma, alpha])
            return qml.probs(wires=wires)

        probabilities = penalty_probability_circuit(params[0], params[1])
        expectation_value = cost_function(params)
        return probabilities, expectation_value, params
```

Generating Graphs

We use the NetworkX library to generate random graphs (varying density or 3 regular) for testing the QAOA algorithm. Note that all graphs generated are connected graphs. Note that these subroutines are also used for QTensor and Qiskit.

```
import networkx as nx
import matplotlib.pyplot as plt
import pickle

def generate_and_save_random_connected_graphs(num_graphs, num_nodes, edge_proba
    graphs = []

    for _ in range(num_graphs):
        # Generate a random graph
        random_graph = nx.fast_gnp_random_graph(num_nodes, edge_probability)

        # Ensure the graph is connected
        while not nx.is_connected(random_graph):
            random_graph = nx.fast_gnp_random_graph(num_nodes, edge_probability)

        graphs.append(random_graph)

    # Save the graphs to a pickle file
    with open(pickle_file, 'wb') as file:
        pickle.dump(graphs, file)

def generate_and_save_random_regular_graphs(num_graphs, num_nodes, degree, pickl
    graphs = []

    for _ in range(num_graphs):
        # Generate a random regular graph
        random_graph = nx.random_regular_graph(degree, num_nodes)
```

```
        graphs.append(random_graph)

    # Save the graphs to a pickle file
    with open(pickle_file, 'wb') as f:
        pickle.dump(graphs, f)
```

Collating Results

To collate results from multiple QAOA runs, we can use the following code snippet. This code reads the results from a pickle file and aggregates the probabilities and expectation values for each graph.

Listing B.5: Collating Results from QAOA Runs

```
import pickle

def solve_pc_from_pickle(pickle_file, num_layers, init_params):
    try:
        with open(pickle_file, 'rb') as file:
            graphs = pickle.load(file)
    except Exception as e:
        raise RuntimeError(f"Error loading pickle file: {e}")

    probs_all = []
    exp_val_all = []
    summed_probs_all = []

    for i, graph in enumerate(graphs):
        probs_graph = []
        exp_val_graph = []
        summed_probs_graph = []

        _, pc_cost = solve_pc_classical(graph)

        for layer in range(0, num_layers + 1):
            probs, exp_val, opt_params = solve_pc(graph, layer, init_params)
            summed_probs = get_summed_probs_pc(graph, probs, pc_cost)

            probs_graph.append(probs)
            exp_val_graph.append(exp_val)
            summed_probs_graph.append(summed_probs)

        probs_all.append(probs_graph)
        exp_val_all.append(exp_val_graph)
```

```
        summed_probs_all.append(summed_probs_graph)

    return {
        'probs': probs_all,
        'exp_vals': exp_val_all,
        'opt_params': opt_params_all,
        'summed_probs': summed_probs_all
    }
```

The code snippets presented here highlight key implementation aspects of the SCOOP framework; the complete implementation, including additional utilities, tests, and example notebooks, is available in our GitHub repository at https://github.com/RigiResearch/pf-opt.

Appendix C

Implementation Details: QTensor

C.1. Circuit Composer Class

The `CircuitComposer` class implements a hierarchical system for constructing QAOA circuits for graph optimization problems. It separates circuit construction from problem-specific implementations. A `CircuitBuilder` class defines a framework for constructing quantum circuits using a common interface (CircuitBuilder) that is specialized for different quantum simulation or computation backends such as QTensor, Qiskit, or Cirq. This design allows for flexibility in circuit construction while maintaining a consistent interface across different quantum computing platforms. The CircuitBuilder serves as an abstract base class that defines a common interface for quantum circuit construction across different backends. This interface includes fundamental operations such as reset, gate application, and circuit inversion. Backend-specific implementations (like 'QiskitBuilder' or 'CirqBuilder') inherit from this class, implementing these operations according to their respective platform requirements while maintaining a consistent interface. This is useful in hybrid quantum-classical workflows and in research code that needs to support multiple simulators or quantum SDKs. The `cone_ansatz` method implements a lightcone version of QAOA, which strategically constructs quantum circuits by focusing on dynamically defined subgraphs relevant to each QAOA layer. The circuit initialization applies a layer of Hadamard gates to all qubits to prepare a uniform superposition state. For each layer from p down to 1, the method creates a subgraph centered around the target edge or vertex, bounded by the distance corresponding to the current layer index. It then applies the cost operator using $\gamma[i]$ on this subgraph and reconstructs a (possibly updated) subgraph to apply the mixer operator with $\beta[i]$. This localized layering strategy is facilitated by `utils.get_edge_subgraph`, and the original graph structure is preserved and restored using `cone_base`. By confining quantum operations to lightcone-relevant regions, this ansatz significantly reduces circuit depth and qubit connectivity requirements.

Figure C.1 illustrates the overall architecture of the CircuitComposer framework, highlighting the interactions between different components and the flow of information during circuit construction. The CircuitComposer, QAOAComposer, and ZZComposer are provided by QTensor. Our contribution is the addition of constrained problems.

Below we show a sample code snippet that illustrates how the CircuitComposer is used to construct a QAOA circuit for a given problem:

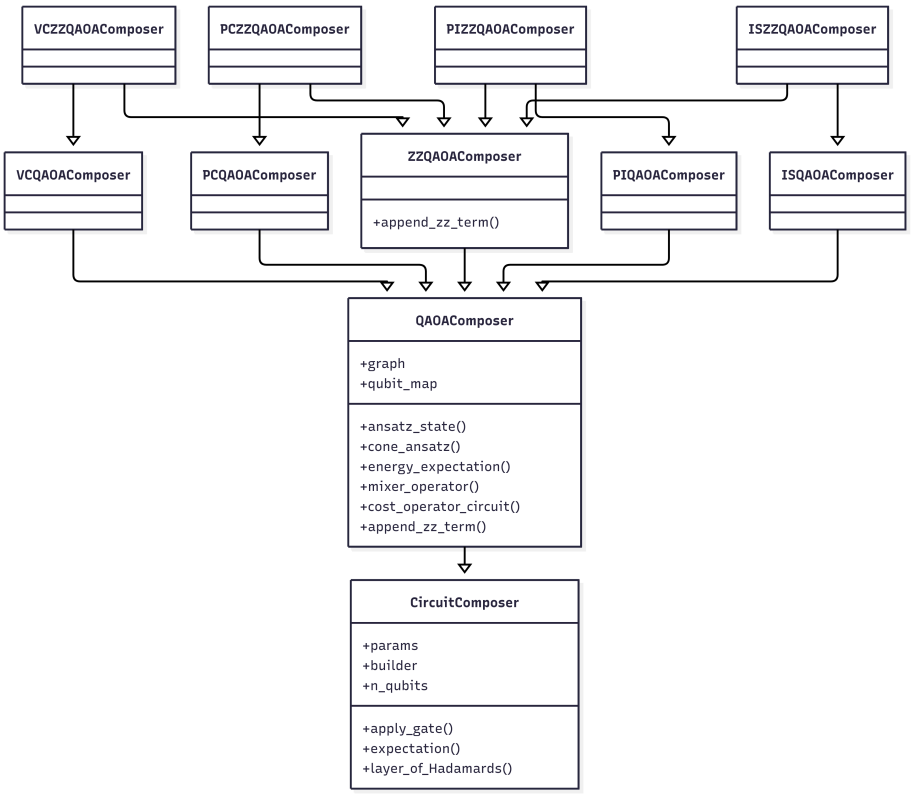


Figure C.1.|: Architecture of the CircuitComposer framework. The CircuitComposer, QAOAComposer, and ZZComposer are provided by QTensor. We extend the framework to support constrained problems by adding both the constrained problem and its SCOOP variant. The problems shown here are Vertex Cover (VC), Independent Set (IS), and its SCOOP variants, Profit Cover (PC), and Profit Independence (PI).

Listing C.1: Using CircuitComposer to construct a QAOA circuit

```
class PCQAOAComposer(QAOAComposer):

    def energy_expectation(self, *args):
        self.cone_ansatz(*args)
        self.energy_term(*args)
        first_part = self.builder.circuit
        self.builder.reset()

        self.cone_ansatz(*args)
        self.builder.inverse()
        second_part = self.builder.circuit
        self.circuit = first_part.compose(second_part)


    def cost_operator_circuit(self, gamma, edges=None):
        if edges is None: edges = self.graph.edges()
        nodes = self.graph.nodes()

        for i, j in edges:
            u, v = self.qubit_map[i], self.qubit_map[j]
            self.append_zz_term(u, v, gamma*(0.25))

        for i in nodes:
            u = self.qubit_map[i]
            self.z_term(u, gamma * (-0.25)*(self.graph.degree(i) -2))

class VCQAOAComposer(QAOAComposer):
    def energy_expectation(self, *args):
        self.cone_ansatz(*args)
        self.energy_term(*args)
        first_part = self.builder.circuit
        self.builder.reset()

        self.cone_ansatz(*args)
        self.builder.inverse()
        second_part = self.builder.circuit
        self.circuit = first_part.compose(second_part)

    def cost_operator_circuit(self, gamma, edges=None):
        if edges is None: edges = self.graph.edges()
        nodes = self.graph.nodes()
        for i, j in edges:
            u, v = self.qubit_map[i], self.qubit_map[j]
```

```
            self.append_zz_term(u, v, gamma*(0.75))


        for i in nodes:
            u = self.qubit_map[i]
            self.z_term(u, gamma*(0.25)*((3*self.graph.degree(i))-4))
class ZZQAOAComposer(QAOAComposer):
    def append_zz_term(self, q1, q2, gamma):
        self.apply_gate(self.operators.ZZ, q1, q2, alpha=2*gamma)

class VCZZQAOAComposer(VCQAOAComposer, ZZQAOAComposer):
    def energy_expectation(self, *args):
        self.cone_ansatz(*args)
        self.energy_term(*args)
        first_part = self.builder.circuit
        self.builder.reset()
        self.cone_ansatz(*args)
        self.builder.inverse()
        second_part = self.builder.circuit
        self.circuit = first_part + second_part

class PCZZQAOAComposer(PCQAOAComposer, ZZQAOAComposer):
    def energy_expectation(self, *args):
        self.cone_ansatz(*args)
        self.energy_term(*args)
        first_part = self.builder.circuit
        self.builder.reset()
        self.cone_ansatz(*args)
        self.builder.inverse()
        second_part = self.builder.circuit
        self.circuit = first_part + second_part
```

C.2. QAOASimulator Class

The `QAOASimulator` class is responsible for simulating the QAOA circuits constructed by the `CircuitComposer`. It provides methods for running the circuits on classical simulators and obtaining results that can be compared with quantum hardware executions. The class is designed to be backend-agnostic, allowing it to work with various quantum simulation frameworks. Tensor network simulation is done using the QTensor-QTree backend. Figure C.2 shows the architecture of the QAOASimulator class, which includes methods for circuit execution, result retrieval, and performance evaluation.

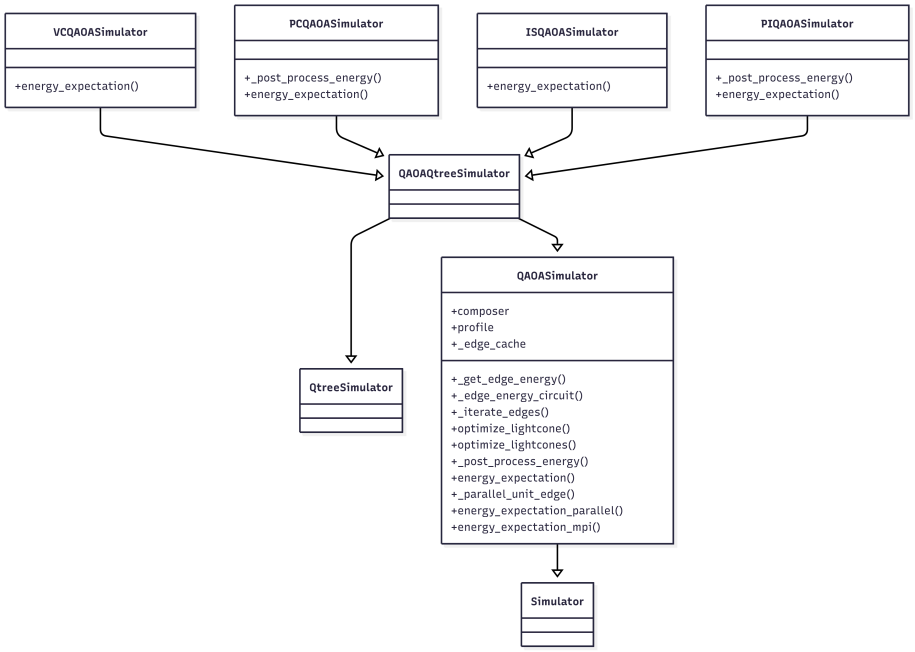


Figure C.2.|: Architecture of the QAOASimulator class. The class is responsible for simulating QAOA circuits constructed by the CircuitComposer. We inherit from the QAOASimulator and QAOAQtreeSimulator classes to implement problem specific simulations for various constrained problems and their SCOOP twins. The problems shown here are Vertex Cover (VC), Independent Set (IS), and its SCOOP variants, Profit Cover (PC), and Profit Independence (PI).

Below we show a sample code snippet that illustrates how the QAOASimulator is used to run a QAOA circuit for a given problem:

Listing C.2: Using QAOASimulator to run a QAOA circuit

```
class VCQAOASimulator(QAOAQtreeSimulator):
    def energy_expectation(self, G, gamma, beta):
        """
        Arguments:
            G: VertexCover graph, Networkx
            gamma, beta: list[float]
        Returns: VertexCover energy expectation
        """
        total_E = 0
        for i, edge in enumerate(G.edges()):
            E = self._get_edge_energy(G, gamma, beta, edge)
            total_E += E
        if self.profile:
            print(self.backend.gen_report())
        return np.real(total_E)

class PCQAOASimulator(QAOAQtreeSimulator):
    def _post_process_energy(self, G, E):
        if (np.abs(np.imag(E))>1e-6).any():

            print(f"Warning: Energy result imaginary part was: {np.imag(E)}")
        """
        Calculate final energy of Profit Cover by adding the offsets
        """
        E = np.real(E)

        Ed = G.number_of_edges()
        V = G.number_of_nodes()

        return E + V/2 - 3*Ed/4

    def energy_expectation(self, G, gamma, beta):
        """
        Arguments:
            G: ProfitCover graph, Networkx
            gamma, beta: list[float]
        Returns: ProfitCover energy expectation
        """
        total_E = 0
        for i, edge in enumerate(G.edges()):
```

```
            E = self._get_edge_energy(G, gamma, beta, edge)
            total_E += E
        if self.profile:

            print(self.backend.gen_report())
        C = self._post_process_energy(G, total_E)
        return C
```

Note that we do not include any imports or class definitions that are used in the code snippets above, as they are assumed to be part of the QTensor library. The code snippets illustrate how to use the CircuitComposer and QAOASimulator classes to construct and simulate QAOA circuits for constrained problems. Code related to QTensor can be found in https://github.com/danlkv/QTensor.

Appendix D

Implementation Details: Qiskit 2.0

D.1. Hamiltonian Representation using `SparsePauliOp`

SparsePauliOp is a Qiskit class that provides an efficient representation of quantum operators in the Pauli basis, particularly useful for large qubit systems with sparse interactions. The primary advantage of `SparsePauliOp` lies in its sparse representation of quantum operators. By storing only non-zero Pauli terms, it significantly reduces memory requirements for N-qubit systems. The class enables practical implementation of large-scale optimization problems involving hundreds of qubits, making it essential for our utility-scale experiments.

```
SparsePauliOp(
    paulis: PauliList,
    coeffs: np.ndarray
)
```

The SparsePauliOp requires two arguments:

- **paulis:** A PauliList object containing Pauli strings (e.g., $IXYZ$).
- **coeffs:** A complex numpy array of coefficients for each Pauli term.

D.1.1. MinVC and MaxPC Hamiltonian Construction

The `build_mvc_paulis` function constructs the quantum Hamiltonian for the Minimum Vertex Cover problem. This implementation translates graph structure into Pauli operators suitable for QAOA execution.

Formulated using the Pauli-Z operators, the Hamiltonian $\hat{H}_{\text{VC}}$ can be expressed as:

$$\hat{H}_{\text{VC}}^{\text{min}} = \frac{A}{4} \sum_{uv \in E} (\hat{Z}_u \hat{Z}_v + \hat{Z}_u + \hat{Z}_v) - \frac{B}{2} \sum_{v \in V} \hat{Z}_v + \Delta_{\text{VC}} \tag{D.1}$$

Note that we multiply both coefficients by 0.5 so as to scale down the Hamiltonian for the application of fractional gates. $\Delta_{\text{VC}} = \frac{1}{4}A|E| + \frac{1}{2}B|V|$, and penalties A and B in $\hat{H}_{\text{VC}}^{\text{min}}$

are set to 3 and 2. Note that $\frac{\hat{H}_{\text{VC}}}{B}$ corresponds to the size of the solution state for (only) the cases that correspond to a feasible vertex cover solution.

Edge Terms:

Each edge (i, j), generates three terms:

- ZZ interaction between vertices i and j
- Single Z operator on vertex i
- Single Z operator on vertex j

Vertex Terms:

Each vertex v generates:

- Single Z operator on vertex v

Listing D.1: Building the Minimum Vertex Cover Hamiltonian using SparsePauliOp

```
from qiskit.quantum_info import SparsePauliOp
def build_mvc_paulis(graph: nx.graph) -> list[tuple[str, float]]:
    pauli_list = []
    for edge in list(graph.edges()):
        paulis = ["I"] * len(graph)

        paulis[edge[0]], paulis[edge[1]] = "Z", "Z"

        coeff = 0.75*0.5

        pauli_list.append(("".join(paulis)[::-1], coeff))

        paulis = ["I"] * len(graph)
        paulis[edge[0]] = "Z"
        pauli_list.append(("".join(paulis)[::-1], coeff))
        paulis = ["I"] * len(graph)
        paulis[edge[1]] = "Z"
        pauli_list.append(("".join(paulis)[::-1], coeff))

    for vertex in list(graph.nodes()):
        paulis = ["I"] * len(graph)
        paulis[vertex] = "Z"
        coeff = -1.0*0.5
        pauli_list.append(("".join(paulis)[::-1], coeff))

    return pauli_list
```

```
mvc_paulis = build_mvc_paulis(graph)

cost_hamiltonian_mvc = SparsePauliOp.from_list(mvc_paulis)
```

Listing D.2: Building the Maximum Profit Cover Hamiltonian using SparsePauliOp

```
def build_mpc_paulis(graph: nx.graph) -> list[tuple[str, float]]:
"""Convert the graph to Pauli list.
"""
pauli_list = []
for edge in list(graph.edges()):
    paulis = ["I"] * len(graph)

    paulis[edge[0]], paulis[edge[1]] = "Z", "Z"

    coeff = 0.25

    pauli_list.append(("".join(paulis)[::-1], coeff))

    paulis = ["I"] * len(graph)
    paulis[edge[0]] = "Z"
    pauli_list.append(("".join(paulis)[::-1], coeff))
    paulis = ["I"] * len(graph)
    paulis[edge[1]] = "Z"
    pauli_list.append(("".join(paulis)[::-1], coeff))

for vertex in list(graph.nodes()):
    paulis = ["I"] * len(graph)
    paulis[vertex] = "Z"
    coeff = -0.5
    pauli_list.append(("".join(paulis)[::-1], coeff))

return pauli_list


mpc_paulis = build_mpc_paulis(graph)

cost_hamiltonian_maxpc = SparsePauliOp.from_list(mpc_paulis)
```

Note that other Hamiltonians can be constructed similarly, and are not shown here for brevity.

D.2. Circuit Compilation

Listing D.3: Circuit decomosition and transpilation

```
from qiskit.circuit.library import QAOAAnsatz
from qiskit.transpiler import generate_preset_pass_manager
from qiskit_ibm_runtime import EstimatorV2 as Estimator
circuit = QAOAAnsatz(cost_operator=cost_hamiltonian, reps=1)
circuit.measure_all()
circuit.decompose(reps=3)
 # QiskitRuntimeService.save_account(channel="ibm_quantum",
 #  token="<MY_IBM_QUANTUM_TOKEN>", overwrite=True, set_as_default=True)
service = QiskitRuntimeService(channel='ibm_quantum')
backend = service.backend('ibm_fez', use_fractional_gates=True)
print(backend)
pm = generate_preset_pass_manager(optimization_level=3,
                                  backend=backend)

candidate_circuit = pm.run(circuit)
```

D.3. Cost Function Estimator

Listing D.4: Using Estimator Primitive to calculate Cost

```
initial_gamma = np.pi/2
initial_beta = np.pi/4
init_params = [initial_gamma, initial_beta]
def cost_func_estimator(params, ansatz, hamiltonian, estimator):
    isa_hamiltonian = hamiltonian.apply_layout(ansatz.layout)
    pub = (ansatz, isa_hamiltonian, params)
    job = estimator.run([pub])
    results = job.result()[0]
    cost = results.data.evs
    objective_func_vals.append(cost)
    return cost
```

D.4. Running QAOA

Listing D.5: Run on IBM Hardware backend

```
from qiskit_ibm_runtime import Session, EstimatorV2 as Estimator
from scipy.optimize import minimize

with Session(backend=backend) as session:
    estimator = Estimator(mode=session)
```

```
estimator.options.default_shots = 1000

# Set simple error suppression/mitigation options
estimator.options.dynamical_decoupling.enable = True
estimator.options.dynamical_decoupling.sequence_type = "XY4"
# Cannot use twirling with fractional gates
estimator.options.twirling.enable_gates = False
# estimator.options.twirling.num_randomizations = "auto"

result = minimize(
    cost_func_estimator,
    init_params,
    args=(candidate_circuit, cost_hamiltonian, estimator),
    method="COBYLA",
    tol=1e-2,
)
```