Institut für Theoretische Physik
Gottfried Wilhelm Leibniz Universität Hannover

Enhancing Branch and Bound Techniques by Quantum Approximate Optimization

Master's Thesis

Paul Johann Christiansen

10019956

Supervisor:

Prof. Dr. Tobias J. Osborne

November 20, 2023

Abstract

On the hunt for a quantum advantage in the NISQ-dominated medium term, where the existing quantum resources are influenced by noise and decoherence, variational quantum algorithms (VQAs) [Cer+21] are generally considered the preferable strategy, as leveraging classical computation power to optimize the output of the underlying parametrized quantum circuit accounts for a limited number of qubits. However, for NP-hard combinatorial optimization problems like the Knapsack Problem (KP) [MT90; KPP04] where exactly solving instances of up to 100,000 variables is a matter of seconds for state-of-the-art classical algorithms [MT90, S.2.10], achieving an improvement with a VQA may still be unrealistic in the next years. One of the most promising ways of creating industrial relevance for quantum computing in the near future might instead be to enhance subroutines with the aid of VQAs instead of fully replacing well-established classical algorithms, as this allows to hold on to tried and tested methods and to make the number of employed qubits adjustable to the available hardware. This thesis showcases a proof of concept for that. More precisely, a classical Branch and Bound (B&B) [LD60; Cla99] is set up for the Knapsack Problem, which asks to find the optimal selection of items to fill in a knapsack, featuring the largest possible aggregated value that does not exceed a certain capacity given from the outset. This classical framework is extended by introducing an alternative (quantum) lower bound in the B&B via a Quantum Alternating Operator Ansatz (QAOA) [Had18] following the Grover-mixer approach [BE20], in which the preparation of the initial state takes care of preserving feasibility. The result is a hybrid quantum-classical B&B algorithm (HQCBB). Due to its quantum-physical focus, classical components are in this work configured in a basic manner while the major portion of the expenditure goes into the design and the implementation of the QAOA. The state preparation in the Grover-mixer approach is here given by the Quantum Tree Generation (QTG), a method that was recently proposed as part of a new quantum algorithm for the Knapsack Problem [Wil+23]. The QTG generates a superposition of all feasible states in N steps for a given KP instance consisting of N elements; thereby, it admits for an exponential speedup compared to the classical analogue. In total, the resulting QAOA comes with a gate cost of $\mathcal{O}(N \log(W_{\max})^2)$ on $N + \lfloor \log W_{\max} \rfloor + 1$ qubits for a capacity $W_{\max}$. Inspired by Wilkening et al. [Wil+23], a high-level simulator is developed for the QAOA which enables to investigate problem instances with a qubit cost of 120 at largest in a reasonable amount of time. It circumvents the computationally expensive simulation of qubits and gate applications on a classical machine by performing the QTG in a classical fashion and evaluating analytical formulae to emulate the QAOA circuits. Simulations are conducted for both the new KP QAOA and the full HQCBB. The expected behavior of an approximation ratio that increases with the circuit depth can be confirmed to a good degree on randomly generated KP instances with different numbers of variables and different capacity vs. total weight ratios. On the other hand,

the QTG-induced QAOA is found to be capable of beating the Greedy heuristic as its classical opponent at various subproblem (qubit) sizes throughout the HQCBB run for every set of considered instances. The top values around a 60% QAOA-surplus can only rarely be observed, the majority of the classical threshold exceedings range below 30%. Finally, it is verified that the partially improved lower bounds can actually lead to a globally measurable effect in the form of a minor reduction in the number of explored nodes.

Contents

Introduction **1**

I. Theoretical Background **6**

1. Basic Concepts **7**
1.1. Complexity Theory 7
1.2. Combinatorial Optimization Problems 11
1.2.1. Notation 11
1.2.2. Quantum Mechanical Version 14
1.3. The Knapsack Problem 16

2. Algorithms **21**
2.1. Branch and Bound 21
2.1.1. Overview of the Algorithm 21
2.1.2. Relations Between Algorithm Components 24
2.1.3. Common Branching Strategies 26
2.1.4. Common Searching Strategies 28
2.2. QAOA 32
2.2.1. Quantum Adiabatic Evolution 32
2.2.2. Quantum Approximate Optimization Algorithm 35
2.2.3. Quantum Alternating Operator Ansatz 42

II. Algorithm Construction **51**

3. Core Idea **52**

4. Classical Part **55**
4.1. Branching Strategy 55
4.2. Bounds 56
4.2.1. Greedy Lower Bound 56
4.2.2. Greedy Upper Bound 57
4.3. Searching Strategy 60

4.4. Further Refinement and Full HQCBB . . . 61

5. Quantum Part . . . 65
5.1. Quantum Tree Generation . . . 65
5.2. Grover-Mixer QAOA . . . 68

III. Implementation and Simulation 70

6. QAOA Implementation . . . 71
6.1. State Preparation via QTG . . . 71
6.1.1. Digital Comparator . . . 71
6.1.2. Subtraction via Quantum Fourier Transform . . . 74
6.1.3. Full Circuit . . . 80
6.2. Grover Mixing and Phase Separation Unitaries . . . 82
6.3. Classical Angle Optimization . . . 86

7. Simulation . . . 87
7.1. High-level Simulator . . . 88
7.2. Grover-Mixer QTG QAOA . . . 91
7.2.1. Solution Probabilities . . . 91
7.2.2. Approximation Ratios . . . 96
7.3. Full HQCBB . . . 102
7.3.1. Lower Bound Comparisons . . . 102
7.3.2. Number of Explored Nodes . . . 111

Conclusions & Outlook . . . 117

Appendix 124
A. Dynamic Programming . . . 125
B. Nelder-Mead Method . . . 127
C. Quantum Circuit Notation . . . 129

Bibliography . . . 133

Declaration of Ownership . . . 140

Acknowledgements . . . 141

Introduction

By now, one can safely attest quantum computing to having emerged as one of the most hyped trends at the interface of basic research and industry. People working in the twilight zone between quantum physics and computer science find themselves confronted with high hopes of business partners across all industry sectors - a development that has increased rapidly in recent years: Between 2015 and 2020, the venture capital invested in related research is estimated to have grown by an incredible 500% [RMO22]. This year, the German federal government decided on a funding of about €3 billion on quantum technologies together with scientific organizations [Bil23; Qua23]. Not long ago, neither the physics nor the computer science department seemed to feel comfortable with devoting significant resources to quantum information research [ZJ22].

Quantum computing came up in the 1980's, mainly driven by the edge-cutting works of Benioff [Ben80] and Deutsch [Deu85]. The famous conjecture of Feynman [Fey82], made when the field was still in its very infancy, that a quantum computer should, in principle, be able to simulate any quantum system, was later proven by Lloyd [Llo96]. However, using a quantum system to simulate other quantum systems did not stay the only intention for long - even before that proof was published, simple abstract examples were constructed artificially to justify the new approach by demonstrating that there are indeed cases in which a quantum computer can do better; most prominently the Deutsch-Josza algorithm [DJ92] that achieves an exponential speedup when trying to figure out whether a black-box function on an even number of N bits is either constant (all inputs are mapped to the same value 0 or 1) or equally balanced (half mapped to 0, half mapped to 1) while it is guaranteed that one of these options is true. The algorithms developed by Shor [Sho94] and Grover [Gro96] address more lifelike problems, namely finding the prime factorization of an integer and searching for a marked element in a list, while achieving superpolynomial and quadratic speedups compared to the best classical methods, respectively. The former even provides a threat to commonly used public-key cryptography systems like the RSA, which is exactly built on the conviction that such a factoring becomes intractable for very large integers [RSA78]. However, the number of qubits needed to speak of a non-negligible impact on the RSA is in the seven-digit magnitude [GE21] and therefore far beyond the reach of any currently available resources. On the other hand, Grover [Gro96] himself describes his algorithm to find the most application in database searches; furthermore,

it can be used in numerous other quantum algorithms to induce similar speedups [Gro98; Amb04]. For being of potential relevance to the industry, Shor's and Grover's publications served as catalysts for the general interest in the then still young field. From that point onward, the number of proposed and invented quantum algorithms has really exploded - currently, the *Quantum Algorithm Zoo* [Jor22] counts over 60 algorithms featuring different time savings and a wide range of applications, as well as 435 related papers; an introduction to and an overview of particularly influential ones clustered in categories was provided by Montanaro [Mon15]. All of them are intended to approach the overarching goal of achieving an advantage over the best classical supercomputers, also known as *quantum advantage*, which is usually understood as an improvement in terms of time or space on real-world applications. Transforming fundamental research into practical use cases is in fact always a big deal, not only here - with gallows humor, it is therefore sometimes also called the *valley of death* [ZJ22].

One main reason for why a significant part of the scientific community used to consider quantum computers as mere science fiction [ZJ22] is that no quantum system is fully free of decoherence [Sch05], imposing a major hurdle for executing large circuit depths[1] and building functional quantum computers in general [Lad+10].[2] The aim of mitigating the effects of decoherence and other types of noise that can appear in quantum computers - like faulty states, faulty gates or faulty measurements - are combined under the umbrella of *Quantum Error Correction (QEC)*, which introduces the concept of *logical* and *physical* qubits. Shor [Sho95] was the first to demonstrate how a quantum error correction code can be set up in that regard via distributing the information held by one logical qubit to nine strongly entangled physical qubits. The presence of decoherence and noise hence enlarges the qubit requirements of working quantum algorithms further and thereby pushes the fault-tolerant era several years or even decades to the future. To give a rough insight in current hardware situation, it should be mentioned that the first cloud-based quantum computer was made publicly accessible in 2016 by IBM [Man21]. They are, as of 2023, the undisputed market leader in the provision of qubit processors: Only one year after the magic barrier of 100 qubits got exceeded by their Eagle chip [Dia22], IBM released the 433-qubit Osprey processor in 2022 [CN22]. On their "road to advantage" [IBM23], the next milestone is already about to be passed - by the end of the year, Condor will boost the limit of what can be done to significantly above 1,000 qubits. With switching to multiple chips and the aid of quantum communication technologies until 2026, this roadmap envisages the way to be paved for 10,000 - 100,000 qubits [IBM23]. Starting from 2022, investigations have also been made to reduce circuit depths and work towards error correction [Joh22]. Actually mitigating noise shall however not be incorporated until next year [IBM23].

[1] The *depth* of a quantum circuit denotes the maximum length between the input and the output in terms of gate layers executed in parallel.

[2] In a quantum computer, the manipulation of states in the form of operations and measurements prevents a perfect isolation that would be necessary to avoid decoherence.

Despite first QEC efforts, the current *noisy intermediate scale quantum (NISQ)* devices - a term that can be traced back to Preskill [Pre18] - are believed to dominate the coming years due to the additional qubit needs in fault-tolerant implementations as emphasized above. While *quantum supremacy* - denoting the ability of quantum computers to outperform classical state-of-the-art supercomputers on abstract mathematical tasks (with potentially little physical meaning) - could already be confirmed with the available hardware [Boi+18; Nei+18; Aru+19], similar evidence for a quantum advantage is still pending. The central question thus is now whether and how this could already be achieved in the medium term with the underlying NISQ resources. Any approach in that regard must, in particular, cope with a bounded number of qubits as well as a restricted gate fidelity due to inevitable noise and decoherence errors, limiting the possible circuit depths. The most promising candidate is the class of *variational quantum algorithms (VQAs)*, denoting algorithms where a classical method is employed to take care of optimizing the output of a parametrized quantum circuit. Leveraging classical computation power arguably makes VQAs hybrid quantum-classical algorithms. In their general review, Cerezo et al. [Cer+21] present the building blocks of VQAs, provide an overview of their numerous applications - containing i.a. the search for ground states of quantum systems, dynamical simulations, error correction, compilation, machine learning, combinatorial optimization and mathematical applications - and discuss current challenges like accuracy and efficiency. Therein, they understand VQAs as "[...] the quantum analogue of highly successful machine-learning methods, such as neural networks" [Cer+21, p.625]. What makes the VQA-approach the leading strategy for the NISQ period is that outsourcing the parameter optimization by drawing on existing classical resources helps in capping the number of required qubits.

One of the VQA flagships is the *Quantum Approximate Optimization Algorithm (QAOA)* proposed by Farhi, Goldstone, and Gutmann [FGG14], that essentially consists of a quasi-adiabatic evolution composed of alternating applications of two different types of parametrized unitaries. As the name suggests, the QAOA is a quantum algorithm that approximates the solution to a given problem which is suitably encoded. The hope for a medium-term quantum advantage made the QAOA research prosper; by now, there exists a plethora of related literature. Some of the most important results are that it admits universal quantum computation [Llo18; MBZ19], an asymptotic analyis [Koß+22], the conducted investigations in strategies for actual implementations on NISQ devices together with their associated performances [Zho+18; Qia+18; Pag+20; Har+21], or exemplary estimations on the qubit requirements for a quantum speedup on the basis of reasonable noise models [GM19]. Of special interest is also the generalized version, called *Quantum Alternating Operator Ansatz (QAOA)* [Had18], that allows to respect the constrained nature of problems properly.

On the hunt for a quantum advantage, a certain class of problems among the various VQA applications has moved to the center of attention, accounting for their industrial

relevance and the intractability on classical computers: NP-hard *combinatorial optimization problems*. Any such problem asks to optimize a Boolean function on many input variables, often with respect to some constraints. The interest in them is caused by the widely believed but still unproven conviction that $\mathbf{P} \neq \mathbf{NP}$ [For09], which would imply that there is no algorithm solving any of them efficiently, i.e. without an exponentially exploding time cost. A standard example for a constrained NP-hard combinatorial optimization problem is the *Knapsack Problem*, which aims at packing a knapsack with items of the utmost aggregated value while simultaneously not exceeding its capacity [MT90; KPP04]. Thanks to the simplicity of its formulation and the wide range of broad scenarios in which it can be identified, the Knapsack Problem is popular in both classical and quantum studies. In his survey, Holst [Hol22] elaborates on three different ideas to design a QAOA for the Knapsack Problem (meaning three different ways to take care of the constraint) based on [Roc+20; GH19; MW19] and compares their approximation qualities, particularly by evaluating their dependencies on different construction parameters. What makes the Knapsack Problem interesting for us as a representative combinatorial optimization problem in this work is that Wilkening et al. [Wil+23] recently proposed a new quantum algorithm for it, a part of which - called *Quantum Tree Generation (QTG)* - is suited to be extracted and recycled to set up a QAOA following the *Grover-mixer approach* [BE20] where constraints are incorporated via the initial state preparation.

For problems like the Knapsack Problem where instances consisting of up to 100,000 variables can be solved classically within a range of a few seconds [MT90, S.2.10], it is pretty involved to come up with a quantum algorithm that achieves a quantum advantage at all - under the current restricted and error-prone circumstances this seems rather impossible. Although VQAs shift an essential part of the effort to comparably cheap classical infrastructure, the qubit resources required for outperforming such sophisticated classical algorithms that have been in active development for over 50 years now are - at least to this point in time - clearly not within reach. An auspicious strategy to tackle this challenge is to build on the VQA idea of combining quantum and classical computation but to turn the tables: Instead of starting from the quantum algorithmic perspective and relinquishing certain parts to a classical system, why not trying to enhance a well-established classical algorithm by enhancing a subroutine in a quantum fashion? Such an approach is e.g. taken by Svensson et al. [Sve+21]. Similarly, we will impose this concept on the *Branch-and-Bound algorithm (B&B)* [LD60; Cla99], which is a standard classical technique for exactly solving combinatorial optimization problems that generates a tree-structured arrangement of subproblems by iteratively assigning variables and rejecting partial configurations according to certain rules. Our target picture is to construct a Branch and Bound for the Knapsack Problem that is extended by a QTG-induced QAOA, to ultimately end up with a hybrid quantum-classical B&B algorithm (HQCBB) for which the number of necessary qubits can be adjusted to the available hardware. How this may be done in detail is to

be worked out. Nevertheless, I want to emphasize at this early stage that this thesis has a quantum-physical focus, which is why more effort will be put in designing the QAOA compared to the rather basic classical components. Irrespective of that, keeping the full classical computation power by employing tried and tested methods might, in combination with a configurable qubit requirement, in fact be the most promising way for quantum computing to gain actual industrial relevance in the medium term.

Ideally, our final HQCBB is capable of improving its classical origin on a measurable scale. Regardless of the best case scenario, the HQCBB will inherit a non-material value by establishing a proof of concept for enhancing well-performing classical algorithms with the use of VQAs. Besides, a subordinate second part that is of scientific interest for the quantum computing community was already implicitly mentioned above: the construction of a new constraint-respecting QAOA for the Knapsack Problem, induced by a revolutionary procedure that prepares the feasible states.

This work consists of three parts. In the first, we will lay the theoretical foundation for constructing our algorithm; more specifically, after reviewing basic concepts like complexity theory and a formal description of combinatorial optimization problems as well as the Knapsack Problem in particular in Chapter 1, we will turn to the two algorithms that are to be employed. Section 2.1 discusses Branch and Bound as the classical framework and its building blocks together with common configurations. In Section 2.2 we will dive deeply into the subject of QAOA - including a rigorous derivation from an adiabatic evolution, an explanation of the original version by Farhi, Goldstone, and Gutmann [FGG14], and a presentation of the Grover-mixer approach [BE20] embedded in a discussion of the generalization suggested by Hadfield [Had18]. Part II can be considered central, as this is where the concrete idea for our algorithm is developed (Chapter 3) and both classical and quantum part are worked out explicitly (Chapter 4 and Chapter 5, respectively). Finally, we will elaborate on the implementation of our designed QAOA and visualize the different components as circuits in Chapter 6 as well as perform simulations in Chapter 7, making up the final part. Both of them have their reason to be of fundamental importance for the thesis: Concerning the implementation, translating the steps to unitary operations and decomposing them into a sequence of known quantum gates really generates the essence of a quantum algorithm, as it enables to categorize and assess its complexity in time and space and thereby establishes comparability. On the other hand, simulating qubits and the application of gates on a classical machine is computationally very expensive. In order to not be governed by severe limits on the problem size, I developed a simulator inspired by Wilkening et al. [Wil+23], which circumvents the conventional techniques. The result is specifically tailored to the Knapsack Problem and allows, as we will see, to emulate the QAOA circuits on more than 100 qubits at its best.

Part I.

Theoretical Background

Basic Concepts

1.1. Complexity Theory

As described in the introduction, the presumption - or at least the hope - that quantum computers will provide a real advantage over classical computers is widely spread. As scientists we need something - some measure - to be able to quantitatively distinguish the capabilities of quantum computers compared to their classical analogs. However, this field of studies is not the first wishing to estimate the resources needed to solve a computational problem. How to, for example, decide which problem out of a given list is the easiest one in terms of solvability, and which the most difficult? Or, alternatively, which algorithm should be used for a given problem to solve it most efficiently in terms of computing time? The theoretical framework equipping computer scientists, physicists and mathematicians with the necessary toolkit to answer these types of questions is the complexity theory.

Solving a problem is obviously not always equally difficult; matrix inversion for instance is way easier if the given matrix is diagonal. This is why one may stumble across terms *best-case* complexity and *worst-case* complexity, referring to the time needed to find solutions to the easiest and hardest instances of a problem, respectively (as far as these properties can be assigned uniquely). As different algorithms may well perform differently on the same problem but with different sizes, one is generally interested in the asymptotic runtime of an algorithm. In fact, we don't need the full relation between an algorithm and its runtime or memory needs - information about the leading term, given without any prefactors, is sufficient to benchmark it. This is formalized by the so-called $\mathcal{O}$-notation: For a function $f : \mathbb{R} \to \mathbb{R}$ we define

$$\mathcal{O}(f) = \{g : \mathbb{R} \to \mathbb{R} \ : \exists\, c, x_0 \in \mathbb{R} \text{ s.t. } 0 \leq g(x) \leq cf(x)\ \forall x \geq x_0\}.$$

Suppose the size of a problem is given by an integer $n \in \mathbb{N}$ which also characterizes the runtime $f(n)$. If there is another integer $k \in \mathbb{N}$ such that $f \in \mathcal{O}(n^k)$, i.e. the algorithm has polynomial runtime, one says the problem is *efficiently solvable*. Note that not any algorithm of polynomial runtime necessarily needs to be efficient in practical means.

But what if we want to make a general statement about a problem, e.g. that it can indeed be solved in polynomial time, or - even stronger - that there cannot be an algorithm achieving that? Even though any working algorithm provides an upper bound on the complexity of the considered problem, the chosen algorithm might just be comparatively bad. Hence, we aim for an algorithm-independent framework for describing the difficulty to solve a problem. Introducing the notion of *complexity classes* enables us of doing so: By definition, all problems contained in the same complexity class are comparable in terms of the resources required to solve them. The most prominent example of a complexity class is **P** (abbreviating "polynomial"), containing all problems that can be deterministically solved in polynomial time, i.e. are efficiently solvable. The next class - central for the purposes of this thesis - is **NP** (shortcut for "non-deterministic polynomial"). The proper definition of **NP** may not be that intuitive: It contains all problems to which a non-deterministic computer can find a solution in polynomial time. A non-deterministic computer (or *Turing machine*) is capable of not only executing single instructions like "Add numbers a and b" but of doing things like "Add a and b or subtract them" in parallel. Iterating these multiple operations leads to an exponentially growing tree instead of only a single path of instructions. The following is another often used definition of **NP**, which is probably easier to grasp: If provided a solution to a problem in **NP**, a usual (deterministic) computer is able to verify it in polynomial time. A standard example is prime factorization - while finding such a factorization into prime numbers is hard, verifying the correctness of a solution couldn't be simpler as the factors have just to be multiplied.

A sophisticated notion of what is meant by a "problem" in this thesis will follow in the next section. However, for a very high-level type of problem it makes sense to already be introduced at this stage.

Definition 1.1. A *decision problem* is a problem that can be fully answered by a "yes" or a "no".

In fact, any "ordinary" problem may be easily transformed into an associated decision problem by comparing the solution (its value) to a certain threshold value specified in advance.

Definition 1.2. A decision problem $\tilde{P}$ is called to be *reducible* in case there exists another decision problem $P \neq \tilde{P}$ such that $\tilde{P}$ can be transformed into P in polynomial time, denoted as $\tilde{P} \leq P$. This is, solving $\tilde{P}$ is equivalent to solving P up to a polynomial-time overhead. One says, $\tilde{P}$ *can be reduced to* P.

This concept can now be used to introduce an important subclass of **NP**.

Definition 1.3. A decision problem P is called *NP-complete* if

(i) $P \in \mathbf{NP}$ and

(ii) $\tilde{P} \leq P \quad \forall \tilde{P} \in \mathbf{NP}$ decision problems.

Another central and strongly related class is defined as follows.

Definition 1.4. A decision problem P is called *NP-hard* if it satisfies property (ii) of Definition 1.3, i.e. if every (other) decision problem in **NP** can be reduced to P.
For an ordinary problem being NP-hard[1] is defined to mean that the corresponding decision problem is NP-complete.

Due to the requirement of only obeying property (ii) in Definition 1.3, NP-hard problems are often referred to as being at least as difficult as the hardest problem instances in **NP**. Even though the expression for that class may be a bit misleading, Definition 1.4 also implies that an NP-hard problem does not necessarily have to also be contained in **NP**.

The following theorem can be understood as having provided the initial ignition for today's wide-spread interest on both scientific and industrial interest in NP-hard and NP-complete problems.

Theorem 1.5 (Cook's Theorem)**.** *The Boolean satisfiability problem, asking to determine whether a given Boolean expression can be satisfied*[2]*, is NP-complete. Here, a* Boolean expression *consists of a logical combination of variables, also called* literals*, each of which can only be assigned the value "true" or "false". It is* satisfiable *if there exists a choice of variables making the whole statement true.*

Proof. A proof of this theorem can be found in the original work by Cook [Coo71]. □

One main reason for being considered so groundbreaking is that Theorem 1.5 laid the foundation for a landmark work by Karp [Kar72] in which he used Cook's theorem to deduce NP-completeness of 21 combinatorial and graph-theoretical (decision) problems by obtaining each of them via an iterative reduction from the Boolean satisfiability

[1] Note that complexity classes themselves are printed in bold, e.g. **NP**, whereas the property of a problem being contained in that class is written without bold letters.

[2] Corresponding to a "yes" in terms of a decision problem as in Definition 1.1.

problem. This process ends up in a tree structure where every of these 21 problems represents a node in the tree, having the problem it is derived from as its respective parent node (the Boolean satisfiability problem therefore corresponds to the root node). In this picture, all problems sitting on the same branch of the resulting tree can be considered related, providing an instructive visualization of what NP-completeness means in practice. The problems addressed in [Kar72] are also known as *Karp's 21 (NP-complete) problems*. As it was one of the first, if not the first, to demonstrate the computational intractability of many standard problems in computer science on the large scale, Karp's paper was responsible for taking the popularity of this research area to a new level.

NP-complete problems are of special interest as solving one efficiently is sufficient to infer that all problems in **NP** are actually efficiently solvable, by which $\mathbf{P} = \mathbf{NP}$ would be implied. Instead of describing how large the impact of such an algorithm would be, I may rather refer to the Clay Mathematics Institute which listed the "**P** versus **NP**" problem among the seven Millennium Prize Problems each worth one million US dollars. To provide one example, let me pick up again what was said in the introduction: Modern cryptography is based on the conjecture that there are indeed problems that cannot be solved in polynomial time (see e.g. the RSA cryptosystem which uses the hardness of prime factorization [RSA78]). In fact, surprisingly little is known about how complexity classes are related, making the assignment of algorithms to specific classes being often based on unproven assumptions.

With the branch of new opportunities opened by quantum computing, the need of an extended complexity theory was recognized. For this sake, a new complexity class has been introduced: **BQP** (shorthand for "bounded-error quantum polynomial"), consisting of all problems that can be efficiently solved on a quantum computer, i.e. a quantum algorithm is able to find a solution in polynomial time, allowing a bounded probability of error, e.g. 1/3. Thereby it is the analogue of the classical complexity class **BPP** ("bounded-error probabilistic polynomial") also characterized by a bounded (but non-zero) error probability. The specific choice on the permitted failure probability however is arbitrary as it may be decreased to any finite non-zero value by running the respective algorithm multiple times, implied by the Chernoff bound [Che52]. Prime factorization can again be exploited as an example problem contained in **BQP** thanks to Shor's algorithm [Sho94]. Analogous to $\mathbf{P} \subseteq \mathbf{BPP}$ (as a deterministic machine is a special case of a probabilistic machine), a fundamental relation in quantum complexity theory is $\mathbf{P} \subseteq \mathbf{BQP}$, expressing the finding that any classical circuit can also be simulated on a quantum computer. That can be traced back to replacing any classical circuit by an equivalent circuit consisting of reversible operations only, mainly due to being able to simulate the NAND gate using the so-called Toffoli gate [NC10].

1.2. Combinatorial Optimization Problems

The Knapsack problem corresponds to a certain type of problems, namely *combinatorial optimization problems*. The characterizing property shared by all combinatorial optimization problems is their discrete domain, meaning that we need to find an optimal object out of countable many finite competitive objects at the end of the day [Coo+]. Clearly, the set of feasible solutions badly depends on the investigated problem in terms of structure and cardinality.

1.2.1. Notation

For the sake of conciseness and comparability, a bunch of definitions and a kind of mathematical formalism shall be introduced to describe combinatorial optimization problems, heavily borrowing notation from [Bin22].

Definition 1.6. A *combinatorial optimization problem* can be uniquely characterized by five components, i.e. is a quintuple

$$\text{COP} := \Big(N,\ \{c_j\}_{j=1}^{a},\ \{C_j\}_{j=1}^{a},\ \{D_k\}_{k=1}^{b},\ \text{ext}\Big) \tag{1.2.1}$$

where $N \in \mathbb{N}$ describes the *size* of the problem; $c_j \in \mathbb{R}_+$ for $j \in \{1, ..., a\}$ denotes the *cost* or *profit* of the corresponding *clause* $C_j : \{0,1\}^N \to \{0,1\}$; the maps $D_k : \{0,1\}^N \to \{0,1\}$ for $k \in \{1, ..., b\}$ however represent the *constraints* of the combinatorial optimization problem COP; ext $\in \{\min, \max\}$ finally specifies the type of the problem, i.e. whether it asks for a minimization of costs or a maximization of profits.
A typical special case of combinatorial optimization problem is the *unconstrained* one where there are no constraints at all, i.e. $\{D_k\}_{k=1}^{b} = \emptyset$ or, equivalently, $b = 0$.

To be more concrete, the size of a combinatorial optimization problem is given by the number of bits required to formulate it quantitatively (the number of bits passed as arguments to the clauses and constraints). Instead of writing $z \equiv (z_1, ..., z_N)$ where z_i for $i \in \{1, ..., N\}$ represents a single bit, one typically combines them to a *bit string* $z \equiv z_1 \cdots z_N$ such that

$$C_j(z_1, ..., z_N) \equiv C_j(z) \equiv C(z_1 \cdots z_N) \quad \text{for } j \in \{1, ..., a\},$$

and analogously for the constraints. A clause C_j is said to be *satisfied* by a bit string $z \in \{0,1\}^N$ if $C_j(z) = 1$, otherwise it is called *unsatisfied*. Again, the same naturally

holds for the constraints. Note that we assume all profits $c_j, j \in \{1, ..., a\}$, to be integers; however, this is not a necessary requirement, we could have also introduced them more generally as $c_j \in \mathbb{R}_+$.

Definition 1.7. The *objective function* of a combinatorial optimization problem COP is given by

$$C : \{0,1\}^N \to \mathbb{N},\ z \mapsto C(z) = \sum_{j=1}^{a} c_j \cdot C_j(z). \tag{1.2.2}$$

The objective function of a combinatorial optimization problem is often understood as its main ingredient. The following terminology provides a higher-level notion of problems as we understand them:

Definition 1.8. An *integer linear program (ILP)* is a problem in which the variables are all restricted to be integers and the objective function as well as the constraints are linear (except for the constraints enforcing the variables to be integers).

The upcoming two definitions will equip us with an improved capability of discussing solutions to combinatorial optimization problems, especially in the case of a constrained COP.

Definition 1.9. A bit string z is called a *feasible solution* to a COP defined as in Definition 1.6 if

$$D_k(z) = 0 \quad \forall k \in \{1, ..., b\},$$

meaning that none of the constraints happens to be violated by z. The set of all feasible solutions to the COP shall be depicted as

$$\text{feas(COP)} := \{z \in \{0,1\}^N : D_k(z) = 0\ \forall k \in \{1, ..., b\}\}. \tag{1.2.3}$$

Naturally, a COP is said to be *infeasible* in case $\text{feas(COP)} = \emptyset$.

Definition 1.10. A bit string z^* is called an *optimal solution* to a COP defined as in Definition 1.6 if

(i) z^* is a feasible solution to COP, i.e. $z^* \in \text{feas(COP)}$, and

(ii) $z^* = \underset{z \in \{0,1\}^N}{\text{ext}\arg} C(z)$,

which can be streamlined to

$$z^* = \underset{z\in\text{feas(COP)}}{\text{ext}\arg} C(z) = \underset{z\in\text{feas(COP)}}{\text{ext}\arg} \sum_{j=1}^{a} c_j \cdot C_j(z),$$

meaning that z^* minimizes the cost or maximizes the profit resulting in a corresponding extremization of the objective function.
Accordingly, the set of the potentially many optimal solutions to COP shall be denoted by

$$\text{opt(COP)} := \left\{ z \ : \ z = \underset{z\in\text{feas(COP)}}{\text{ext}\arg} C(z) \right\}. \tag{1.2.4}$$

Obviously, finding optimal solutions to a given COP is the holy grail of mathematical optimization. It is clear that the following heuristic or algorithm solves any COP to optimality.

Definition 1.11. Given a COP as in Definition 1.6, a *brute-force search* systematically enumerates all bit strings $z \in \{0,1\}^N$, checks whether each such *candidate* satisfies the constraints and, if yes, caches the corresponding bit string together with its associated objective-function value.

Since the search space of a problem formulated in N binary variables has 2^N possible candidates, the brute-force search is of complexity $\mathcal{O}(2^N)$. As we will see, searching for optimal solutions of a given COP can easily become an intractable task, which itself can be understood as the most fundamental motivation for this thesis and the idea of exploiting quantum computation in mathematical optimization in general.

As kind of a marginal note, I will lastly show how maximization and minimization problems[3] can easily be transformed into each other. Naturally, the clauses need to be reversed in some sense. The constraints however stay unchanged as we could otherwise not speak of the same problem only being transformed in the wanted extremum anymore. This is, for COP as in Definition 1.6,

$$\text{COP} \mapsto \overline{\text{COP}} := \left(N,\ \{c_j\}_{j=1}^{a},\ \{1 - C_j\}_{j=1}^{a},\ \{D_k\}_{k=1}^{b},\ \overline{\text{ext}} \right)$$

with

$$\overline{\text{ext}} \equiv \begin{cases} \min & , \text{ if } \quad \text{ext} = \max \\ \max & , \text{ if } \quad \text{ext} = \min \end{cases}$$

[3] Here and here after, when speaking of a "problem" we actually mean a combinatorial optimization problem.

describes the relation between formulating a combinatorial optimization problem as maximization problem versus formulating it as a minimization problem. As COP and $\overline{\text{COP}}$ by construction do not differ in terms of constraints, we can directly infer $\text{feas}(\text{COP}) = \text{feas}(\overline{\text{COP}})$. Even further, using Eq. (1.2.4) it is easy to verify that a (feasible) solution z^* to $\overline{\text{COP}}$ is optimal if and only if z^* is also an optimal solution to COP, meaning $\text{opt}(\text{COP}) = \text{opt}(\overline{\text{COP}})$. This in turn implies that COP and $\overline{\text{COP}}$ are fully equivalent.

1.2.2. Quantum Mechanical Version

So far, our notion of combinatorial optimization problems is merely classical. However, if it was not possible to tackle those problems using a quantum computer, i.e. leveraging quantum mechanics, this thesis would have no reason d'être. And indeed, there is a general approach for addressing a COP with quantum computers for which it makes sense to be presented at this stage.

When making the transition from classical to quantum computation, every (classical) bit z_n is replaced by a qubit $|z_n\rangle$. A single-qubit system shall be denoted by $\mathfrak{q} \cong \mathbb{C}^2$, implying that an N-qubit system may be represented as

$$\mathfrak{q}^N \equiv \mathfrak{q}^{\otimes N} = \bigotimes_{n=1}^{N} \mathfrak{q} \cong \bigotimes_{n=1}^{N} \mathbb{C}^2 \cong \mathbb{C}^{2^N}.$$

Accordingly, its computational basis is given by

$$\{|z\rangle \equiv |z_1 \cdots z_N\rangle \,:\, z_n \in \{0,1\}\ \forall n \in \{1,...,N\}\} = \left\{|z\rangle \,:\, z \in \{0,1\}^N\right\}.$$

There is actually a second way of labeling the computational basis states to be mentioned that is totally equal in terms of the frequency of use and that we will encounter again later: An element in the computational basis may equivalently be referred to via the natural number encoded in the bit string $z \equiv z_1 \cdots z_N$, namely

$$z_1 \cdots z_N \equiv z \equiv \sum_{n=1}^{N} 2^{N-n} z_n, \tag{1.2.5}$$

also known as *binary representation*. In this sense, the computational basis of an N-qubit system $\mathfrak{q}^N$ reads

$$\left\{|0\rangle, ..., |2^N - 1\rangle\right\} = \left\{|z\rangle \,:\, z \in \left\{0, ..., 2^N - 1\right\}\right\}.$$

The idea now is to treat the objective function C of a given COP as in Definition 1.6 as a Hamiltonian acting on the N-qubit system $\mathfrak{q}^N$, i.e. understand it essentially as a

self-adjoint operator acting on the Hilbert space of the qubit system instead of a map $\{0,1\}^N \to \mathbb{N}$ as in Definition 1.7. This objective Hamiltonian is generally designed such that it is diagonal in the computational basis:

$$C\,|z\rangle := C(z)\,|z\rangle \quad \forall z \in \{0,1\}^N. \tag{1.2.6}$$

Thereby, the problem of finding an optimal solution to a given COP with objective function $C : \{0,1\}^N \to \mathbb{N}$ translates to the quantum mechanical context as finding an extremal eigenstate of the restricted operator $C|_F$ with C as in Eq. (1.2.6) and F being defined via:

Definition 1.12. The *feasible solution space* (also called *feasible subspace)* of an optimization problem COP is given by

$$F := \mathrm{span}\{|z\rangle \,:\, z \in \mathrm{feas(COP)}\} \subseteq \mathbb{q}^N. \tag{1.2.7}$$

Analogously:

Definition 1.13. The *optimal solution space* of an optimization problem COP is given by

$$F_{\mathrm{opt}} := \mathrm{span}\{|z\rangle \,:\, z \in \mathrm{opt(COP)}\} \subseteq F. \tag{1.2.8}$$

F_{opt} describes the eigenspace of $C|_F$ corresponding to the extremal eigenvalues.

Concerning the possible constraints of a combinatorial optimization problem, transferring them to a quantum mechanical fashion is only straight forward when softcoding them via a translation to additional objective function clauses with penalties that make violating them energetically unfavorable (see also Section 2.2.2). Roughly, this is due to using only the information provided by the objective function to build a Hamiltonian for describing the problem at hand and determining the evolution of the system.[4] The type of the problem, i.e. whether it asks for the maximization or the minimization of the objective function, can however be simply incorporated by choosing the sign of the objective Hamiltonian accordingly.

[4] As usual when operating on the level of quantum mechanics, the term "determining" has to be taken with a grain of salt as it is not a deterministic but a probabilistic theory.

1.3. The Knapsack Problem

This section aims at formally introducing the problem for which our algorithm will be built. It is a standard example of a combinatorial optimization problem and can be read up in more or less detail in standard literature. I confine myself to only discuss the formulation that is to be used in the thesis and embed it in some accompanying information, as it is a pretty rich subject on its own, including diverse variants and a multitude of different solving ideas, approaches and techniques. The appropriate place and time to talk about classical results and, especially, bounds is when constructing our algorithms explicitly, since only some of the developments achieved over the last decades will actually be exploited by us. Even among standard combinatorial optimization problems (assuming that there is such a more or less sharply defined list) the so-called Knapsack Problem occupies a special role. It can even be called popular for two main reasons: first and foremost, its ridiculous structural simplicity and, on the other hand, its excellent applicability to daily-life situations.

Imagine you are a school kid packing all the heavy books you need for the subjects today before you leave; however, your satchel can only carry a limited weight - the task to optimize this situation, i.e. to select the best combination of books (assuming some subjects are more important than others) without simultaneously overloading your school bag, really materializes the Knapsack Problem.

Definition 1.14. The *Knapsack Problem* (KP) is an ILP (cf. Definition 1.8) consisting of a list of N items such that every item is assigned two positive integers, namely a profit and a weight, and a knapsack which is associated with a capacity. It is then defined as follows:

$$\text{maximize} \quad P(x) := \sum_{j=1}^{N} p_j x_j \tag{1.3.1}$$

$$\text{subject to} \quad W(x) := \sum_{j=1}^{N} w_j x_j \leq W_{\max}, \tag{1.3.2}$$

$$x_j \in \{0,1\}\ \forall j \in \mathbf{N} \equiv \{1,...,N\} \tag{1.3.3}$$

where the $p_j \in \mathbb{N}$ are called *profits* and the $w_j \in \mathbb{N}$ are referred to as *weights*, $W_{\max}$ denotes the *capacity* of the knapsack and $j \in \mathbf{N}$ indicates the item such that

$$x \equiv x_1 \cdots x_N \quad \text{and} \quad x_j := \begin{cases} 1 & , \text{ if item } j \text{ is selected} \\ 0 & , \text{ otherwise.} \end{cases}$$

Indeed, Eqs. (1.3.1) to (1.3.3) represent the so-called *canonical form* of an integer linear program. To put it in the shape of Definition 1.6, the Knapsack Problem as

combinatorial optimization problem can be written as

$$\mathrm{KP} = \Big(N,\ \{p_j\}_{j=1}^{N},\ \{x_j\}_{j=1}^{N},\ W',\ \max\Big) \tag{1.3.4}$$

where

$$W'(x) := \begin{cases} 0 & , \text{ if } \quad W(x) \leq W_{\max} \\ 1 & , \text{ otherwise.} \end{cases}$$

in accordance with Definitions 1.6 and 1.9.

As argued by Martello and Toth [MT90, S.2.1], we are indeed justified to make the following three assumptions without loss of generality[5]:

(i) The profits p_j and weights w_j for $j \in \mathbf{N}$ as well as the capacity $W_{\max}$ are non-negative integers, i.e. $p_j, w_j, W_{\max} \in \mathbb{N}\ \forall j \in \mathbf{N}$.

(ii) The sum of all weights is larger than the capacity, i.e. $\sum_{j=1}^{N} w_j > W_{\max}$.

(iii) No single weight is larger than the capacity, i.e. $w_j \leq W_{\max}\ \forall j \in \mathbf{N}$.

The latter two assumptions ensure that the Knapsack Problem stated as in Definition 1.14 is providing just as much information as necessary. This is, there is no other KP instance which can be considered equivalent in the sense of the number of variables to be optimized while actually being reduced by those variables that need to be trivially set to secure feasibility at all. For instance, suppose assumption (ii) is violated, the optimal solution in this case is trivially given by $x_j = 1\ \forall j \in \mathbf{N}$ (even selecting all items is not overfilling the knapsack). On the other hand, if assumption (iii) is not satisfied, any optimal solution must agree on $x_j = 0$ for those $j \in \mathbf{N}$ for which $w_j > W_{\max}$ in order to preserve feasibility (cf. Eq. (1.2.3)). Verifying that assumption (i) can be made safely is a bit more involved. More specifically, at least concerning the non-negativity - facing fractions instead of integers can be circumvented by multiplying through by a proper factor. For the non-negativity I will briefly reconstruct the four steps described by Martello and Toth [MT90] (inspired by Glover [Glo65]) how to transform a KP instance with negative profits and/or weights into our standard version as presented in Definition 1.14:

(1) Set $x_j = 0$ for each $j \in \mathbf{N}^0 := \{j \in \mathbf{N} : p_j \leq 0 \text{ and } w_j \geq 0\}$.

(2) Set $x_j = 1$ for each $j \in \mathbf{N}^1 := \{j \in \mathbf{N} : p_j \geq 0 \text{ and } w_j \leq 0\}$.

[5] Hereafter referred to as "w.l.o.g.".

(3) Define a set of new binary variables $y_1, ..., y_N \in \{0, 1\}$ via

$$y_j := \begin{cases} 1 - x_j & , \text{ if } \quad j \in \mathbf{N}^- := \{j \in \mathbf{N} \,:\, p_j < 0 \text{ and } w_j < 0\} \\ x_j & , \text{ if } \quad j \in \mathbf{N}^+ := \{j \in \mathbf{N} \,:\, p_j > 0 \text{ and } w_j > 0\} \end{cases}$$

while also performing the following rescaling of profits and weights, respectively:

$$\overline{p}_j := \begin{cases} -p_j & , \text{ if } \quad j \in \mathbf{N}^- \\ p_j & , \text{ if } \quad j \in \mathbf{N}^+ \end{cases} \quad \text{and} \quad \overline{w}_j := \begin{cases} -w_j & , \text{ if } \quad j \in \mathbf{N}^- \\ w_j & , \text{ if } \quad j \in \mathbf{N}^+. \end{cases}$$

(4) Finally, instead of solving the original Knapsack Problem instance containing negative profits and/or weights, solve the following COP

$$\begin{aligned} \text{maximize} \quad \overline{P}(y) &:= \sum_{j \in \mathbf{N}^- \cup \mathbf{N}^+} \overline{p}_j y_j \; + \sum_{j \in \mathbf{N}^1 \cup \mathbf{N}^-} p_j \\ \text{subject to} \quad \overline{W}(y) &:= \sum_{j \in \mathbf{N}^- \cup \mathbf{N}^+} \overline{w}_j y_j \quad \leq \quad W_{\max} - \sum_{j \in \mathbf{N}^1 \cup \mathbf{N}^-} w_j \;\; =: \overline{W_{\max}}, \\ & y_j \in \{0, 1\} \; \forall j \in \mathbf{N}^- \cup \mathbf{N}^+, \end{aligned}$$

which can be interpreted as an equivalent ordinary Knapsack Problem instance in the sense of Definition 1.14 with non-negative profits and weights, having some offset in the objective function as well as an correspondingly enlarged capacity[6].

As summarized by Eq. (1.3.4), KP has just as many clauses as there are binary variables needed to formulate the problem (namely the number of items). The aforementioned structural simplicity of KP can also be read off Eq. (1.3.4), more specifically via the trivial clauses just being given by the respective bits on the one hand and the property of KP having only one constraint on the other. It is indeed a bit unfortunate that, unlike in Section 1.2.1, bit strings are denoted by x instead of z, profits are labeled p_j instead of c_j and the objective function of KP (given in Eq. (1.3.1)) is referred to as P instead of C. I chose this deviating notation to be in line with how the Knapsack Problem is usually formulated in the literature.

Just in order to give you a second real-life example relatable to the Knapsack Problem: For those whose school days were already to long ago and are more comfortable with assets, investments and portfolios: Suppose you have got a certain capital to extend or create your portfolio and you are considering a fixed number of investments to choose between. Every investment is assigned a cost and a profit you expect to obtain from it. When deciding on which investments to make in order to achieve the maximum profit

[6]The new capacity $\overline{W_{\max}}$ is indeed never smaller than the old, $W_{\max}$, since $w_j \leq 0 \; \forall j \in \mathbf{N}^1 \cup \mathbf{N}^-$.

with simultaneously not exceeding your financial leeway, you constructed an instance of the Knapsack Problem.

To be more rigorous, KP as defined in Eq. (1.3.4) should rather be called something like *single 0-1 Knapsack Problem* to be able to distinguish it properly from the various other variants that exist of the problem. As this longer name suggests, there is also a *multiple Knapsack Problem* where the items can be packed in more than one knapsack. The additional term "0-1" is related to the possible values of the variables $x_1, ..., x_N$, meaning in the context that items can only be either selected or left out. Allowing multiple instances of every item to be packed in the knapsack(s) gives the *multiple-choice Knapsack Problem*. When restricting this number of copies permitted (either equally for all items or not) to a value greater than one but smaller than infinity we arrive at the *bounded Knapsack Problem*. Another even simpler variant which is sometimes considered independent - and therefore also carries a not-knapsack related name - arises when $p_j = w_j$ for every item, called *Subset-sum Problem*. A detailed analysis of each of these variants is e.g. provided by Martello and Toth [MT90]. Even despite this bunch of different variations we will stick to referring to the problem specified in Definition 1.14 as Knapsack Problem as it is the only version of interest for this thesis.

The simplicity of KP as in Definition 1.14 makes it indeed interesting both for the theoretical and the practical side. Theorists mainly consider it relevant due to the following reason: Being able to find optimal solutions to the Knapsack Problem may aid in solving more involved combinatorial optimization problems as the structure of some version of the Knapsack Problem can often be found in subproblems. From the practical point of view, the Knapsack Problem is worth investigating since it is suitable to model diverse industrial situations as is indicated by the second descriptive KP example given above. Many advanced industry KP-applications such as cutting stock, financial decision problems like capital budgeting or portfolio selection as well as asset-backed securitization (just to name a few) are discussed by Kellerer, Pferschy, and Pisinger [KPP04, Ch.15].

Due to exactly this simplicity, it may be tempering to underestimate the complexity of solving the Knapsack Problem to optimality. The number of different bit strings to be compared in a brute-force approach (i.e. walk through every possible combination and pick the best iut of those satisfying the constraint) is 2^N for a set of N items. Hence, taking up the thought game in [MT90, p.1], even a hypothetical computer that is capable of checking one billion bit strings per second would need over 30 years for a set of 60 items, about ten centuries for 65 items and the computing time for 88 items is already in the order of the age of the universe. However, classical algorithms that are specifically designed for or adapted to the Knapsack Problem are able to solve the most problem instances within seconds, even those with extremely large amounts of

items up to 100,000 [MT90, S.2.10].

What is striking about the Knapsack Problem is its simple-looking formulation whereas actually being an NP-hard optimization problem (cf. Section 1.1).

Theorem 1.15. *The Knapsack Problem as it is formulated in Definition 1.14 is NP-hard. Its corresponding decision problem is NP-complete.*

Proof. In fact, the Knapsack Problem is one of Karp's 21 problems (cf. Section 1.1) [Kar72]. However, Karp's definition of the Knapsack Problem differs from the one in Definition 1.14 - it is indeed closer related to what we call the Subset-sum Problem. Therefore, we cannot just refer to his proof of NP-completeness, which is itself based on Cook's theorem (cf. Theorem 1.5). But nevertheless, what we are able to do is leveraging his proof for another NP-complete problem in the list, namely the *Partition Problem*. This approach is inspired by Martello and Toth [MT90, S.1.3].
Formulated as a decision problem (cf. Definition 1.1), the Partition Problem asks whether there exists a subset $S \subseteq \bar{N} \equiv \{1, ..., N\}$ for a list of N positive integers $s_1, ..., s_N$ such that $\sum_{j \in S} s_j = \sum_{j \in \bar{N} \setminus S} s_j$ (i.e. the subset S divides the total sum $\sum_{j \in \bar{N}} s_j$ in half, meaning that the answer is "no" if the $s_j, j \in \bar{N}$ do not sum to an even number). For this problem we can indeed refer to the proof given by Karp [Kar72].
The next intermediate problem whose NP-completeness is to be derived is the *Subset-sum Problem*. The task of its decision version is to determine whether there is a subset $S \subseteq \bar{N} = \{1, ..., N\}$ satisfying $\sum_{j \in S} s_j \geq t$ while simultaneously $\sum_{j \in S} s_j \leq c$ for $N + 2$ positive integers $s_1, ..., s_N$, t (threshold) and c (capacity).[7] As the Subset-sum Problem can be easily transformed into an equivalent instance of the Partition Problem by setting $t = c = 1/2 \cdot \sum_{j=1}^{N} s_j$ (which especially is a polynomial-runtime transformation), the Subset-sum Problem also needs to be NP-complete.
Finally, consider KP formulated as a decision problem, asking for a subset $S \subseteq \bar{N} = \{1, ..., N\}$ obeying

$$\sum_{j \in S} p_j \geq T \quad \text{and} \quad \sum_{j \in S} w_j \leq W_{\max} \quad \text{with } T \in \mathbb{N} \text{ (threshold).}$$

As indicated above, starting from that the Subset-sum decision problem is obtained by choosing $p_j = w_j \ \forall j \in \{1, ..., N\}$, meaning that it is a particular case of the decision version of the Knapsack Problem. Hence, we can infer the NP-completeness of the latter. This in turn is equivalent to the Knapsack Problem being NP-hard (cf. Definition 1.4). □

[7] Note that the Subset-sum Problem is often introduced as a decision problem solely. In that case, threshold and capacity are usually chosen as $t = c$, meaning that the question is whether there is a solution $S \subseteq \bar{N}$ to $\sum_{j \in S} s_j = c$.

Algorithms

In this chapter we are going to get familiar with the basic ideas and the general concepts behind the different algorithms that will be used in our HQCBB (cf. Part II). To this end, we will deal with both classical and quantum algorithms, laying the crucial foundation for the main part.

2.1. Branch and Bound

The so-called *Branch-and-Bound algorithm* (also referred to just as *Branch and Bound* or even *B&B*) is a classical algorithm for exactly solving combinatorial optimization problems. It was proposed by Land and Doig [LD60] in 1960; however, the first use of the term "Branch and Bound" to label this algorithm can be traced back to the work by Little et al. [Lit+63] some three years later.

2.1.1. Overview of the Algorithm

The general idea of Branch and Bound is to recursively divide the problem at hand into smaller and smaller subproblems and to dynamically investigate only the promising paths further. Thereby reducing the search space to explore in order to gain a significant advantage in terms of computing time compared to a simple brute-force search (cf. Definition 1.11) in turn is what makes the objective of the algorithm. By now, Branch and Bound finds itself among the most commonly used algorithms and techniques to tackle NP-hard optimization problems [Cla99]. The big advantage of B&B, decisive for its popularity, can be considered to be its very generic formulation, making it, first of all, applicable to any arbitrary COP (cf. Definition 1.6); on the other hand, all main ingredients of B&B are highly customizable such that the algorithm may be precisely tailored to the respective problem at hand. In the aim of finding out

what that concretely means, we will be guided mainly by the excellent state-of-the-art overview provided by Morrison et al. [Mor+16].

In the following, we will learn about the general structure of Branch and Bound in more detail; a concrete algorithm design is, however, to be postponed to Part II. The rationale of Branch and Bound can be summarized to be the iterative generation of subproblems associated with subsets of the search space and their exploration based on certain criteria, leading to a tree-structured set of subproblems T. At each iteration, the algorithm selects a new one out of the list of unexplored subproblems while a feasible solution $\hat{z}$ (cf. Definition 1.9), also called *incumbent*, is stored globally throughout the run. This incumbent is updated if another candidate with better objective-function value[1] is found within the currently investigated subproblem and returned in case that there are no further unexplored subproblems. Given a combinatorial optimization problem with search space $\mathbf{Z}$ and objective function C, Branch and Bound - encapsulating a whole family of algorithms all sharing the above rationale as a common core - can be compactly written, in its most generic form, as follows:

Algorithm 1: Branch-and-Bound($\mathbf{Z}, C$)

```
1  Set T = {Z} and initialize ẑ, ĉ := C(ẑ) ;
2  while T ≠ ∅ do
3      Select a (sub-)problem S ∈ T to explore ;
4      if S cannot be pruned then
5          Partition S into subproblems S_1, ..., S_r ⊂ S ;
6          T ← T ∪ {S_1, ..., S_r} ;
7          T ← T \ S ;
8          continue
9      if ∃ ẑ' ∈ S such that ext(C(ẑ'), ĉ) = C(ẑ') then
10         Update ẑ ← ẑ' and ĉ ← C(ẑ') ;
11     T ← T \ S ;
12 return ẑ, ĉ
```

As usual, ext $\in \{\max, \min\}$ in Line 9, depending on the type of optimization problem we have (compare Definition 1.6). Note that Algorithm 1 is guaranteed to terminate if the search space $\mathbf{Z}$ contains only finitely many elements and in case the partitioning procedure in Line 5 produces proper subsets $S_i \subset S$, i.e. $S_i \neq S \ \ \forall i \in \{1, ..., r\}$. As this thesis only deals with a binary ILP (cf. Definition 1.8), we can restrict our attention to these types of optimization problems, meaning that $\mathbf{Z} = \{0, 1\}^N$ for any problem of

[1] Assessing one objective-function value to be "better" than some other depends, of course, on the kind of optimization problem, i.e. whether it is a maximization or a minimization problem.

size N. Hence, in the knowledge of $\left|\{0,1\}^N\right| = 2^N$, the guarantee of termination can be given for all of our applications.

The complexity of a B&B algorithm on the basis of Algorithm 1 is determined by two properties of the specific implementation (and therefore depends on the problem in question). The first is the *branching factor* of the tree, denoted by b, which is given by the maximum value of r occurring throughout the algorithm, i.e. the largest number of *children* $S_1, ..., S_r$ generated by any node S.[2] In order to recognize the second, we need to realize that any path of nodes starting from the root of T will sooner or later come to an end, namely when the currently considered child can by pruned off according to our chosen criteria or if there are no more degrees of freedom. A path of nodes connecting the root of T with an "ending" node that does not have any children itself is called a *leaf*. Using this, the second factor is represented by the *search depth* of the tree, denoted by d, meaning the length of the longest *leaf* of the tree. Roughly speaking, the complexity depends on the maximum length and width dimensions of the tree. Hence, any B&B implementation has a worst-case running time of $\mathcal{O}(Mb^d)$ where M bounds the computing time that is maximally required to explore a subproblem.[3]

As $b > 1$ in any case (there need to be at least two children generated in Line 5 for a meaningful partitioning procedure), a general B&B implementation may well have exponential runtime. However - and that is why Branch and Bound has become established to today's extent - there are three ingredients identifiable in Algorithm 1 that allow for an almost arbitrarily advanced fine-tuning: the *searching strategy* in Line 3 determining the superior ordering according to which the next subproblem is selected in each iteration, the *pruning strategy* or *pruning rules* in Line 4 being responsible for a resource-efficient node investigation via preventing the further exploration of provably non-optimal "regions" of the search space and, last but not least, the *branching strategy* in Line 5 according to which the solution space is divided and new subproblems are created. The latter - as the only of the three - even made it into the name of the algorithm for being such a unique selling point. Having said that, the term "bound" can admittedly be considered included in what we understand as the pruning rules: Leveraging bounds on the objective function value of an optimal solution is indeed the most common example for such a rule to constitute whether a node is investigated further. More specifically, early B&B approaches operated on the basis of a single bound only, see e.g. the introductory course by Weinberg [Wei73]. However, modern approaches rather employ both upper and lower bounds on the objective function (compare e.g. the Stanford lecture notes by Boyd and Mattingley [BM10]) to estimate

[2] S is accordingly called *parent node*.

[3] Of course, it is not a realistic expectation to have a feasible estimation of M when constructing a B&B algorithm as it is a highly problem-specific quantity that could at most be found retrospectively.

and assess the quality of a partial solution.[4] So, the only of the three ingredients in Algorithm 1 not being reflected in the name "Branch and Bound" is the searching strategy. However, people often associate the searching and the branching strategy with only one component that includes both the generation of subproblems and the subsequent selection of the next tree node.

2.1.2. Relations Between Algorithm Components

Even though the three discussed constituents of Algorithm 1 seem to be quite independent, they can actually have an influence on each other. This is, fixing one of these may affect the selection of another one in the aim of a best performance of the algorithm. For instance, the specification of pruning rules typically impacts the choice of a searching and a branching strategy by restricting the feasible options; however, this occurs mainly in more advanced implementations where pruning is not only carried out on the basis of bounds but also includes involved methods like *cutting planes*, *column generation* or *dominance rules* [Mor+16]. Moreover, a good choice of a searching strategy may depend on the explicit branching strategy, as e.g. an unbalanced tree structure (meaning that the tree is not symmetric around an arbitrary axis passing vertically through some parent node and thereby dividing the corresponding set of children in two parts) can be compensated by a suitable node selection heuristic [Mor+16]. These relationships between the three algorithm components are depicted by dashed lines in Fig. 2.1; the corresponding directions of influence are represented by arrows in a natural way.

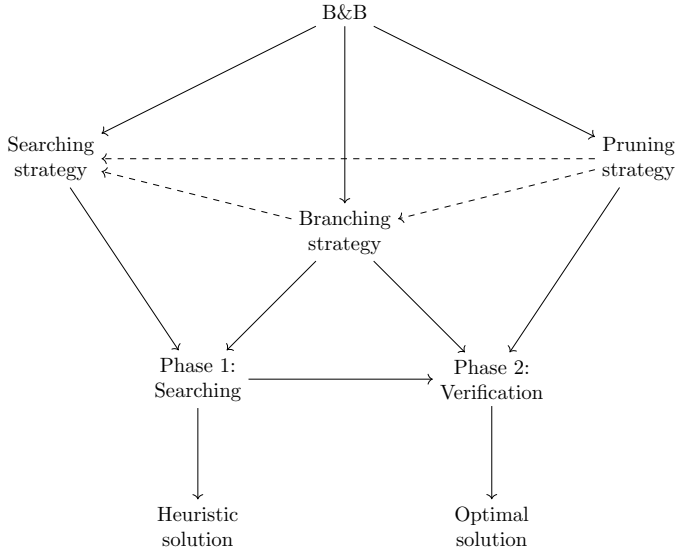


Figure 2.1.: Overview of B&B components, phases and their respective relationships.

[4] It would therefore be appropriate to refer to the algorithm as "Branch and Bounds".

Fig. 2.1 also outlines that any B&B run may be considered to consist of two consecutive phases - the *searching phase* and the *verification phase*. As their names suggest, the algorithm has not yet found an optimal solution during the former whereas there are still untouched tree nodes (meaning children that were generated at some previous iteration but have not been selected for exploration) with the incumbent being already optimal in the latter. Note that a certificate of optimality for the incumbent cannot be obtained as long as there are unexplored nodes remaining in the tree, since any of the corresponding search space regions could potentially contain a better solution. Therefore, these two phases are only identifiable retrospectively or from an outside view point, as it is not possible to prove optimality of an incumbent and thereby detect a "phase transition" during the run of the algorithm. This also means that prematurely aborting the algorithm run can only result in a heuristic candidate solution whose quality cannot be assessed with certainty. Not without reason the same terms are used in the names "searching strategy" and "searching phase" - this strategy primarily impacts the eponymous phase: In the ideal case, the heuristic according to which the next node is selected shall not influence the performance of the algorithm once the incumbent occupies an optimal solution, as it is not necessary anymore to investigate any further node in terms of children generation. Pruning rules, on the other hand, will have the strongest effect as soon as an optimal solution is found due to the vast majority of open subproblems performing poorly in comparison afterwards. In contrast, the branching strategy's area of influence is not restricted to only one phase; instead, it impacts the searching phase via being capable of leading towards or away from an optimal solution (in terms of computing time) and the verification phase by determining the number of nodes that are still to be explored. While all three discussed strategies may significantly improve the performance of a B&B algorithm, most literature focuses on the pruning rules [Mor+16]. The *Branch and Cut* [PR91] and the *Branch and Price* [Bar+98] algorithms, having developed from the standard Branch and Bound concept presented here by using the techniques of *cutting planes* and *column generation* as pruning approaches, respectively, and being considered as separate algorithms in their own right, underpin this claim. Now the circle closes with regard to what I mentioned earlier, namely that exploiting bounds on the objective function is the most basic or most frequently applied technique to establish a criterion based on which subproblems are pruned off, which is why people often have this specific kind of pruning strategy in mind when talking about *Branch and Bound*.

For the sake of completeness, there is one step in Algorithm 1 that we ignored so far but whose importance is actually not to be neglected: the initialization of the incumbent and its associated objective-function value in Line 1. As first noticed by Danna [Dan08], tree search methods tend to feature a high sensitivity to initial conditions. For instance, starting with an optimal solution in our B&B scheme will very likely reduce the size of the computed tree (i.e. fewer nodes need to be investigated) by a significant amount if the pruning strategy is chosen appropriately. While this

"erratic" behavior is often treated as a drawback, there are also approaches that exploit the performance variability to improve the overall strength of the B&B algorithm, see e.g. the work by Fischetti and Monaci [FM14].

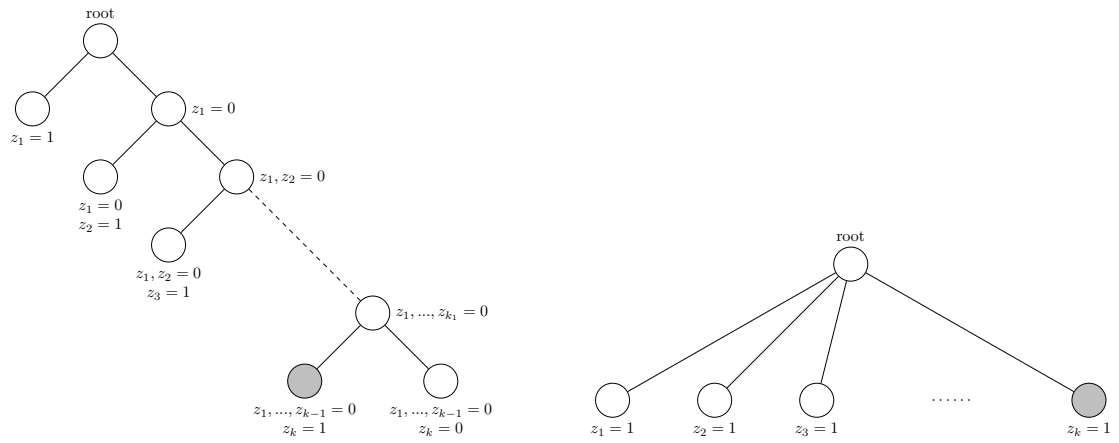
Even though this is not the appropriate time for specific algorithm design decisions, one aspect of Algorithm 1 may reasonably be concretized at this stage. As already mentioned, we are limiting our investigations to 0-1 ILPs or, to put it differently, a search space $\mathbf{Z} = \{0,1\}^N$ with N denoting the problem size. In this case, Line 5 of Algorithm 1 can be simplified: Each branching step reduces the problem to solve by fixing certain variables to either value 1 or value 0. This is, the partitioning procedure resp. the generation of subproblems is given by the assignment variables. But nevertheless there are still different types of branching strategies that can be applied, namely two in particular as we will see soon. In a sense of foreshadowing, let me already give it away that we will stick to the basic B&B version in which bounds on the objective function are making up the only pruning rules, since the main focus of this thesis is still located in the quantum physical area. Apart from that, the performance of the basic B&B implementation even using the standard pruning rules is so good that it suffices for our purposes.[5] So before we get to discuss some common options for the searching and the branching strategy only, I will elaborate a bit on how the pruning with bounds works, as the neat idea behind it is easy to grasp: In case of a maximization problem, for each subproblem an upper bound on the objective function is computed in a specified way; thereupon, the corresponding node can be pruned if its associated upper bound is smaller than the best lower bound found in the investigation of the tree so far, possibly updated at each child generation iteration (this is done vice versa for a minimization problem). This heuristic is pretty intuitive, as a region of the search space can be guaranteed to not contain an optimal solution if the best possible value is still worse than worst achievable value in another region already explored.

2.1.3. Common Branching Strategies

Let us now start with the branching strategies, as there are fewer of them, namely two as mentioned above. The first (and also the most common one) is the *binary branching* where any subproblem is divided in two smaller, mutually-exclusive but collectively exhaustive (MECE) subproblems. In the case of 0-1 ILPs this branching heuristic is simply given by setting a certain variable to 1 for the first subproblem and to 0 to obtain the second. Of course, the corresponding search space regions are disjoint, as this specific variable cannot occupy both values at the same time, and their union is again the full search space, as the variable can only have value 1 or 0

[5] This is to be confirmed in Part III.

in the case of binary ILPs. Recalling what we said about the worst-case complexity of Branch and Bound in Section 2.1.1, employing binary branching means that the run time of the algorithm is at worst of the order $\mathcal{O}(M2^d)$. A suitable example is, in fact, the Knapsack Problem (cf. Section 1.3) where setting a variable to 1 or 0 is equivalent to including it in or excluding it from the knapsack, respectively. The second strategy is called *wide branching* and - as you can imagine - is characterized by the generation of more than two children, i.e. $r > 2$ in Line 5 of Algorithm 1. However, the corresponding subproblems do not necessarily have to be MECE as in the case of binary branching, meaning that it is generally possible to arrive at the same subproblems following several different paths of nodes in the search tree. Instead, the typical heuristic associated with wide branching selects one out of a set of different options and accordingly fixes the corresponding variable to value 1 or 0, depending on the definition of the variables. A wide branching could e.g. be naturally set up for the *Max Independent Set Problem* - searching for the largest set of vertices in a graph such that no two of them are adjacent - by forming the set of children in each partition procedure as the list of currently unspecified vertices in the graph with each subproblem being interpreted as including the corresponding vertex in the independent set. As the number of "open" vertices decreases in each iteration of the algorithm, this example directly implies that wide branching does not make any statement about the exact amount of children generated; this number may actually vary throughout the algorithm run, triggered by certain criteria. The example of Max Independent Set also shows how wide branching can be used to diminish the size (the height) of the search tree: Assuming the vertices are handled in proper order, including only the k^{th} vertex in the independent set, with $1 < k \lesssim |V|$, requires $k - 1$ branching steps using branching strategy while wide a branching can decide on it immediately, see Fig. 2.2.



(a) Binary branching. **(b)** Wide branching.

Figure 2.2.: Binary branching compared to wide branching for an exemplary problem formulated using the binary variables $z \equiv z_1 \cdots z_N$ with $1 < k \lesssim N$.

To sum up, the related research question to think of when constructing a B&B algorithm is something like "How should the branching be set up in order to generate the smallest number of unhelpful subproblems?".

2.1.4. Common Searching Strategies

In fact, almost any searching strategy can be classified into one of four categories. Each of them, together with their distinguishing characteristics shall now be briefly presented.

One of the most frequently used strategies (not even only in Branch and Bound) is called *depth-first search (DFS) strategy*. In a nutshell, the DFS strategy first explores a path to the end (i.e. until a leaf is reached) before a new one is opened, and backtracks to the closest parent node that still has unexplored subproblems, meaning that a branch starting from the root of the tree is completed before a new one is investigated. There are some advantages of the DFS strategy that can be immediately inferred from this high-level definition: First of all, it is pretty easy to implement for nearly all problems - no matter how different they may be, as it does not rely on the specific problem structure. Moreover, the height of the tree only increases by proceeding from a node to one of its direct children, meaning that information about the parent node may be used in the next step of exploration as a kind of starting point (other searching strategies would need to store these information in memory in order to achieve the same) [Mor+16]. Memory is actually a good keyword for turning to the third and probably most appreciated advantage of the DFS strategy: Since the heuristic does not induce huge jumps in the search tree, it is capable of outperforming most of the other searching strategies on the level of memory requirements. More specifically, the algorithm here only needs to keep track of the path from the root of T to the current subproblem at hand together with the index of the previously explored node: Either there is a child remaining and thus selected for exploration or the algorithm backtracks to the ancestor associated with the stored index and proceeds from there in the same fashion. However - as always - these benefits of the DFS strategy do not come without cost. In particular, not referring to the structure of the problem may also be interpreted negatively, as it can mean that the algorithm spends a large amount of time in search space regions where there is nothing to gain, simply by being too desperate to give up the current try and start a new one. Another problem can be found in a bit more of an edge-case situation, namely when the tree structure is largely unbalanced: Even though optimal solutions may be located close to the root of T, implementations of the DFS heuristic as described above are especially not sensible to this property and could, with a bit of bad luck, become ensnared into tedious investigations of long poor paths.

Analogous to the two branching strategies discussed in Section 2.1.3, there is also a counterpart of the depth-first search strategy, namely the *breadth-first seach (BrFS) strategy*. The defining property of the BrFS heuristic to induce the B&B algorithm to first explore all nodes on the same tree level before increasing the depth by proceeding with the exploration of children imparts it its name. This second type of searching strategy is also not tied to the structure of the problem in question and therefore, of course, features the same advantage-disadvantage pair as the DFS strategy, namely that it can be implemented in a simple way one one hand, while an algorithm equipped with the BrFS strategy is not capable of leveraging the information already provided by the respective problem formulation on the other. A true virtue of BrFS clearly is that it always finds the optimal solution that is closest to the root of the tree. Moreover, what was named as a drawback of the DFS strategy is here speaking in favor of the BrFS strategy: As it finishes the investigation of a tree layer first before taking children into account, it is naturally performing well on unbalanced search trees. However, full solutions - especially in our case of binary ILPs with simplified node generation - can only be found at larger depths. This in turn also means that the probability for a path in the tree to already be prunable at small depth is not that high. Hence, the number of nodes to be stored for exploration at a later stage is often quite high, implying above-average memory requirements for BrFS. This is the reason why the BrFS strategy is generally considered inefficient and why it is, thus, usually not used in the context of Branch and Bound [Mor+16].

The next searching strategy to be introduced is the first to be a bit more involved or, in other words, that is customizable by design. The *best-first search (BFS) strategy* is linked to a *measure-of-best function* $\mu : 2^{\mathbf{Z}} \to \mathbb{R}$ (recall that $\mathbf{Z} = \{0,1\}^N$ in our case), returning a value $\mu(S)$ for any newly generated subproblem $S \in T$. The BFS heuristic is then given by selecting that subproblem out of the list of unexplored nodes as the next to explore that optimizes the measure-of-best function μ. This is, it picks the subproblem with the smallest value of μ in the case of a minimization problem and, accordingly, that one with the largest μ value if the problem at hand asks for maximizing the objective function. Hence, as you can see, it requires a list of all unexplored nodes stored in memory together with their corresponding measure-of-best value as a key in order to make the above definition of the BFS strategy work in practice. In fact, there is a multitude of conceivable options for the measure-of-best function μ. Among these, a very natural choice is given by lower or upper bounds on the objective function, depending on the type of the optimization problem, especially making the BFS strategy comparable to the DFS and BrFS strategies. Obviously, because the BFS heuristic does not enforce a whole path or layer to be fully explored before going over to a possibly more promising region of the search space, the main advantage of BFS is that it generally finds good and optimal solutions earlier in the process of exploring the tree. On the other hand, just like for BrFS, the huge memory requirements raised by the need to keep track of all unexplored nodes at any time can

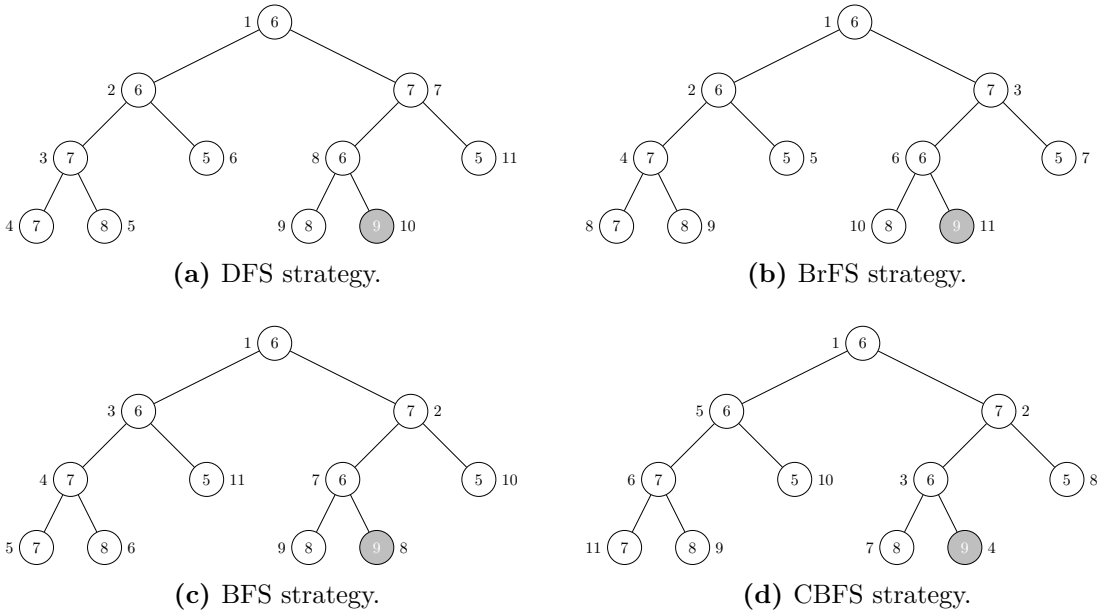
be seen as a drawback of the strategy. Beyond that, if the choice of μ causes that many subproblems in T whose measure-of-best value equals the optimal objective function value, i.e. $\mu(S) = C(z^*)$ for an optimal solution z^*, it could happen that it takes an algorithm employing the BFS strategy comparably long to actually arrive at an optimal solution due to being delayed in middle regions of the search tree [Mor+16].

The last strategy to be presented here is a hybrid heuristic made up by the DFS and the BFS strategy. It is called *cyclic best-first search (CBFS) strategy* and also based on a measure-of-best function $\mu : 2^{\mathbf{Z}} \to \mathbb{R}$. The main idea now is to not store all unexplored subproblems in a common dictionary with the keys given by the μ values but instead divide the open nodes into different classes, also called *contours*, such that newly generated nodes are assigned to distinct dictionaries according to certain rules. Then, instead of moving forward vertically or horizontally in the tree or jumping to the node with the best μ value, the CBFS strategy repeatedly iterates through the non-empty contours and every time selects the best subproblem according to the measure-of-best function. The contours can be understood to group subproblems that are similar or comparable in some sense (determined by the contouring rules). In this regard, the measure-of-best function then establishes a local ranking among every contour. The cycling between the different contours aims at weeding out the disadvantageous behavior of DFS to not overcome a path even if it is unpromising. On the other hand, it ensures that no region of the search space is overlooked by frequently entering each class of related subproblems. One common example for a quantity according to which nodes are categorized in contours is the depth in the tree T. In the case of depth-based contours, the inheritance of the DFS strategy can be clearly recognized, as each cycle arrives at a leaf of the search tree before starting from the top again. There are two important results about the CBFS strategy that make sense to be mentioned; both have been shown by Morrison et al. [Mor+17]. The first provides an upper bound on the number of nodes explored by CBFS for which the generation of children cannot be avoided: It is at maximum proportional to the number of subproblems explored by BFS when using the same measure-of-best function, the same branching strategy and the same pruning rules. Intuitively, this can be seen as follows: During each cycle through the non-empty contours, at least one node in the whole pass would have also been selected for exploration by the BFS strategy until an optimal solution is found. The second states that for any searching one can find a set of contour rules such that the associated CBFS strategy explores the same sequence of subproblems. This is, the CBFS strategy can be interpreted as a generalization of all other searching strategies, which ultimately means that it would have been sufficient to only introduce the CBFS heuristic and derive any of the other three strategies discussed here from it.

Note that all of these strategies, except for BrFS, can be further refined by specifying a rule according to which ties are broken. In the case of BFS and CBFS, ties of course refer to the measure-of-best function which is a direct component of the corresponding

searching strategies. As mentioned above, too many ties (in terms of the optimal objective-function value) can indeed have a negative impact on the runtime of a B&B algorithm equipped with the BFS strategy. However, ties can become important likewise when using the DFS heuristic, depending on the pruning rules chosen in the concrete algorithm design. A common example is given by exploiting bounds on the objective function to decide on pruning; matching values of the objective function are not a rarity in this setting (see also Fig. 2.3). Therefore, the tie breaking rules may also influence the performance of an DFS-algorithm non-negligibly.

Fig. 2.3 conclusively provides an illustration of how (different) the discussed searching strategies can perform on the same problem instance, assuming the same branching strategy and equal pruning rules.



(a) DFS strategy.

(b) BrFS strategy.

(c) BFS strategy.

(d) CBFS strategy.

Figure 2.3.: Exemplary instance of a 3-variable maximization problem for which the four discussed searching strategies are applied, resulting in four B&B versions differing in the order in which the nodes are explored. For the sake of comparability, the branching strategy and the pruning rules are chosen equally in Figs. 2.3a to 2.3d. In particular, binary branching is chosen over wide branching. The node colored in gray corresponds to the optimal solution. The numbers inside the nodes represent lower bounds on the optimal objective function value (except for those in the last tree level, there they denote the actual value of the corresponding leafs) while those outside indicate the exploration order determined by the respective searching strategy. These lower bounds serve as measure-of-best values in for BFS and CBFS in Figs. 2.3c and 2.3d with ties being broken randomly. On the other hand, DFS and BrFS in Figs. 2.3a and 2.3b are equipped with the heuristics of preferring to move left in the tree and passing through each layer from left to right, respectively.

2.2. QAOA

QAOA differs from Branch and Bound (cf. Section 2.1) in two main respects, namely that it is a quantum (or quantum-classical) algorithm and that it only approximates the optimal solution of the problem at hand. In fact, as we will see, while QAOA is theoretically converging to an optimal solution, actually occupying this value is not possible in practice. The first version of QAOA as an algorithm to tackle optimization problems was proposed by Farhi, Goldstone, and Gutmann [FGG14]. For an even more rigorous mathematical treatment of the three components dealt with in this section I would refer the reader to [Bin22].

2.2.1. Quantum Adiabatic Evolution

QAOA is itself based on another quantum algorithm that was initially presented almost 15 years before, also by Farhi et al. [Far+00]. The idea behind the so-called *quantum adiabatic algorithm (QAA)* is to generate optimal solutions to optimization problems via an adiabatic evolution of extremal eigenstates of a time-dependent Hamiltonian.

The QAA in turn is based on the adiabatic theorem, stating that the eigenspaces of an operator are preserved when evolving slowly and for a sufficiently long time according to a time-varying Hamiltonian (sometimes also called perturbation) unless they are mutually intersecting and the time-variation of the Hamiltonian itself is sufficiently smooth. Its original formulation in that shape goes back to Born and Fock [BF28]. This shall now be exploited as follows: In Section 1.2.2 we already saw how the objective function of a problem, for which an optimal solution is wanted, translates to a Hamiltonian for which an extremal eigenstate is searched in the quantum mechanical context (see especially Eq. (1.2.6)). From now on we will assume w.l.o.g. that our optimization problem is a maximization problem (recall that we discussed how to transform between a minimization and a maximization problem at the end of Section 1.2.1), meaning that we are aiming for a highest-energy eigenstate (or its associated eigenvalue) of the corresponding objective Hamiltonian C. The idea then is to introduce a second Hamiltonian B for which a highest-energy eigenstate is, in contrast, known in advance and, moreover, easy to construct. Leveraging the adiabatic theorem to map this state, chosen as a starting point, to a highest-energy eigenstate of C (i.e. an optimal solution) then solves our problem. The time-varying Hamiltonian performing the transition $B \mapsto C$ is chosen by Farhi et al. [Far+00] as the simple linear interpolation

$$H(t) := \left(1 - \frac{t}{T}\right) B + \left(\frac{t}{T}\right) C \quad , \, t \in [0, T], \tag{2.2.1}$$

obeying $H(0) = B$ and $H(T) = C$. As always, the unitary time evolution generated by $H(t)$, denoted by $U(t)$, is the solution of the Schrödinger equation

$$\frac{d}{dt}U(t) = -iH(t)U(t) \quad , \; t \in [0, T] \tag{2.2.2}$$

(assuming $\hbar = 1$). It can be written in the form

$$U(t) = \lim_{n\to\infty} \prod_{j=0}^{n} e^{-iH(j\frac{t}{n})\frac{t}{n}}. \tag{2.2.3}$$

As $H(t)$ is obviously sufficiently smooth, the adiabatic theorem implies that, assuming the initial state is chosen as described above, in the limit $T \to \infty$ we obtain an optimal solution if B satisfies that there is a non-vanishing gap between the highest-energy eigenstate and the rest of its spectrum. However, waiting an infinite amount of time is generally not a good approach when designing an algorithm that shall work in practice. A simple heuristic how to avoid these complications it to just abort the evolution with $U(t)$ after a sufficiently large time T in order to get a state that is close to the desired highest-energy eigenstate of C. So, the QAA is made up by three steps: (1) preparing the initial state on a quantum computer, (2) applying $U(t)$ up to certain finite time T, then called the *quasi-adiabatic evolution*, and (3) repeatedly measuring the final state in the computational basis (cf. Section 1.2.2) to obtain a distribution of optimal solution approximations. However, a feasible choice of T - e.g. driven by a certain specified error not to be exceeded by the outcome - is not possible without information about the minimum spectral gap, $g_{\min}$, between the largest eigenvalue of C and the next; Farhi et al. [Far+00] argue that the magnitude of T is mainly governed by $g_{\min}^{-2}$.

What remains to be answered is the question how the auxiliary Hamiltonian B could be chosen. Applying the adiabatic theorem to the highest-energy eigenstates, the preservation of eigenspaces also means that the geometric multiplicity[6] of the largest eigenvalue is an invariant quantity of the adiabatic evolution. Hence, as C is given by the problem, B must be chosen such that the geometric multiplicty of its largest eigenvalue matches with that of an optimal solution. Problematic now is that this property of C is in general not known, meaning that B can not be engineered on a sound basis. Nevertheless, the following one has emerged as emerged as a popular choice:

$$B = \sum_{n=1}^{N} \sigma_n^x \tag{2.2.4}$$

with

$$\sigma_n^x \equiv \underbrace{\mathbb{1} \otimes \cdots \otimes \mathbb{1} \otimes \underbrace{\sigma^x}_{n} \otimes \mathbb{1} \cdots \otimes \mathbb{1}}_{N}$$

[6]The *geometric multiplicity* of an eigenvalue is the dimension of its associated eigenspace.

where $\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ denotes the *first Pauli matrix* and N the problem size (cf. Definition 1.6). One can show that B as in Eq. (2.2.4) has no non-trivial invariant coordinate subspace, meaning that apart from 0 and $\mathbb{q}^N$ there is no $\mathcal{I} = \text{span}\{|j\rangle : j \in I \subseteq \mathbf{N}\}$ satisfying $B(\mathcal{I}) \subseteq \mathcal{I}$. Therefore, B is said to *completely non-diagonal*. Using the knowledge that σ_n^x has eigenstates $|+\rangle_n$ and $|-\rangle_n$, where $|\pm\rangle = 1/\sqrt{2}(|0\rangle \pm |1\rangle)$, with corresponding eigenvalues 1 and -1, respectively, the highest-energy eigenstate of B as in Eq. (2.2.4) is obtained by tensoring up all the single highest-energy eigenstates:

$$|s\rangle := |+\rangle^{\otimes N} := \bigotimes_{n=1}^{N} |+\rangle_n = \bigotimes_{n=1}^{N} \frac{1}{\sqrt{2}}(|0\rangle_n + |1\rangle_n) = \frac{1}{\sqrt{2^N}} \sum_{z \in \{0,1\}^N} |z\rangle \,. \tag{2.2.5}$$

We will revisit the following in the actual implementation of the QAOA (cf. Part III): In order to obtain a working algorithm, the QAA first needs to prepare this initial state $|s\rangle$ with any qubit being initialized in state $|0\rangle$. This is actually pretty simple here where $|s\rangle$ is given as in Eq. (2.2.5). Using the action of a Hadamard gate

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \,, \tag{2.2.6}$$

we find that $|s\rangle$ may be generated from the zero-state $|0, ..., 0\rangle$ by a unitary U_S via

$$|s\rangle = |+\rangle^{\otimes N} = \bigotimes_{n=1}^{N} |+\rangle_n = \bigotimes_{n=1}^{N} H|0\rangle_n = H^{\otimes N}|0\rangle^{\otimes N} =: U_S|0\rangle^{\otimes N} \,. \tag{2.2.7}$$

Apart from that, a high-level reasoning that the application of the adiabatic theorem with respect to[7] Eq. (2.2.1) for this particular choice of B works as intended is given by Farhi, Goldstone, and Gutmann [FGG14]. However, this only holds for COPs with exactly one optimal solution, meaning that the largest eigenvalue of C needs to be non-degenerate. A more fundamental convergence proof without a spectral gap condition[8], showing that this is actually not a necessary requirement, is provided by Binkowski [Bin22, S.3.1] and stated there as follows:

Theorem 2.1 (Convergence of the QAA)**.** *Let F_{opt} be the optimal solution space (cf. Definition 1.13) of an unconstrained maximization problem COP in the sense of Definition 1.6 with objective Hamiltonian C defined via Eq.* (1.2.6)*. Then,*

$$\lim_{T \to \infty} U(T)|s\rangle \in F_{\text{opt}}$$

for the quasi-adiabatic evolution $U(T)$ induced by the Hamiltonian $H(t)$ as in Eq. (2.2.1) *and $|s\rangle$ defined in Eq.* (2.2.5)*.*

[7] Referred to as "w.r.t." hereafter.

[8] The spectral gap is supposed to approach 0 as $t \to T$ in the case of multiple optimal solutions.

In fact, the complete non-diagonality of B turns out to be a necessary property of the initial Hamiltonian. Not only is it used in the proof of Theorem 2.1, Binkowski [Bin22] further shows that there always exists an unconstrained maximization problem for which the convergence fails in case that the starting Hamiltonian has at least one proper invariant coordinate subspace, even when the restriction of the initial state being given by $|s\rangle$ as in Eq. (2.2.5) is relaxed.

2.2.2. Quantum Approximate Optimization Algorithm

Beyond what was previously mentioned, the first QAOA version can be considered as a refinment of the QAA. Its development was motivated by the lack of a construction component reliably determining the quality of the returned approximations or, to put it differently, a parameter by which it can be regulated.

Recall how the adiabatic evolution, generated by the interpolating Hamiltonian in Eq. (2.2.1), could be expressed in Eq. (2.2.3). The following result for general matrices will be of great usage to rewrite this expression.

Theorem 2.2 (Trotter product formula)**.** *For two arbitrary $n \times n$ real or complex matrices X and Y,*

$$e^{X+Y} = \lim_{m\to\infty} \left(e^{\frac{X}{m}} e^{\frac{Y}{m}}\right)^m$$

where e^A denotes the matrix exponential of a diagonal matrix A.

Proof. A proof can e.g. be found in [Hal15, Thm.2.11]. □

Now applying the Trotter product formula to the unitary time evolution induced by $H(t)$ as the weighted sum of Hamiltonians B and C yields

$$\begin{aligned} U(t) &= \lim_{n\to\infty} \prod_{j=0}^{n} e^{-i\left(\left(1-\frac{j}{n}\frac{t}{T}\right)B + \left(\frac{j}{n}\frac{t}{T}\right)C\right)\frac{t}{n}} \\ &= \lim_{n\to\infty} \prod_{j=0}^{n} \lim_{m\to\infty} \left(e^{-i\left(1-\frac{j}{n}\frac{t}{T}\right)B\frac{t}{nm}} e^{-i\left(\frac{j}{n}\frac{t}{T}\right)C\frac{t}{nm}}\right)^m . \end{aligned}$$

Of course, we are interested in the quasi-adiabatic evolution at time $t = T$, assuming

that T is chosen sufficiently large:

$$\begin{aligned} U(T) &= \lim_{n\to\infty} \prod_{j=0}^{n} \lim_{m\to\infty} \left(e^{-i\left(1-\frac{j}{n}\right)B\frac{T}{nm}} e^{-i\left(\frac{j}{n}\right)C\frac{T}{nm}} \right)^m \\ &= \lim_{n\to\infty} \prod_{j=1}^{n} e^{-i\beta_j B} e^{-i\gamma_j C} \\ &=: \lim_{n\to\infty} \prod_{j=1}^{n} U_B(\beta_j) U_C(\beta_j) \end{aligned}$$

for suitably chosen β_j, γ_j with $j \in \{1, ..., N\}$, and

$$U_B(\beta_j) := e^{-i\beta_j B} \quad \text{and} \quad U_C(\gamma_j) := e^{-i\gamma_j C}; \tag{2.2.8}$$

p is called the *depth* of the QAOA. Again, implementing an infinite series of these pairs of unitary operators is not a feasible approach for any algorithm on any kind of computer. Instead, we introduce a parameter p and use that, for $p < \infty$ sufficiently large,

$$U(T) \approx \prod_{j=1}^{p} U_B(\beta_j) U_C(\gamma_j). \tag{2.2.9}$$

This can now be applied to the initial state $|s\rangle$ defined in Eq. (2.2.5), yielding a state

$$|\beta, \gamma\rangle_p := \left(\prod_{j=1}^{p} U_B(\beta_j) U_C(\gamma_j) \right) |s\rangle = \prod_{j=1}^{p} U_B(\beta_j) U_C(\gamma_j) |s\rangle \tag{2.2.10}$$

that is determined by the $2p$ parameters $\beta \equiv (\beta_1, ..., \beta_p)$ and $\gamma \equiv (\gamma_1, ..., \gamma_p)$. Let me say a word about the domain of the single parameters, which can actually be restricted to not be the full $\mathbb{R}$. As we are only working with integer-valued objective functions (cf. Definitions 1.6 and 1.7), Eq. (1.2.6) implies that C has integer eigenvalues solely. The same holds true for B as in Eq. (2.2.4), since each single σ^x-term has eigenvalues ± 1. Therefore, the parameters may be restricted to the domains $\beta_j \in [0, \pi)$ and $\gamma_j \in [0, 2\pi)$ for all $j \in \{1, ..., p\}$, which is why they are mostly referred to as *angles*.[9] For what comes in the near future, it is worth taking a look at how the unitaries U_B and U_C act on single computational basis states. Since C is diagonal in the computational basis by design (cf. Eq. (1.2.6)),

$$U_C(\gamma_j) |z\rangle = e^{-i\gamma_j C} |z\rangle = e^{-i\gamma_j C(z)} |z\rangle$$

for $z \in \{0, 1\}^N$, i.e. U_C adds a phase to any state based on the respective value of the objective function by linear extension. Therefore, C is called *phase separator* and

[9] Further restricting β to the interval $[0, \pi)$ instead of $[0, 2\pi)$ is possible thanks to its spectrum being symmetric around 0.

U_C accordingly *phase separation unitary*. To see the effect of U_B on a computational basis state is a bit more involved. First, notice that, due to $[\sigma_n^x, \sigma_m^x] = 0 \quad \forall\, n, m \in \{1, ..., N\}$,

$$U_B(\beta_j) = e^{-i\beta_j B} = e^{-i\beta_j \sum_{n=1}^{N} \sigma_n^x} = e^{-i\beta_j \sigma_1^x - ... - i\beta_j \sigma_N^x} = e^{-i\beta_j \sigma_1^x} \cdots e^{-i\beta_j \sigma_N^x} = \prod_{n=1}^{N} e^{-i\beta_j \sigma_n^x}.$$

Known from elementary quantum mechanics classes, $e^{-i\beta_j \sigma_n^x}$ is a rotation operator, here rotating the n^{th} qubit around the x-axis of "its" Bloch sphere by β_j. Implied by $(\sigma^x)^2 = \mathbb{1}$, meaning that $(\sigma^x)^k = \mathbb{1}$ if $k \in \mathbb{N}$ is even and $(\sigma^x)^k = \sigma^x$ if k is odd, we can make use of the identity

$$e^{-i\beta_j \sigma_n^x} = \cos(\beta_j)\mathbb{1} - i \sin(\beta_j)\sigma_n^x \tag{2.2.11}$$

to obtain

$$\begin{aligned}
U_B(\beta_j) &= \prod_{n=1}^{N} e^{-i\beta_j \sigma_n^x} = \prod_{n=1}^{N} \left(\cos(\beta_j)\mathbb{1} - i \sin(\beta_j)\sigma_n^x\right) \\
&= \cos(\beta_j)^N \mathbb{1} \\
&\quad - i \cos(\beta_j)^{N-1} \sin(\beta_j)\sigma_1^x - ... - i \cos(\beta_j)^{N-1} \sin(\beta_j)\sigma_N^x \\
&\quad - \cos(\beta_j)^{N-2} \sin(\beta_j)^2 \sigma_1^x \sigma_2^x - ... - \cos(\beta_j)^{N-2} \sin(\beta_j)^2 \sigma_1^x \sigma_N^x \\
&\quad + ... + (-i)^N \sin(\beta_j)^N \sigma_1^x \cdots \sigma_N^x \\
&= \sum_{k=0}^{N} (-i)^k \cos(\beta_j)^{N-k} \sin(\beta_j)^k \sum_{\substack{l_1+...+l_N=k \\ l_1,...,l_N \in \{0,1\}}} \prod_{n=1}^{N} (\sigma_n^x)^{l_n}.
\end{aligned}$$

where $\mathbb{1}$ here means the identity operator on all N qubits[10] and $(\sigma_n^x)^0 = \mathbb{1}$. Recall that the single σ_n^x flips the states $|0\rangle_n$ and $|1\rangle_n$, i.e. $\sigma_n^x |0\rangle_n = |1\rangle_n$ and $\sigma_n^x |1\rangle_n = |0\rangle_n$, respectively. Having that in mind, we see that especially the last sum of σ^x's in the above expression implies that the application of U_B to an arbitrary state results in a superposition of all computational basis states, as it covers any possible combination of σ^x operators on the N positions, meaning that the qubits in the computational basis representation of our state are flipped in any possible way (including no flip at all). Therefore, B is called *mixer* and U_B accordingly *mixing unitary*.

The underlying concept of what we have seen about QAOA so far is that the alternating application of phase separation and mixing unitaries prefers some states over others,

[10] It may indeed be a bit unfortunate that the same symbol $\mathbb{1}$ can represent the identity on different numbers of qubits depending on the situation; however, it should always be clear from the context what it refers to.

meaning that their amplitudes in the final state $|\beta, \gamma\rangle_p$ will be larger than those of others. The hope then is that these states are the desired ones, namely those corresponding to optimal or near-optimal solutions. Especially the approximation Eq. (2.2.9) gives rise to the name *Quantum Approximate Optimization Algorithm* - or QAOA in short. In contrast to the QAA (cf. Section 2.2.1), the choice of initial Hamiltonian B is not intended to be configurable in the Quantum Approximate Optimization Algorithm; instead it is predefined as in Eq. (2.2.4).[11] Based on this, Eq. (2.2.10) justifies us to think of the QAOA as a parametrized version of the QAA with fixed B. We evaluate the obtained state $|\beta, \gamma\rangle_p$ via the expectation value of the objective Hamiltonian as

$$E_p(\beta, \gamma) := {}_p\langle\beta, \gamma| C |\beta, \gamma\rangle_p \tag{2.2.12}$$

and set

$$E_p^* = \max_{\beta,\gamma} E_p(\beta, \gamma). \tag{2.2.13}$$

As increasing the depth p is adding degrees of freedom by introducing more angles, the result for E_p can only become better, meaning that

$$E_{p+1}^* \geq E_p^*. \tag{2.2.14}$$

Even further, we find:

Corollary 2.3. *Let C be the objective function of an unconstrained maximization problem $COP = (N, \{c_j\}_{j=1}^a, C_j{}_{j=1}^a, \emptyset, \max)$ in accordance with Definitions 1.6 and 1.7. Then,*

$$\lim_{p\to\infty} E_p^* = \max_{z\in\{0,1\}^N} C(z). \tag{2.2.15}$$

Proof. This result is an immediate consequence of Theorem 2.1. □

This result, together with Eq. (2.2.9), determines the impact the depth has on the quality of the approximation, namely that it improves with increasing value of p. In contrast, the success probability $|\langle z^*| U(T) |s\rangle|^2$ of the QAA to arrive at an optimal solution $z^* \in \{0,1\}^N$ at time $t = T$ can in general not found to be a monotonic function of T, see e.g. [Cro+14, fig.2]. Assuming a fixed depth p, Eq. (2.2.13) implies that the task of maximizing the objective function is thus translated to optimizing the $2p$ angles β, γ. While we are on the comparison of QAOA and QAA: An example where the QAA fails but the QAOA succeeds is described in [FGG14, S.VI]; the former is there trapped in a false optimum in case of subexponential computing times.

[11] This property of the first QAOA to start from a fixed mixer will actually turn out to be crucial for the question why this first version is generally not sufficient.

Farhi, Goldstone, and Gutmann [FGG14] propose some strategies aiming to find good angles. The first is highly problem-specific as it exploits the structure of the investigated problem in order to express E_p for a fixed value of p in a form that can be evaluated on a classical computer whose resource requirements do not necessarily grow with the problem size. However, as you may imagine, this is not a very generally applicable approach. Therefore, the second method is the more widely used one: It embeds Eq. (2.2.10) in the setting of a variational algorithm. The resulting QAOA scheme for a general unconstrained maximization problem with objective function C and depth d is depicted as pseudocode in Algorithm 2.

Algorithm 2: QAOA(COP, p)

1 Set $i = 0$ and initialize $\beta^{(0)}, \gamma^{(0)}$
2 Evaluate $E_p = E_p\left(\beta^{(0)}, \gamma^{(0)}\right)$ according to Eq. (2.2.12) via repeated measurements
3 **while** *termination condition not satisfied* **do**
4 Set $|\psi\rangle = |0\rangle^{\otimes N}$ (initialization of the qubits)
5 $|\psi\rangle \leftarrow |s\rangle = U_S\,|\psi\rangle$ (preparation of the initial state)
6 Set $j = 0$
7 **while** $j < p$ **do**
8 $|\psi\rangle \leftarrow U_C\left(\gamma_j^{(i)}\right)|\psi\rangle$ (application of the phase separation unitary)
9 $|\psi\rangle \leftarrow U_B\left(\beta_j^{(i)}\right)|\psi\rangle$ (application of the mixing unitary)
10 $j \leftarrow j + 1$
11 Evaluate $E_p^{(i)} = E_p\left(\beta^{(i)}, \gamma^{(i)}\right)$ via repeated measurements
12 **if** $E_p^{(i)} > E_p$ **then**
13 $E_p \leftarrow E_p^{(i)}$ (update of best expectation value found so far)
14 $i \leftarrow i + 1$
15 **return** E_p

The transition to the next iteration step in Line 14 of Algorithm 2 encapsulates an important characteristic of the Quantum Approximate Optimization Algorithm: The update of the parameter sets $\beta^{(i)}, \gamma^{(i)} \mapsto \beta^{(i+1)}, \gamma^{(i+1)}$ according to a classical optimization routine turns the QAOA into a hybrid quantum-classical algorithm - more specifically, a VQA. The termination condition in Line 3 depends on the specific chosen classical optimizer; it can e.g. be based on a maximum number of iterations or a minimum deviation between the results of two successive iterations to not be undercut. For instance, Farhi, Goldstone, and Gutmann [FGG14] propose to choose the angles (β, γ) from a grid on the set $[0, \pi)^p \times [0, 2\pi)^p$, using a maximum number of iteration steps that is screwed sufficiently upwards such that the grid is fine enough to not miss any peaks of the objective function on the two intervals; finally, the maximum one out

of the finite sequence of expectation values computed during the run of Algorithm 2 in Line 11 is returned. However, I can already reveal that we will employ a different classical optimization method. What we should learn from Algorithm 2 is the following: Beyond the quantum mechanical part (i.e. the application of phase separation and mixing unitaries) and the specified depth, the performance of the QAOA depends to a large extent on classical optimization properties, namely the initial guesses of angles, $\beta^{(0)}$ and $\gamma^{(0)}$, the chosen optimization method and the termination condition related thereto. Nevertheless, what we said in the context of Branch and Bound (cf. Section 2.1) also applies for the QAOA: This is still a quantum physics thesis, which is why resources will not be allocated excessively to the task of classical parameter optimization even despite its great influence on the QAOA performance.

Finally, we can state the analogue of Theorem 2.1 for the Quantum Approximate Optimization Algorithm:

Theorem 2.4 (Convergence of the Quantum Approximate Optimization Algorithm)**.** *Let F_{opt} be the optimal solution space (cf. Definition 1.13) of an unconstrained maximization problem COP in the sense of Definition 1.6 with objective Hamiltonian C defined via Eq.* (1.2.6)*. Suppose U_B and U_C to be given as in Eq.* (2.2.8) *with B defined via Eq.* (2.2.4)*. Then, for any $\varepsilon > 0$ there are finitely many angles $\beta = (\beta_1, ..., \beta_p)$ and $\gamma = (\gamma_1, ..., \gamma_p)$, $p < \infty$, such that the final state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p$ defined by Eq.* (2.2.10) *satisfies*

$$\operatorname{dist}\left(|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p, F_{\text{opt}}\right) < \varepsilon$$

where dist denotes the smallest distance between $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p$ and any state in F_{opt}.

Proof. For a rigorous and constructive proof of this fundamental theorem I would again refer to Binkowski [Bin22, S.3.2]. □

Theorem 2.4 states that the Quantum Approximate Optimization Algorithm is able to approximate an optimal solution of our maximization problem to an arbitrary degree. However, there is, in fact, one important property of the QAOA that cannot be seen from Theorem 2.4 and is therefore worth to be stated separately as we will permanently exploit it in Part II: In combination with the found monotonicity of the expectation value (cf. Eq. (2.2.14)), Corollary 2.3 implies that the approximation returned by Algorithm 2 is always providing a lower bound on the actual optimal solution value.

A totally different path into investigating how the QAOA works is taken by Koßmann et al. [Koß+22]. Instead of proceeding in the probably more NISQ-friendly and intuitive direction of slowly increasing a shallow depth p, they start off considering the

asymptotic limit $p \to \infty$, where it turns out that the quantum part of Algorithm 2 may be properly analyzed in a differential geometric fashion using the tools of Lie theory (see e.g. [Hal15]). More specifically, the set of accessible states $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_{p\to\infty}$ can be found to form a differentiable manifold, on which each of the two families of unitary operators (phase separation and mixing) corresponds to a vector field. This geometric approach goes hand in hand with some nice visualizations for a single qubit of how moving along these vector fields on the surface of the qubit's Bloch sphere can lead from the initial state to one representing an optimal solution. Then, the authors step back from asymptotic depth to deep circuits for the sake of realizability and discuss how the description and the found properties need to be adapted.

Now comes the drawback of the QAOA version we have seen so far. Everything that was discussed in this chapter until this point was developed on the basis of an unconstrained optimization (maximization) problem. However, only the least real-world problems come with no constraints. One possible strategy to mitigate the discrepancy between expectation and ability is called *softcoding the constraints*, which transforms a constrained combinatorial optimization problem as in Definition 1.6 into an unconstrained one. In a nutshell, the idea of softcoding the constraints is to introduce penalties that are sufficiently large to ensure that violating a constraint is never favorable, especially in comparison to other clauses. These penalties are then used in place of the costs or profits when considering the constraints as clauses. When given a maximization problem with objective function C, we are, however, not able to figure out whether some softcoding of constraints has already been performed previously, meaning that Algorithm 2 does not distinguish between problems that are unconstrained in their nature and those that have just been transformed into such. Therefore, the Quantum Approximate Optimization Algorithm is also called *Softconstraint QAOA*. Crucial here is to note that the success of applying Algorithm 2 to a softcoded combinatorial optimization problem badly depends on the specific choice of penalties: If the chosen values are too small, the intentional separation between feasible and infeasible states might turn out to not be sufficiently pronounced. This can cause a bad approximation quality obtained by the Softconstraint QAOA, especially in the realistic case where constraint-violating bitstrings $z \in \{0,1\}^N$ correspond to very profitable objective-function values. Hence, it is an involved task to adjust the penalties such that the desired QAOA property of returning only lower bounds of the optimal solution value is persisted. On the other hand, too large penalty values may impede the optimization of the angles β and γ. Apart from this consideration, one thing is for sure: As no operation in Algorithm 2 completely eliminates any (unwanted) states and only adds different phases and mixes them through, the final state with optimal angles $|\beta^{(\mathrm{opt})}, \gamma^{(\mathrm{opt})}\rangle$ will not get around also containing non-vanishing amplitudes corresponding to infeasible states. Thus, no matter how good the approximation might be, this presence of infeasible states always introduces some magnitude of error in the result. The quantum-computing community was not willing to settle for that.

2.2.3. Quantum Alternating Operator Ansatz

Unfortunately, the Quantum Approximate Optimization Algorithm is generally not capable of solving hardconstrained optimization problems[12]. It is surprisingly simple to discover the reason for that: Feasibility (cf. Eq. (1.2.3)) is just not respected in any regard by the alternating application of phase separation unitaries, that solely rely on the objective function, and mixing unitaries, which are not related to the specific problem at all with B as in Eq. (2.2.4). Given a hardconstrained maximization problem COP with feasible solution space F and a specified depth p, a meaningful feasibility requirement for the QAOA would, for instance, be something like

$$|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k \in F \;\; \forall\, k \in \{0,...,p\}, \;\; \forall\, (\boldsymbol{\beta},\boldsymbol{\gamma}) \in [0,\pi)^k \times [0,2\pi)^k \quad \text{if} \quad |s\rangle \in F \tag{2.2.16}$$

with $|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k$ being defined according to Eq. (2.2.10), $|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_0 := |s\rangle$, and where $|s\rangle$ denotes the initial state[13], meaning that neither $U_C(\gamma)$ nor $U_B(\beta)$ maps outside the feasible subspace (as the angles (β,γ) are free to vary over the full intervals $[0,\pi)\times[0,2\pi)$). For C being diagonal in the computational basis by design in Eq. (1.2.6), $U_C(\gamma)$ naturally leaves F invariant. In contrast, as already mentioned, B is completely non-diagonal if chosen as defined in Eq. (2.2.4), implying that also $U_B(\beta)$ does not have any non-trivial invariant coordinate subspaces (except for $\beta = 0$ obviously), i.e. especially not F. Thus, the issue the Quantum Approximate Optimization Algorithm has with hardconstrained problems is located in the mixing procedure. As we just saw, while the complete non-diagonality of the mixer was necessary to guarantee convergence of the QAOA in the unconstrained case, it is remarkably responsible for the failure in the hardconstrained setting.

After we have tried to adapt the objective Hamiltonian C to incorporate constraints into the QAOA at the end of Section 2.2.2 and ended up not fully happy, the next logical step - in particular against the background of the above failure analysis - is to direct our attention to the mixer instead. It is intuitively clear that this approach will be more involved than the penalizing of the objective function; designing a mixer B such that an application of U_B creates a pure superposition of feasible states badly depends on the problem structure at hand, whereas implementing the QAOA with softcoded constraints according to Algorithm 2 is comparatively easy and can be improved by adjusting the penalties after the first results have been obtained. This also means that the underlying concept can only be sketched on a pretty generic level here, leaving most of the work for the concrete construction.

[12] Combinatorial optimization problems whose constraints were not softcoded into the objective function are called *hardconstrained*. The term is used to suggest that the problem shall be handled as is, namely with the constraints in place.

[13] Here not necessarily given by Eq. (2.2.5) but considered from a general perspective.

The idea of generalizing the Quantum Approximate Optimization Algorithm goes back to Hadfield [Had18]. Although our rationale is to tinker with the mixer, the first hurdle we stumble over when going through the steps in Algorithm 2 is that we do not know how the initial state $|s\rangle$ has to be prepared. Now where B is not a priori given by Eq. (2.2.4), $|s\rangle$ cannot simply be chosen as in Eq. (2.2.5) anymore. Of course, the same holds for the domain of the parameters $\boldsymbol{\beta}$; as the final mixers used in the end might have non-integer eigenvalues, we are no longer allowed to restrict $\beta \in [0, \pi)$, but instead need to generally enlarge the domain to $\beta \in \mathbb{R}$. In the understanding of Hadfield [Had18], the quantum part in the QAOA procedure consists of three main components that may be configured in order to properly address the problem in question: the initial state, the phase separation unitaries and the mixing unitaries. In [Had18, Ss.6.2.1], certain "design criteria" are formulated aiding in making choices on each of them by providing guidelines for sensible constructions. The initial state $|s\rangle$, for instance, shall be easy to implement, meaning that it can be prepared starting from the state $|0\rangle = |0, ..., 0\rangle$ (cf. Eq. (1.2.5)) without requiring a circuit depth[14] that scales too badly with the problem size. If neither a constant nor a logarithmic depth can be guaranteed, the initialization should be considered separately from the residual QAOA part according to Hadfield [Had18]. In case of the Quantum Approximate Optimization Algorithm, the initial state $|s\rangle = |+, ..., +\rangle$ (cf. Eq. (2.2.5)) can be seen, based on what we found about the action of U_B in Section 2.2.2 with B as in Eq. (2.2.4), to be generated by one additional, initial application of the mixing unitary as

$$|s\rangle = U_B(\beta_0) |0\rangle \tag{2.2.17}$$

for some additional angle $\beta_0 \in [0, \pi)$. For the mixing unitaries there are two expressive criteria in particular named in [Had18]: First, they shall respect Eq. (2.2.16), i.e. $U_B(\beta)$ should map feasible states to feasible states for any non-zero value of the parameter $\beta \in \mathbb{R} \setminus \{0\}$, meaning that U_B preserves the feasible subspace. The second criterion formulated for the mixing unitaries extends that demand in the sense that U_B is not only not leaving the feasible subspace invariant but rather exploring it; this is, for any two feasible states $|z_1\rangle, |z_2\rangle \in F$, there is some parameter value β^* and some positive integer $m > 0$ such that the m-times repeated application of the corresponding mixing unitary connects them, i.e. $|\langle z_2| (U_B(\beta^*))^m |z_1\rangle| > 0$. Note that in general $(U_B(\beta))^m \neq U_B(m\beta)$. Turning to the third and last component, one simple requirement for the phase separation unitary is listed in [Had18], namely that it is diagonal in the computational basis. Incorporating the concept discussed so far in Algorithm 2 (i.e. undefining the inital state, the phase separation unitaries and the mixing unitaries and making them free parameters of the algorithm) takes it to a new level of generality; the result is called *Quantum Alternating Operator Ansatz*, a shrewd invention by Hadfield [Had18] that allows to continue using the abbreviation

[14] Recall the depth of a quantum circuit describes the maximum number of consecutive gates on one and the same qubit.

QAOA. As the Quantum Approximate Optimization Algorithm with the particular choices made for the three components in Section 2.2.2 is clearly a special case of the Quantum Alternating Operator Ansatz, we will w.l.o.g. mean the latter when talking about QAOA from now on. In terms of a final remark, we can state that even though the phase separation unitaries are also listed as a customizable constituent of the algorithm in [Had18, S.6.2], it is admitted that $U_C(\gamma) = e^{-i\gamma C}$ as in Eq. (2.2.8) with objective Hamiltonian C defined via Eq. (1.2.6) is, up to a global phase that can be ignored, a good enough choice in almost every case, leaving merely the initial state $|s\rangle$ and the mixer B for investigation.[15] The Quantum Alternating Operator Ansatz is reworked and further elaborated in [Had+19]; the discussion is in particular extended by additional problems and examples, for which especially mixers are investigated.

Theorem 2.5 (Convergence of the Quantum Alternating Operator Ansatz)**.** *Let $F_{opt} \subseteq F$ be the optimal solution space (cf. Definition 1.13) of a constrained maximization problem COP in the sense of Definition 1.6 with objective Hamiltonian C defined via Eq.* (1.2.6) *and feasible subspace F (cf. Definition 1.12). Suppose U_B and U_C to be given as in Eq.* (2.2.22) *with a B that is chosen to obey the two design criteria by Hadfield [Had18], namely*

(i) $U_B(\beta)\,|z\rangle \in F$ *if* $|z\rangle \in F$ *and*

(ii) there is $\mathbb{N} \ni m > 0$ *and* $\beta^* \in \mathbb{R}$ *such that* $|\langle z_2|\,(U_B(\beta^*))^m\,|z_1\rangle| > 0$ *for any* $|z_1\rangle, |z_2\rangle \in F$.

Then, for any $\varepsilon > 0$ there are finitely many angles $\boldsymbol{\beta} = (\beta_1, ..., \beta_p) \in \mathbb{R}^p$ and $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_p) \in [0, 2\pi)^p$, $p < \infty$, such that the final state $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p$ defined by Eq. (2.2.10) *satisfies*

$$\operatorname{dist}\left(|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle_p, F_{\text{opt}}\right) < \varepsilon$$

where dist denotes the smallest distance between $|\gamma, \beta\rangle_p$ and any state in F_{opt}.

Proof. Once more I would refer to Binkowski [Bin22, S.3.3] for a proof. □

An interesting approach was proposed by Bärtschi and Eidenbenz [BE20], suggesting to not focus on the mixer as emphasized above, but to shift the effort to the preparation of the initial state instead. Note that this approach was developed specifically for hardconstrained optimization problems. The wording "preparation" is indeed chosen

[15] Note that this automatically allows to restrict the parameter γ to the interval $[0, 2\pi)$ again.

deliberately, as the desired initial state is always of the same shape, independent of the respective problem at hand: For a given COP, we want to yield

$$|s\rangle = |F\rangle := \frac{1}{\sqrt{|\text{feas(COP)}|}} \sum_{z \in \text{feas(COP)}} |z\rangle \tag{2.2.18}$$

with feas(COP) as in Definition 1.9, i.e. the goal is an equal superposition of all feasible states of our problem. $|F\rangle$ is well-defined for $\{0,1\}^N \supseteq \text{feas(COP)}$ having finite cardinality. The labeling of the state $|F\rangle$ is based on the knowledge that F is generated by those states $|z\rangle$ with $z \in \text{feas(COP)}$ (cf. Definition 1.12). Of course, the exact form of the initial state $|s\rangle$ will vary greatly from problem to problem. For example, in the case of an unconstrained (or softconstrained) optimization problem of size N, $|F\rangle$ just evaluates to the initial state $|+\rangle^{\otimes N} = 1/\sqrt{2}^N \sum_{z \in \{0,1\}^N} |z\rangle$ in Eq. (2.2.5) used in the original QAOA, since any state is feasible in the absence of (hard) constraints. Now starting from the state $|0\rangle = |0, ..., 0\rangle$ as in the Quantum Alternating Operator Ansatz by Hadfield [Had18], the initial state is prepared using a *state preparation unitary* U_S via

$$|F\rangle = |s\rangle = U_S |0\rangle . \tag{2.2.19}$$

As the title of the approach by Bärtschi and Eidenbenz [BE20] already suggests, figuring out how to construct U_S is central for this strategy. The phase separation unitaries in this approach are again simply given by $U_C(\gamma) = e^{-i\gamma C}$ as in Eq. (2.2.8) where C as always is the Hamiltonian corresponding to the objective function of the problem in the sense of Eq. (1.2.6). The pressing question thus is how to choose the mixer and, accordingly, the mixing unitaries. As already indicated, this will be based on the way we obtained our initial state. At this stage, recall the relation between the mixer and the initial state that we found in Section 2.2.1 to be necessary for leveraging the adiabatic theorem: In order to ensure that the quasi-adiabatic evolution, approximated by an alternating application of phase separation and mixing unitaries, converges to an optimal solution of our hardconstrained maximization problem with maximum objective function value, the process needed to be started in a highest-energy eigenstate of the initial Hamiltonian. Translating this to the situation here, we have to guarantee that $|F\rangle$ is such a highest-energy eigenstate of the chosen mixer. Bärtschi and Eidenbenz [BE20] propose to use the following mixer:

$$B = |F\rangle \langle F| , \tag{2.2.20}$$

i.e. the projector onto the state $|F\rangle$. Consequently, the mixing unitary is then given by

$$U_B(\beta) = e^{-i\beta B} = e^{-i\beta |F\rangle\langle F|}. \tag{2.2.21}$$

To see how that operator can be implemented using the state preparation unitary U_S,

write out that matrix exponential:

$$\begin{aligned}
U_B(\beta) = e^{-i\beta|F\rangle\langle F|} &= \sum_{k=0}^{\infty} \frac{(-i\beta)^k}{k!} (|F\rangle\langle F|)^k \\
&= \frac{(-i\beta)^0}{0!} (|F\rangle\langle F|)^0 + \sum_{k=1}^{\infty} \frac{(-i\beta)^k}{k!} \underbrace{|F\rangle\langle F|F\rangle \cdots \langle F|F\rangle \langle F|}_{k} \\
&= \mathbb{1} + \sum_{k=1}^{\infty} \frac{(-i\beta)^k}{k!} |F\rangle (\underbrace{\langle F|F\rangle}_{=1})^{k-1}) \langle F| \\
&= \mathbb{1} + \left(\sum_{k=1}^{\infty} \frac{(-i\beta)^k}{k!} \right) |F\rangle\langle F| = \mathbb{1} + \left(\sum_{k=0}^{\infty} \frac{(-i\beta)^k}{k!} - 1 \right) |F\rangle\langle F| \\
&= \mathbb{1} + \left(e^{-i\beta} - 1 \right) |F\rangle\langle F| \\
&= \mathbb{1} + \left(e^{-i\beta} - 1 \right) U_S |0\rangle\langle 0| U_S^\dagger \quad \text{by Eq. (2.2.18)} \\
&= U_S U_S^\dagger + U_S \left(e^{-i\beta} - 1 \right) |0\rangle\langle 0| U_S^\dagger \quad \text{by unitarity of } U_S \\
&= U_S \left(\mathbb{1} - \left(1 - e^{-i\beta} \right) |0\rangle\langle 0| \right) U_S^\dagger
\end{aligned} \tag{2.2.22}$$

Slightly differing from the nomenclature in [BE20], mixers of the form in Eq. (2.2.20) are called *Grover mixers*, as the middle part of the mixing unitaries in Eq. (2.2.22) generated by them, flanked by the state preparation unitaries, is very reminiscent of the diffusion operators used by Grover [Gro05]; they indeed match exactly for $\beta = -\pi/3$.

As emphasized above, a QAOA employing a Grover mixer will then use the standard phase separation unitary $U_C(\gamma) = e^{-i\gamma C}$ for an objective Hamiltonian C. Of course, the classical part likewise can be taken one-to-one from the Quantum Approximate Optimization Algorithm (cf. Section 2.2.2). Hence, the only components of Algorithm 2 changing in this approach compared to the original one are the initial state $|s\rangle$ and therefore the state preparation unitary U_S as well as the mixer B and thus the mixing unitaries U_B. Due to only having adapted the respective definitions here compared to Section 2.2.2 (but not the used symbols), Algorithm 2 can also be used to depict the Grover mixer approach QAOA, as none of $|s\rangle, U_S, B, U_B$ are specifically referred to a concrete definition in Algorithm 2 - just like the name "QAOA", there abbreviating "Quantum Approximate Optimization Algorithm" and here standing for "(Grover-mixer) Quantum Alternating Operator Ansatz".

One characteristic of a QAOA based on Grover mixers, that was considered a unique selling point by Bärtschi and Eidenbenz [BE20], shall be stated and proven explicitly, namely that feasible basis states end up with the same amplitude if the objective function values of the corresponding bit strings are equal.

Proposition 2.6. *Suppose the quantum part of a depth-p QAOA for an optimization problem COP of size N to be equipped with a state preparation unitary U_S as in Eq.* (2.2.19)*, phase separation unitaries given by $U_C(\gamma) = e^{-i\gamma C}$ and Grover mixing unitaries $U_B(\beta) = e^{-i\beta|F\rangle\langle F|}$ as in Eq.* (2.2.20)*. Let $\left(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_p\right)_n$ denote the amplitude of the n^{th} computational basis state, corresponding to the basis state $|n\rangle$ via Eq.* (1.2.5)*, in the final state $|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_p$ obtained after one QAOA iteration (cf. Eq.* (2.2.10)*). Then, the following holds for those final amplitudes:*

(i) For $|n\rangle \notin F$ with $n \in \{0,1\}^N$, $\left(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_p\right)_n = 0$.

(ii) For $|n\rangle, |m\rangle \in F$ with $n, m \in \{0,1\}^N$, $\left(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_p\right)_n = \left(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_p\right)_m$ whenever $C(n) = \langle n|C|n\rangle = \langle m|C|m\rangle = C(m)$.

Proof. Inspired by the proof in [BE20], Proposition 2.6 is proven via induction. So, start with the 0^{th} iteration, i.e. the initial state preparation, whose result by construction in Eq. (2.2.18) satisfies

$$(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_0)_n = (|F\rangle)_n = \begin{cases} 1/\sqrt{|\text{feas(COP)}|} & \text{, if} \quad |j\rangle \in F \\ 0 & \text{, if} \quad |n\rangle \notin F, \end{cases}$$

with $n \in \{0,1\}^N$ and thereby naturally properties (i) and (ii). Now assume that this is also true for the k^{th} iteration, $k \in \{1, ...p-1\}$ arbitrary but fixed, i.e. $(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n = 0$ if $|n\rangle \notin F$ and $(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n = (|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_m$ if $|n\rangle, |m\rangle \in F$ and $C(n) = C(m)$ with $n, m \in \{0,1\}^N$. Due to its diagonality in the computational basis, U_C obviously leaves F invariant. and acts as a phase shift. Using the assumption that $(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n = 0$ if $|n\rangle \notin F$, we find that $U_C(\gamma_{k+1})$ acts as

$$\begin{aligned} U_C(\gamma_{k+1})\,|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k &= \sum_{n\in\text{feas(COP)}} U_C(\gamma_{k+1})\,(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n\,|n\rangle \\ &= \sum_{n\in\text{feas(COP)}} e^{-i\gamma_{k+1}C}\,(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n\,|n\rangle \\ &= \sum_{n\in\text{feas(COP)}} e^{-i\gamma_{k+1}C(n)}\,(|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle_k)_n\,|n\rangle\,. \end{aligned}$$

Using that, consider now the arithmetic mean of the amplitudes of that state obtained

after applying $U_C(\gamma_{k+1})$:

$$\begin{aligned}
\mathrm{AM} := \mathrm{AM}\left(U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right) &:= \frac{1}{|\mathrm{feas(COP)}|}\sum_{n\in\mathrm{feas(COP)}} e^{-i\gamma_{k+1}C(n)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n \\
&= \frac{1}{|\mathrm{feas(COP)}|}\sum_{n\in\mathrm{feas(COP)}} e^{-i\gamma_{k+1}C(n)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n \underbrace{\langle n|n\rangle}_{=1} \\
&= \frac{1}{|\mathrm{feas(COP)}|}\sum_{n\in\mathrm{feas(COP)}} \langle n|\, e^{-i\gamma_{k+1}C(n)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n |n\rangle \\
&= \frac{1}{|\mathrm{feas(COP)}|}\sqrt{|\mathrm{feas(COP)}|}\,\langle F|\, U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k \\
&= \frac{1}{\sqrt{|\mathrm{feas(COP)}|}}\langle F|\, U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k
\end{aligned}$$

Leveraging what we found during the derivation of Eq. (2.2.22), this arithmetic mean turns out to be useful when proceeding with the consecutive application of $U_B(\beta_{k+1})$, then giving the state after iteration $k+1$:

$$\begin{aligned}
\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1} &= U_B(\beta_{k+1})U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k \\
&= \left(\mathbb{1} - \left(1-e^{-i\beta_{k+1}}\right)|F\rangle\langle F|\right)U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k \\
&= U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k - \left(1-e^{-i\beta_{k+1}}\right)|F\rangle\langle F|\, U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k \\
&= U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k - \sqrt{|\mathrm{feas(COP)}|}\left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM}\,|F\rangle\,.
\end{aligned}$$

Since $\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1}$ can apparently be expressed as a superposition of $\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k$ and $|F\rangle$, the first conclusion we draw is that $\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1}\right)_n = 0$ if $|n\rangle \notin F$, using the assumption that the same holds for $\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k$. Furthermore, together with the assumption that $\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k$ also obeys property (ii), this result also implies

$$\begin{aligned}
\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1}\right)_n &= \left(U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n - \sqrt{|\mathrm{feas(COP)}|}\left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM}\cdot(|F\rangle)_n \\
&= e^{-i\gamma_{k+1}C(n)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n - \sqrt{|\mathrm{feas(COP)}|}\left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM}\frac{1}{\sqrt{|\mathrm{feas(COP)}|}} \\
&= e^{-i\gamma_{k+1}C(m)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_n - \left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM} \\
&= e^{-i\gamma_{k+1}C(m)}\left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_m - \left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM} \\
&= \left(U_C(\gamma_{k+1})\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_k\right)_m - \sqrt{|\mathrm{feas(COP)}|}\left(1-e^{-i\beta_{k+1}}\right)\mathrm{AM}\cdot(|F\rangle)_m \\
&= \left(\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1}\right)_m
\end{aligned}$$

for $|n\rangle, |m\rangle \in F$ if $C(n) = C(m)$. Hence, $\left|\boldsymbol{\beta},\boldsymbol{\gamma}\right\rangle_{k+1}$ satisfies both (i) and (ii). □

I still owe you to reason that the Grover mixer B in Eq. (2.2.20) is a permitted choice, i.e. that it does not violate any of the prerequisites we have recapped above. Like any

projector, due to

$$B^2 = (|F\rangle\langle F|)(|F\rangle\langle F|) = |F\rangle \underbrace{\langle F|F\rangle}_{=1} \langle F| = |F\rangle\langle F| = B,$$

B has eigenvalues 0 and 1 as

$$\lambda|\varphi\rangle = B|\varphi\rangle = B^2|\varphi\rangle = B(B|\varphi\rangle) = \lambda B|\varphi\rangle = \lambda^2|\varphi\rangle \quad \Rightarrow \quad \lambda^2 \overset{!}{=} \lambda$$

for an eigenstate $|\varphi\rangle$ of B with corresponding eigenvalue λ can only be satisfied by $\lambda = 0$ or $\lambda = 1$. Since

$$B|F\rangle = (|F\rangle\langle F|)|F\rangle = |F\rangle \underbrace{\langle F|F\rangle}_{=1} = |F\rangle,$$

the initial state $|s\rangle = |F\rangle$ is indeed an eigenstate of B corresponding to the largest eigenvalue 1. Unlike the standard mixer in Eq. (2.2.4), the Grover mixer does generally not have a spectrum that is symmetric around 0 (cf. Section 2.2.2). Nevertheless, the projection property implies that we are able to restrict the domain of the parameters $\beta \in [0, 2\pi)$ thanks to the integer eigenvalues. Let us now check the both criteria established by Hadfield [Had18] concerning the mixing unitaries induced by B. To this end, suppose $|z\rangle \in F$; then, using the calculation leading to Eq. (2.2.22),

$$\begin{aligned} U_B(\beta)|z\rangle = e^{-i\beta|F\rangle\langle F|}|z\rangle &= \left(\mathbb{1} - \left(1 - e^{-i\beta}\right)|F\rangle\langle F|\right)|z\rangle \\ &= |z\rangle - \left(1 - e^{-i\beta}\right)|F\rangle \underbrace{\langle F|z\rangle}_{\neq 0} = |z\rangle - \text{const}\,|F\rangle, \end{aligned}$$

i.e. $U_B(\beta)|z\rangle \in F$, meaning that U_B maps feasible states to feasible states independent on the value of β. On the other hand, let $|z_1\rangle, |z_2\rangle \in F$; then, recycling what we just saw directly gives

$$\langle z_2|U_B(\beta)|z_1\rangle = \langle z_2|\left(|z_1\rangle - \text{const}\,|F\rangle\right) = \langle z_2|z_1\rangle - \text{const}\,\underbrace{\langle z_2|F\rangle}_{\neq 0} \neq 0,$$

i.e. we do not even need a repeated application of U_B or any specific value β^* such that there is some non-vanishing transition probability between $|z_1\rangle$ and $|z_2\rangle$ under U_B.[16]

Now where we have managed to trace anything back to the initial state preparation, we must not forget that it is not a priori clear that a unitary U_S as in Eq. (2.2.18), which is not requiring an exponential number of qubits or gates itself, exists at all

[16] An exception of course is $\beta = 0$ where $U_B(0) = \mathbb{1}$, meaning that const $= 0$. However, the critierion by Hadfield [Had18] asks for the existence of at least one such value of β, which is not corrupted by this special case.

for every combinatorial optimization problem. Using Grover mixers should rather be understood as an approach that might be beneficial, depending on the existence and, in particular, the implementability of such a polynomial-size state preparation.

Compared to the general Quantum Alternating Operator Ansatz by Hadfield [Had18], one difference not to be underestimated is the property of the Grover mixing unitaries in Eq. (2.2.22) to generate a superposition of all feasible states in every iteration, which does not necessarily need to be true for general mixing unitaries as used in [Had18]. This can immediately be inferred from $U_B(\beta)\,|z\rangle = |z\rangle - \text{const}\,|F\rangle$ for $|z\rangle \in F$. In the general setting, it might take the QAOA multiple repeated applications of phase separation and mixing unitaries in order to reach such a "perfect mixing", requiring a larger depth of the entire QAOA to achieve optimal solution approximations of comparable quality, which in turn makes it less NISQ-friendly. The Grover mixers conversely are especially designed to have that feature.

Part II.

Algorithm Construction

Core Idea

This part can be considered central as it develops the core idea behind our algorithm - without a theoretical construct there is no implementation and no simulation. Here we are going to put together the different concepts introduced and discussed in Part I. Presenting the underlying construct that determines how these components are joined to collectively form a hybrid quantum-classical Branch-and-Bound algorithm - which we will shortcut denote as *HQCBB* - is the point of this chapter.

In the introduction I emphasized that the classical algorithm provides the framework of the HQCBB; the quantum part - given by the QAOA - is then integrated to optimize the Branch and Bound via improving its overall performance. Therefore, the basic structure of the algorithm will naturally highly resemble Algorithm 1. What was not done in Section 2.1 is to discuss specific choices for the three building blocks of a general B&B, namely the searching strategy, the pruning rules and the branching strategy. Of course, explicit configurations that are tailored specifically to the problem at hand cannot be part of an overarching concept. However there are some high-level designs among the pruning rules that can be described here.

Going through Algorithm 1 in order, let us first say a word about the initialization of the algorithm. The HQCBB implementation will start with an "empty" incumbent where no variables are assigned. In general, updating the incumbent (cf. Lines 9 and 10 in Algorithm 1) only becomes relevant when the searching procedure reaches (feasible) leafs of the tree, as these correspond to full solutions with all variables being assigned, for which the final objective function value can just be evaluated. Of course, if no full solution has been encountered so far, meaning that the incumbent is still in its initial empty state, it is updated in any case, turning it into a feasible solution once and for all. Otherwise, an update is only performed if the current leaf comes along with a better objective function value than the incumbent has at that point.

The first thing that can be considered part of a general pruning strategy is a feasibility check. This verification of feasibility is not only the first toll stop where recently selected subproblems may be rejected, but represents also the first action executed by the

HQCBB after choosing the next node for exploration - the corresponding search space region is not worth further investigation if it does not contain any feasible solutions at all and should therefore be immediately discarded. Verifying the feasibility of a node is straight-forward: We just need to check whether the weights of the selected items do exceed the capacity when being aggregated.

Now proceeding with the pruning rules, recall that I outlined at the end of Section 2.1.2 why we will rely all of our pruning decisions on bounds of the respective objective function: further refinement is overambitious as the standard Branch and Bound is fast enough on problems of those sizes that can be tackled with our quantum algorithms.[1] To become concrete, there will be two kinds of bounds[2], namely an *upper bound* and a *lower bound* (meant as functions here). Although their features are slightly different when tackling maximization or minimization problems, pruning nodes off the search tree is in either case based on comparing upper and lower bounds. For KP as a maximization problem, we calculate an upper bound whenever selecting a subproblem for exploration. The outcome is valid for the restricted region of the search space which is determined by the variable assignment going along with this last node selection and all other nodes in its path to the root (cf. Section 2.1.2). Thus, computing an upper bound for a subproblem starts off from a partial assignment of variables. Crucial now is that the currently explored subproblem may then be discarded if the obtained upper bound is smaller than the best lower bound found so far during the exploration of the search tree. The best lower bound is initialized according to the initial incumbent: it is just set to the lower bound obtained when no variables are assigned, which trivially evaluates to 0. This heuristic is pretty intuitive - it is not possible to find an optimal solution in a search space region about which we know from the outset that the maximum objective-function value we can achieve is still worse than best lower bound found for a different region. This is, even the worst solution contained in this other region is still better than the best in the one currently explored. Of course, instead of the current best lower bound, the best value obtained for a full solution may be used for this comparison if the searching procedure has already encountered a leaf whose final value has not been exceeded by some other region yet. In fact, it is even sufficient for rejection if the current upper bound is not strictly greater than this comparison value, meaning that a node is also pruned in the case of equality, as the currently explored region is apparently not capable of leading to any improvement. In case the node at hand cannot be discarded, we also evaluate its associated lower bound and potentially update the best lower bound, obviously depending on whether the just computed value would increase the best lower bound. This behavior of upper and lower bounds works, as you may expect, vice versa when considering a minimization problem instead.

[1] Still to be verified in Part III.

[2] Whenever we simply speak about "bounds" from now on, we mean bounds on the objective function.

Note that for a maximization problem it is generally easier to obtain lower bounds than upper bounds. This is attested by the fact that any feasible solution automatically provides a lower bound for the entire problem. Upper bounds, on the other hand, are often based on relaxations, since increasing the set of possible feasible solutions can only yield an optimal solution of better or equal quality. The same again holds the other way round for minimization problems. So, that is already set concerning the pruning rules.

Now comes the interesting part: integrating the QAOA. As explained in Section 2.2.2, the output of a QAOA for a maximization problem as derived in Section 2.2 is always providing a lower bound on the optimal solution value; again the analogue is true vice versa for a minimization problem.[3] Therefore, a QAOA may be used as an alternative lower bound for the Knapsack Problem. The idea then is to not only compute the classical lower bound for the currently explored node but also run the constructed QAOA to obtain a quantum lower bound - thereupon the better of the two obtained values can be used as lower bound in the B&B. The crucial rationale here is the following: As we are adding information to the algorithm by computing a second bound, the result cannot get worse. Thereby, non-optimal regions of the search space should in principle be detected and rejected earlier by exploring fewer unhelpful subproblems whose associated nodes can be pruned off; this in turn leads to the legitimate hope that the additional integration of a quantum subroutine given by the QAOA might improve the performance of the overall Branch-and-Bound algorithm. That conviction can really be considered the heart of the HQCBB.

As emphasized in the introduction, the QAOA we are about to construct for the Knapsack Problem based on a novel Grover mixer implicitly represents a second part - besides the HQCBB's intrinsic motivation described above - that is of interest for the quantum computing community in this thesis: the construction of a new feasibility-preserving QAOA for the Knapsack Problem that is based on a Grover mixer which is in turn induced by a recently proposed revolutionary state preparation.

[3] By adjusting signs of the objective Hamiltonian and the final outcome the QAOA logic can simply be adapted in order to properly address minimization problems.

Classical Part

As we will see throughout this chapter, the classical tools to be employed in the HQCBB for the Knapsack Problem for computing upper and lower bounds, generating subproblems according to the assignment of variables as well as in terms of the branching and searching strategy can be considered standard and are not very involved.

4.1. Branching Strategy

Here we are likewise starting off with the easiest part that can be fully specified in one sentence: For the Knapsack Problem, we will employ binary branching, leading to a separation of the search space into two MECE[1] regions in every step. More precisely, the partitioning procedure (the analogue of Line 5 in Algorithm 1) is given by creating two subproblems that are induced by assigning the next variable to have value 0 in one branch and value 1 in the other. Hence, given the list of unexplored subproblems T and a bitstring x corresponding to the node to branch from, this routine updates the *stack* as:

Algorithm 3: `Branching`(T, x)

1 $T \leftarrow T \cup \{x + \text{"0"}\}$
2 $T \leftarrow T \cup \{x + \text{"1"}\}$
3 **return** T

Recalling Definition 1.14, KP is formulated using only one type of binary variables, i.e. a variable set to 1 in the context directly means that the corresponding item is chosen to be included in the knapsack while a value of 0 analogously expresses that it is not.

[1] Mutually exclusive, collectively exhaustive.

4.2. Bounds

As the searching strategy will rely on the used bounds, the latter should be elaborated on priorly. Lower and upper bounds for KP are in fact based on the same simple heuristic called *Greedy*.

4.2.1. Greedy Lower Bound

The *Greedy lower bound* exploits a fact that I have emphasized earlier in Chapter 3, namely that any feasible solution immediately provides a lower bound to the problem, since the optimal solution can obviously not have a worse value by definition of the term "optimality" (cf. Definition 1.10). As lower bounds for a maximization problem like KP are stronger the larger they are, the strategy now is to find a feasible solution that is as good as possible. The following trick will be mainly responsible for why this objective is not as hard as it might seem: Given a list of N items forming a KP instance, each consisting of two integer values (cf. Definition 1.14), our first action will always be to sort them in descending order according to their share of profit by weight such that

$$\frac{p_{\pi(1)}}{w_{\pi(1)}} \geq \frac{p_{\pi(2)}}{w_{\pi(2)}} \geq ... \geq \frac{p_{\pi(N)}}{w_{\pi(N)}}.$$

where π is a permutation of $\mathbf{N} = \{1, ..., N\}$. In case that the given list of N items is not sorted accordingly, this can be done efficiently with $\mathcal{O}(N \log N)$ complexity (cf. Section 1.1), see e.g. [AHU83, Ch.8]. We will therefore assume from now on that the items are already sorted in that fashion, i.e. $p_i/w_i \geq p_j/w_j$ whenever $i < j$ for $i, j \in \mathbf{N}$. This heuristic implies that the HQCBB processes the items and decides on whether or not to include them in the knapsack according to how promising they are, for which this relative profit is a clear indicator. The Greedy heuristic then runs through the list of accordingly sorted items and selects items to be included as long as the capacity is not exceeded, stopping at the first item for which the respective residual capacity is not enough. This procedure indubitably generates a feasible solution and thereby a lower bound. However, it can still be slightly improved in the following way: Instead of breaking at the so-called *critical item* for which the capacity would be exceeded, just skip it and proceed with the next item and check whether that may still be fitting; this is repeated until the end of the list unless the entire capacity has been consumed, see Algorithm 4.

What remains to be clarified is how the Greedy lower bound is computed for subproblems in the tree for which a certain amount of variables have already been set. One of several equivalent options is to generate a subproblem for the remaining variables

Algorithm 4: `GreedyLowerBound`(KP)

```
if items not sorted in descending order of relative profit then
    Sort items such that p_j/w_j ≥ p_{j+1}/w_{j+1}  ∀ j ∈ {1, ..., N − 1}
Set P = 0, W = 0
for j ∈ {1, ..., N} do
    if W + w_j ≤ W_max then
        W ← W + w_j
        P ← P + p_j
        if W = W_max then
            break
return P
```

that are still open and just calculate the lower bound for that reduced problem via Algorithm 4. The result must then be added to some offset that is determined by the already specified variables. However, it may not always be so simple as in the KP case to create an equivalent subproblem from a partial solution; here, we just have to remove the specified items from the list of all items without changing the order of the remaining ones to stick to the relative-profit descending sorting and lastly reduce the capacity by the sum of weights whose corresponding variables are assigned value 1 (each item chosen to be included in the knapsack decreases the remaining capacity by exactly its weight). The offset, on the other hand, is in case of the Knapsack Problem obviously given by the sum of profits corresponding to 1-valued variables.

4.2.2. Greedy Upper Bound

Let us now turn to the upper bound we are going to use from the classical side. In accordance with what I emphasized in Chapter 3, the *Greedy upper bound* is based on the relaxation of the integer constraints featuring the binary variables. This is, demanding $x_j \in \{0,1\} \;\; \forall\, j \in \{1,...,N\}$ in Definition 1.14 for a KP instance of size N is replaced by the continuous domains $x_j \in [0,1]$. In the context, this essentially means that we are also allowed to include only fractions of items into the knapsack. Apart from this allegedly small change in Eq. (1.3.3), the rest of Definition 1.14 stays the same - the result is called the *continuous Knapsack Problem (cKP)*. It is clear that the optimal solution to the continuous Knapsack Problem provides an upper bound on the optimal solution of the standard Knapsack Problem, as an optimal solution $x^* \in \{0,1\}^N$ of the latter is also contained in $[0,1]^N$, although it is probably not optimal in the continuous case. The key gaining of this seemingly unimpressive

relaxation is that the problem is now not longer exponentially hard to solve. To show this we will see that the optimal solution to cKP can be stated right away.

Proposition 4.1. *Let c denote the index of the critical item of a KP instance in the sense of Definition 1.14 whose items are sorted descending according to their relative profits, i.e.*

$$c = \min \left\{ n \in \{1, ..., N\} \, : \, \sum_{j=1}^{n} w_j > W_{max} \right\}.$$

In fact, $1 < c \leq N$ due to assumptions (ii) and (iii) in Section 1.3. Then, the optimal solution $\overline{x}^*$ of cKP is given by

$$\overline{x}_j^* = \begin{cases} 1 & , \text{ if } \quad 1 \leq j \leq c-1 \\ \Delta W_{max}/w_c & , \textit{ if } \quad j = c \\ 0 & , \text{ if } \quad c+1 \leq j \leq N \end{cases}$$

where $\Delta W_{max} = W_{max} - \sum_{j=1}^{c-1} w_j$ denotes the remaining capacity that is not large enough to also include the critical item into the knapsack.

Proof. Inspired by Martello and Toth [MT90, Ss.2.2.1], first note than any optimal solution of cKP needs to exactly consume the capacity, as the whatsoever big remaining part may otherwise be used to additionally include a fraction of any item that is not already selected to be fully included in the knapsack. Now assume $p_j/w_j > p_{j+1}/w_{j+1} \;\; \forall \, j \in \{1, ..., N-1\}$ w.l.o.g.[2] and denote the optimal solution of cKP by $\overline{y}^*$ for which we suppose that $\overline{y}_k^* < 1$ for a $k < c$. In order to still fully consume the entire capacity, there must then at least be one $q \geq c$ such that $\overline{y}_q^* > \overline{x}_q^*$ with $\overline{x}^*$ as defined in Proposition 4.1. Hence, when choosing $\varepsilon > 0$ small enough, we can increase the value of $\overline{y}_k^*$ by ε and diminish the value of $\overline{y}_q^*$ by $\varepsilon \cdot w_k/w_q$ while upholding $\overline{y}_k^* < 1$ and $\overline{y}_q^* < \overline{x}_q^*$, respectively, leading to a difference in the total profit of

$$\varepsilon p_k - \varepsilon \frac{w_k}{w_q} p_q = \varepsilon w_k \underbrace{\left(\frac{p_k}{w_k} - \frac{p_q}{w_q} \right)}_{>0} > 0 \quad \text{as } k < q.$$

This is obviously a contradiction to the assumption that $\overline{y}^*$ is the optimal solution, corresponding to the maximal achievable profit. The same logic allows to show that $\overline{y}_k^* > 0$ for $k > c$ leads to a analogous contradiction. As the capacity needs to be

[2] Note that we above demanded $p_1/w_1 \geq p_2/w_2 \geq ... \geq p_N/w_N$ instead of strict inequalities. However, we can always find a $\tilde{p}$ shifting all profits $p_j \mapsto \tilde{p}_j = p_j + \tilde{p}$ such that $\tilde{p}_j/w_j > \tilde{p}_{j+1}/w_{j+1} \;\; \forall \, j \in \{1, ..., N-1\}$ is satisfied. The same could obviously also be done with the weights.

completely exhausted for optimality, this finally implies that the optimal solution to cKP must be given by $\overline{y}^* = \overline{x}^*$, whose total weight (cf. Eq. (1.3.2)) evaluates to

$$W(\overline{x}^*) = \sum_{j=1}^{N} w_j \overline{x}_j^* = \sum_{j=1}^{c-1} w_j + w_c \frac{\Delta W_{\max}}{w_c} = \sum_{j=1}^{c-1} w_j + \Delta W_{\max} = W_{\max}.$$

□

Proposition 4.1 shifts the effort of finding an optimal solution of cKP to finding the critical item. Martello and Toth [MT90, Ss.2.2.2] show how this can be done in $\mathcal{O}(N)$ time for a cKP instance of size N, which in turn means that the continuous Knapsack Problem is not NP-hard. The optimal solution $\overline{x}^*$ as in Proposition 4.1 has a total profit (cf. Eq. (1.3.1)) of

$$P(\overline{x}^*) = \sum_{j=1}^{N} p_j \overline{x}_j^* = \sum_{j=1}^{c-1} p_j + p_c \frac{\Delta W_{\max}}{w_c} = \sum_{j=1}^{c-1} p_j + \frac{p_c}{w_c} \Delta W_{\max}.$$

Due to the integer constraints and the integer-valued profits featuring the (standard) Knapsack Problem according to Definition 1.14, a valid upper bound on its optimal solution value is therefore given by

$$\lfloor P(\overline{x}^*) \rfloor = \sum_{j=1}^{c-1} p_j + \left\lfloor \frac{p_c}{w_c} \Delta W_{\max} \right\rfloor.$$

The Greedy upper bound computes this value iteratively as illustrated by Algorithm 5.

Algorithm 5: `GreedyUpperBound`(KP)

```
1  if items not sorted in descending order of relative profit then
2      Sort items such that p_j/w_j ≥ p_{j+1}/w_{j+1}  ∀ j ∈ {1, ..., N − 1}
3  Set P = 0, W = 0
4  for j ∈ {1, ..., N} do
5      if W + w_j > W_max then
6          Set ΔW_max = W_max − W
7          P ← P + (p_c/w_c) ΔW_max
8          break
9      W ← W + w_j
10     P ← P + p_j
11 return ⌊P⌋
```

Algorithms 4 and 5 show that both the Greedy lower bound as well as the Greedy upper bound for KP with N items can be computed within a complexity of $\mathcal{O}(N)$. This is, the further down we get in the tree during the searching procedure, i.e. the more variables are specified or items decided on, the faster these bounds can be evaluated.

4.3. Searching Strategy

Now we can turn to the searching strategy we will employ in the Knapsack HQCBB. At the risk of repeating myself, we will again keep it simple to not waste our energy too much at the classical stuff. Accordingly, the searching procedure will follow a primitive heuristic that most closely resembles the depth-first search (cf. Section 2.1.4). However, it is engineered in one respect in order to shift the trade-off between its simplicity on the one hand and its inability to react on the given problem structure to a fair balance. To be concrete, this is done by taking into account the chosen classical lower bound just discussed in Section 4.2, making it a mixture of DFS and BFS where the lower bound takes the role of the measure-of-best function.[3]

Moreover, our searching heuristic is going to be composed of two parts: one function corresponding to the standard node selection after a branching has occurred and one method that backtracks and decides on which node to proceed with if the current search space region is not to be investigated further. This case arises if we find the currently selected subproblem to be infeasible or that it can be ordinarily pruned via a bound comparison or when we have reached a leaf of the tree. More specifically, our two-component heuristic is given as follows: After a branching step, always pick one of the two generated children next for exploration; even more precisely, choose the one with the better lower bound (i.e. that one whose Greedy lower bound is larger) with ties being broken randomly. Two exceptions to this rule exist, namely if either not both of the partial solutions resulting from the branching are feasible or if the corresponding nodes generated represent leafs. In the former case the infeasible one might probably win the comparison while it is meaningless to compute bounds for leafs in the second, as their associated solution value can be directly evaluated. Both exceptions are handled via randomly selecting one of the two nodes. This might seem strange at the first sight, as an infeasible node may thereby have the same chance of being chosen compared to a feasible one; however, this edge-case can safely be ignored as the next algorithm iteration first checks the picked node for feasibility and potentially throws it away immediately. Given the list of unexplored subproblems T, this behavior is summarized by Algorithm 6.

Due to always choosing one of the direct child nodes after a branching - meaning that increasing the depth in the tree is preferred over moving back to a potentially more promising region - our strategy clearly inherits from the DFS. But however, among the two options, the more auspicious path is been taken based on the lower bound, bringing some BFS aspects into play. What is still open is the question how our algorithm

[3] As we learned in Section 2.1.4, there exist some contour drawing rules such that this custom searching heuristic can be understood as a CBFS strategy.

Algorithm 6: `NodeSelection`(T)

Set $x^{(0)}, x^{(1)} = T[-2], T[-1]$ (initialization of options as lastly appended nodes)
if $\left|x^{(0)}\right| = \left|x^{(1)}\right| = N$ *or* $z^{(0)}$ *is not feasible or* $x^{(1)}$ *is not feasible* **then**
    return random $\left\{x^{(0)}, x^{(1)}\right\}$
else
    return $\underset{i \in \{0,1\}}{\operatorname{argmax}}$ `GreedyLowerBound`$\left(x^{(i)}\right)$

backtracks, i.e. which node is selected after an exploration path has come to an end. The method implementing this is even simpler than the node selection - it just picks the node at the last position in the list of unexplored subproblems T:

Algorithm 7: `Backtracking`(T)

$x = T[-1]$
return x

That is, it always selects that open node which has the least distance from the current node. Due to our branching strategy being configured to always be binary, there are only two options for the next node selected for exploration by the backtracking function: either it is the second of the two child nodes generated during the last branching step that was not chosen first by the node selection procedure or it is the node with the largest depth among the ones that survived the branching and subsequent node selection up to this point, as its distance from the current node is minimal. Hence, comparing with what was said in Section 2.1.4, our backtracking logic was adopted one-to-one from the DFS strategy. Of course, this kind of backtracking only works if there are still unexplored subproblems in the tree.

4.4. Further Refinement and Full HQCBB

Apart from the three main components of a Branch-and-Bound algorithm that we worked through in Sections 4.1 to 4.3, there are some smaller refinements that shall be additionally incorporated due to their good effectiveness vs. implementation complexity ratios.

The first is the feasibility check emphasized in Chapter 3, which is performed right away after deciding on the next node to explore via the standard node selection or

the backtracking heuristic described in Section 4.3. For the Knapsack Problem (cf. Definition 1.14), a partial solution $x = x_1 \cdots x_n$ with $n \in \mathbf{N} = \{1, ..., N\}$ is feasible according to Definition 1.9 if $W(x) = \sum_{j=1}^{n} w_j x_j \leq W_{\max}$.

When talking about bounds (cf. Section 4.2) we were already talking about how to generate a subproblem with the usual shape of a KP instance (cf. Definition 1.14) based on a partial assignment of variables $x = x_1 \cdots x_n$ with $n \in \mathbf{N}$. However, the simplistic rule described there may be extended in one respect: In order to guarantee that the generated subproblem meets the assumptions made in Section 1.3, the weight of every remaining item should be checked against the residual capacity to make sure that any item is in principle affordable. Extracting all non-specified items and performing this check in every step yields a method that shall be denoted `GenerateSubproblem`(x).

Another trick that exploits the KP problem structure and aims at improving the HQCBB performance from the classical point of view is to check for a currently selected non-leaf node[4] - after its feasibility being verified - whether the corresponding partial solution happens to already fully consume the capacity. Why would we want to know that? When there is nothing left from the capacity, we can directly infer that there is only one feasible complete solution remaining in the search space region induced by the currently explored subproblem, which is obtained by assigning value 0 to all other yet unspecified variables. Any other solution having the current node as a parent in the tree can with certainty be assessed infeasible for including at least one more item into the knapsack, although the capacity was already exhausted entirely. By creating that single feasible solution via appending only 0's to the current partial assignment of variables, keeping its associated objective function value and by potentially updating the incumbent accordingly, we might save computing time which would otherwise be spent in regions where there is not much to be gained.

Now we have classically everything together for our HQCBB. In accordance with the core idea explained in Chapter 3, combining the branching strategy from Section 4.1, the lower and upper bounds from Section 4.2 making up the pruning rules as well as the searching strategy from Section 4.3 yields the fundamental framework for our algorithm. The general QAOA pattern given by Algorithm 2 was visited in Section 2.2.2. Although configuring it specifically for the Knapsack Problem is still outstanding, it can already be high-level referenced here. Embedding it into the classical Branch and Bound as described in Chapter 3 finally leads to the following algorithm structure, depicted in Algorithm 8 with pseudocode.

[4]Leafs are treated separately as described in Chapter 3; for them, the following is meaningless.

Algorithm 8: `Knapsack-HQCBB`(KP)

Set $T = \{\}$ (initialization of the stack as empty list)
Set $\hat{x} =$ "" (initialization of the incumbent as empty bitstring)
Set $\widehat{lb} =$ `GreedyLowerBound`(KP) (initialization of the best lower bound)
$T \leftarrow$ `Branching`$(T, \hat{x})$
Set $x =$ `NodeSelection`(T) (initiallization of the currently explored node)
while $T \neq \emptyset$ **do**
  if *x is not feasible, i.e. $W(x) > W_{max}$* **then**
    $T \leftarrow T \setminus \{x\}$
    if $T \neq \emptyset$ **then**
      $x \leftarrow$ `Backtracking`(T)
    continue
  if *x is a leaf, $|x| = N$, or the capacity is exhausted, $\Delta W_{max}(x) = 0$,* **then**
    if *x is not a leaf, i.e. $|\hat{x}| \neq N$* **then**
      Define $\hat{x}_{\text{candidate}} = x +$ "0"$^{N-|z|}$ (completion of partial solution)
    else
      Define $\hat{x}_{\text{candidate}} = x$
    if *$\hat{x}$ is empty, i.e. $|\hat{z}| = 0$* **then**
      $\hat{x} \leftarrow \hat{x}_{\text{candidate}}$
      Set $\hat{P} = P(\hat{x}_{\text{candidate}})$ (initialization of incumbent profit)
    else
      if $P(\hat{x}_{candidate}) > \hat{P}$ **then**
        $\hat{x} \leftarrow \hat{x}_{\text{candidate}}$
        $\hat{P} \leftarrow P(\hat{x}_{\text{candidate}})$
    $T \leftarrow T \setminus \{x\}$
    if $T \neq \emptyset$ **then**
      $x \leftarrow$ `Backtracking`(T)
    continue
  KP$' =$ `GenerateSubproblem`(z) (translation of partial choice into subproblem)
  $ub = P(z) +$ `GreedyUpperBound`(KP$'$)
  if `canBePruned`$\left(z, ub, \widehat{lb}\right) =$ `true` **then**
    $T \leftarrow T \setminus \{z\}$
    if $T \neq \emptyset$ **then**
      $z \leftarrow$ `Backtracking`(T, z)
    continue

35 $lb = \texttt{ClassicalvsQuantumLowerBound}(\mathrm{KP}', x)$
36 **if** $lb > \widehat{lb}$ **then**
37 $\widehat{lb} \leftarrow lb$
38 $T \leftarrow \texttt{Branching}(T, x)$
39 $T \leftarrow T \setminus \{x\}$
40 $z \leftarrow \texttt{NodeSelection}(T)$

return $\hat{x}$, $\hat{P}$

Algorithm 8 accesses Algorithm 3 (branching), Algorithm 5 (Greedy upper bound), Algorithm 6 (node selection) and Algorithm 7 (backtracking). Although we developed the HQCBB to be part of the general Branch and Bound family, Algorithm 8 somehow differs structurally from Algorithm 1. In the general environment the standard order may feed the intuition of how a B&B should work. In our case however, where deciding on the next subproblem to explore is done at the initialization and is then concluding each iteration, this adaption is a consequence of having the searching routine split up into two distinct methods - the node selection after a branching and the backtracking in case a path in the tree has come to a natural or premature end (cf. Section 4.3). That being said, the Boolean function `canBePruned` - which returns "true" in case that the node at hand may be pruned off the tree - is outsourced to Algorithm 9. It is in line with the pruning strategy introduced in Chapter 3.

Algorithm 9: $\texttt{canBePruned}\big(z, ub, \widehat{lb}\big)$

1 **if** $|z| = 0$ **then**
2 **if** $ub < \widehat{lb}$ **then return** `true` **else return** `false`;
3 **else**
4 **if** $ub \leq \widehat{lb}$ **then return** `true` **else return** `false`;

Lower bounds are determined in Line 35 following the concept described in Chapter 3:

Algorithm 10: $\texttt{ClassicalvsQuantumLowerBound}(\mathrm{KP}, z)$

1 Calculate classical lower bound $lb_{\text{classical}} = \texttt{GreedyLowerBound}(\mathrm{KP})$
2 Calculate quantum lower bound $lb_{\text{quantum}} = \texttt{KP-QAOA}(\mathrm{KP})$
3 **return** $P(z) + \max\{lb_{\text{classical}},\ lb_{\text{quantum}}\}$

It is now about time to assign a meaning to set up a QAOA specifically tailored to the Knapsack Problem, thereby assigning a meaning to `KP-QAOA`.

Quantum Part

As already announced in Chapter 3, the QAOA used in our HQCBB (cf. Algorithm 8 or, more precisely, Algorithm 10) is based on a recent approach for a very different quantum algorithm tackling the Knapsack Problem proposed by Wilkening et al. [Wil+23]. This algorithm consists of two driving parts: a state preparation procedure creating a superposition of all feasible states and a combined version of quantum amplitude amplification [Bra+00] and quantum maximum finding [DH99]. Appreciating the value of the completely new procedure how to create a superposition of all feasible states for a given KP instance, their algorithm can arguably be called *QTG-based search*[1]. Based on what we learned in Section 2.2.3, we will take advantage of the feasible state preparation solely, enabling us to construct a Grover mixer and thereby design a QAOA for the Knapsack Problem with an inherent feasibility preservation that allows to circumvent to softcode the constraint.

5.1. Quantum Tree Generation

The first thing that usually needs to be figured out for the concrete implementation of a quantum algorithm is the amount of necessary qubits. As described in Section 1.2.2, any classical bit is replaced by a qubit when translating a combinatorial optimization problem to the quantum mechanical context. Hence, a register made up by N qubits is first and foremost required to tackle a KP instance in the sense of Definition 1.14, consisting of N elements, on a quantum computer. In accordance with Section 1.2.2, the corresponding Hilbert space describing this *item register* I is denoted by $\mathbb{q}^N$. The state preparation in the QTG-based search approach however also needs an additional auxiliary register of $\lfloor \log W_{\max} \rfloor + 1$ *ancilla qubits* for maintaining feasibility among the generated states which shall later be part of the overall superposition by

[1] What the abbreviation *QTG* stands for will be clarified soon, for the moment it is just a namesake for the algorithm.

keeping track of the residual capacity $w_{\max}$ in every operation step.[2] Analogously, the associated Hilbert space to this *capacity register* W is $\mathbb{q}^{\lfloor \log W_{\max} \rfloor + 1}$. The alternative to introducing ancilla qubits would be to perform mid-circuit measurements; however, any measurement naturally destroys the current superposition at that respective point in the state preparation procedure, which is why the former approach is preferred. Hence, to sum up, any state during the algorithm is represented as $|x\rangle_I |w_{\max}\rangle_W$ for some $x \in \{1, ..., 2^N - 1\}$ using Eq. (1.2.5) and where $w_{\max} \in \{0, ..., W_{\max}\}$.

Inspired by the Bellman recursion in Dynamic Programming (cf. Eq. (A.1) in Appendix A), the objective of the QTG-based search, given a KP instance, is to find N unitaries $\mathcal{G}_n, n \in \{1, ..., N\}$, one for each item, such that

$$\mathcal{G}_n |x\rangle_I |w_{\max}\rangle_W = \begin{cases} |x\rangle_I |w_{\max}\rangle_W & , \text{ if } \quad w_n > w_{\max} \\ \frac{1}{\sqrt{2}} \left(|x\rangle_I |w_{\max}\rangle_W + |x + 2^{N-n}\rangle_I |w_{\max} - w_n\rangle_W \right) & , \text{ otherwise} \end{cases} \tag{5.1.1}$$

$\mathcal{G}_n$ as in Eq. (5.1.1) generates a uniform superposition of the state corresponding to the n^{th} item getting packed and the one where it is not in case that the item is still affordable, otherwise it acts as the identity. The whole state preparation routine then simply processes all items successively, i.e. it is described by $\mathcal{G} = \prod_{n=1}^{N} \mathcal{G}_n$. In Dynamic Programming, the recursive algorithm starts with no item being allowed to be put in the knapsack. Here, the initial state is given by $|0\rangle_I |W_{\max}\rangle_W$ where all qubits in the item register are initialized in state $|0\rangle$ while the capacity register encodes the full capacity as no item is selected.[3] Note that, according to Eq. (1.2.5), the state $|x + 2^{N-n}\rangle_I$ indeed describes the same assignment of variables as $|x\rangle_I$ except for the n^{th} position where the corresponding qubit is instead being flipped to state $|1\rangle$.[4] The underlying idea of $\mathcal{G}$ - with $\mathcal{G}_n$ acting as in Eq. (5.1.1) - is to iteratively construct a binary tree with nodes corresponding to quantum states $|x\rangle_I |w_{\max}\rangle_W$, always representing full classical solutions (complete assignments of variables), but to generate only those nodes that are associated with feasible solutions, making the branching trivial in case the next item is too expensive. After all N items have been processed, the leafs should represent the feasible solutions of the given KP instance. This behavior is why $\mathcal{G}$ is, in accordance with the terminology in [Wil+23], called *quantum tree generator (QTG)*. The example in Fig. 5.1 should make that clearer.

[2] Beware that a register of size $\lfloor \log W_{\max} \rfloor + 1$ is capable of storing integers up to $2^{\lfloor \log W_{\max} \rfloor + 1} \geq 2^{\log W_{\max}} = W_{\max}$ where log denotes the logarithm to base 2.

[3] Note that this does not mean that all qubits in register W shall initially be in state $|1\rangle$, since $2^{\lfloor \log W_{\max} \rfloor + 1} \geq W_{\max}$; the capacity might not be the largest number that can be encoded with these qubits.

[4] Since the item register is initialized as $|0\rangle_I$ and every item is considered exactly once, $|x\rangle_I$ cannot already have $x_n = 1$ before applying $\mathcal{G}_n$

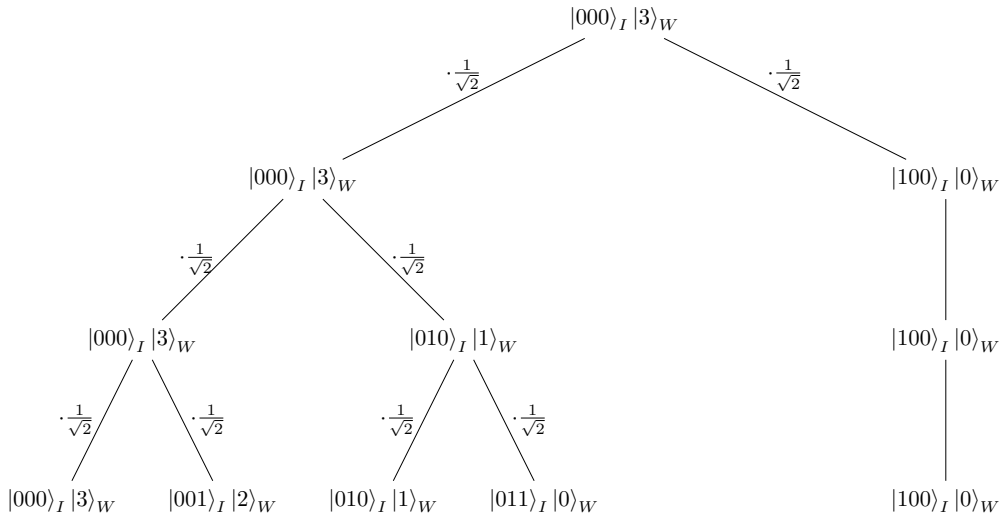


Figure 5.1.: Visualization of the action of the quantum tree generator $\mathcal{G}$ for a simple exemplary KP instance of three items with profits $\mathbf{p} = (4, 2, 1)$, weights $\mathbf{w} = (3, 2, 1)$ and capacity $W_{\max} = 3$. $\mathcal{G}$ is applied to the initial state $|0\rangle_I |W_{\max}\rangle_W = |000\rangle_I |3\rangle_W$. To save the reader from having to make the translation in Eq. (1.2.5) live, the item register qubits are written in binary here, directly depicting which items are included and which are not during the respective steps. Numbers besides branching lines connecting two nodes indicate the extra phase the child node gets via the application of the corresponding $\mathcal{G}_n, n \in \{1, 2, 3\}$, compared to its parent. In this example, we obtain five leafs representing the feasible solutions, namely $|000\rangle_I |3\rangle_W, |001\rangle_I |2\rangle_W, |010\rangle_I |1\rangle_W, |011\rangle_I |0\rangle_W$ and $|100\rangle_I |0\rangle_W$, where the latter appears with an amplitude of $1/\sqrt{2}$ while each of the former four comes with amplitude $1/\sqrt{2}^3$. The optimal solution $|100\rangle_I |0\rangle_W$ happens to end up with the largest amplitude in this example.

Classically, this procedure would need $\mathcal{O}(2^N)$ operations; the creation of superpositions in the quantum case conversely allows to follow different paths simultaneously. As a consequence, the cost of generating the tree via the procedure above can be found to not scale with the maximum number of possible nodes but with the amount of items instead, yielding a computing time of order $\mathcal{O}(N)$. However, the advantage of this exponential speedup in the tree generation generally comes along with exponentially small amplitudes of the single states in the final superposition.

Let us sum up the essence of what we have seen so far as a direct consequence of the explicit construction in Eq. (5.1.1).

Proposition 5.1. *Let W_{max} be the capacity of a given a KP instance in the sense of Definition 1.14. Then, applying the quantum tree generator $\mathcal{G} = \prod_{n=1}^{N} \mathcal{G}_n$ to the initial state $|0\rangle^I |W_{max}\rangle^W$ generates a superposition of all and only feasible solution states,*

denoted by

$$|\mathrm{KP}\rangle = \mathcal{G}\left(|0\rangle_I |W_{max}\rangle_W\right). \tag{5.1.2}$$

How $\mathcal{G}$ can be implemented using known quantum gates will be discussed in detail in Part III, more specifically, Section 6.1. This chapter is only supposed to elaborate the HQCBB algorithm for the Knapsack Problem without touching implementation-related questions such as gate counts, required circuit depths and the like.

5.2. Grover-Mixer QAOA

With the discussion of the quantum tree generation being done, we can finally construct the QAOA that shall be employed in the HQCBB for the Knapsack Problem. As already emphasized in the beginning of Section 5.1, the quantum tree generation procedure strongly suggests to pursue the Grover-mixer approach presented in Section 2.2.3.

To become concrete, $|\mathrm{KP}\rangle$ will play the role of the desired initial state, meaning that our (Grover) mixer will be given by

$$B = |\mathrm{KP}\rangle \langle \mathrm{KP}|. \tag{5.2.1}$$

Thanks to Section 5.1, we already know how to prepare the initial state: via $\mathcal{G}$ as in Eq. (5.1.2). At risk of stating the obvious, $\mathcal{G}$ therefore corresponds here to what U_S was in Section 2.2.3. However, in order to employ the quantum tree generation as state preparation procedure in a Gover mixer QAOA, one discrepancy between $\mathcal{G}$ and a general U_S has to be overcome. More precisely, while $\mathcal{G}$ acts on the initial state $|0\rangle_I |W_{\max}\rangle_W$, Eq. (2.2.19) states that a proper U_S prepares the desired QAOA initial state by operating on $|0\rangle$. Hence, $\mathcal{G}$ needs to be extended by an initial unitary that transforms $|0\rangle_W \mapsto |W_{\max}\rangle_W$ in order to obtain our final state preparation; the result shall be denoted by G. This is,

$$|\mathrm{KP}\rangle = G\left(|0\rangle_I |0\rangle_W\right) \tag{5.2.2}$$

by Proposition 5.1. Although finding the unitary $|0\rangle_W \mapsto |W_{\max}\rangle_W$ is straight forward, its discussion is postponed to the implementation part (cf. Section 6.1). Accordingly, the Grover mixing unitaries are induced by Eqs. (5.2.1) and (5.2.2) to evaluate to

$$U_B(\beta) = e^{-i\beta|\mathrm{KP}\rangle\langle\mathrm{KP}|} = G\left(\mathbb{1} - \left(1 - e^{-i\beta}\right)|0\rangle\langle 0|\right) G^\dagger \tag{5.2.3}$$

where the last equality can be derived analogously to Eq. (2.2.22), based on the unitarity of G, which has to be verified after discussing how to realize the transformation

$|0\rangle_W \mapsto |W_{\max}\rangle_W$, taking $\mathcal{G}$ to G. So far, so good. In accordance with what we said in Section 2.2.3 about the general Grover-mixer ansatz, our QAOA employs the standard phase separation unitaries $U_P(\gamma) = e^{-i\gamma P}$ with objective Hamiltonian P, determined by the KP objective function in Eq. (1.3.1) and defined operator-valued via Eq. (1.2.6). Now everything should be configured in order to establish the capability of setting up Algorithm 2 for the Knapsack Problem.

However, the attentive reader may have noticed that there is one subtlety that should at least briefly be addressed in a consistent thesis as this one aims to be. Compared to the desired initial state $|F\rangle$ given by Eq. (2.2.18) in the general Grover-mixer approach, the result of applying the QTG, $|\mathrm{KP}\rangle$, is not a uniform superposition of the feasible solution states with all of them sharing the same amplitude $1/\sqrt{|\mathrm{feas(KP)}|}$. In contrast, Eq. (5.1.1) implies that the single states making up that superposition may have any of the amplitudes $1/\sqrt{2^n}, n \in \{1, ..., N\}$, meaning that they will in general not be equal.[5] Since any KP instance handled in this thesis has a non-trivial solution space due to assumption (ii) in Section 1.3, some states will generally have larger amplitudes than the desired uniform one, others will naturally end up with lower values.[6] Due to this deviation between $|F\rangle$ and $|\mathrm{KP}\rangle$, we lose the property of the Grover mixing unitaries to mix equal feasible solutions at equal amplitudes (cf. Proposition 2.6). The second property, in contrast, is clearly upheld: Since only the relations between the different amplitudes have changed but not the states in the superposition themselves, we can infer that $U_B(\beta) = e^{-i\beta|\mathrm{KP}\rangle\langle\mathrm{KP}|}$ also maps feasible states to feasible states without the need of any further reasoning, making $B = |\mathrm{KP}\rangle\langle\mathrm{KP}|$ a valid choice for a mixer according to Hadfield [Had18] in the first place. Of course, we could aim at transforming $|\mathrm{KP}\rangle$ into a uniform superposition using the results of applying $\mathcal{G}$ once. However - especially based on the rather moderate consequences seen above when using $|\mathrm{KP}\rangle$ as in Eq. (5.1.2) instead of $|F\rangle$ as in Eq. (2.2.18) - this is considered not worth the computational overhead in terms of both gates and qubits. Generating a uniform superposition using the state preparation may, in general, be desirable for inducing an unbiased mixing effect; however, it seems rather artificial when desperately trying to balance the unequal amplitudes just in order to satisfy the alleged ideal conditions imposed by Bärtschi and Eidenbenz [BE20].

[5] Here we see the potentially exponentially small amplitudes mentioned above.

[6] Note that assumption (iii) on the other hand implies that an amplitude of $1/\sqrt{2}^0 = 1$ is not possible. Otherwise no item would be affordable without immediately exceeding the capacity, making the optimal solution trivial. This is, there is at least one branching in the tree generated by the QTG.

Part III.

Implementation and Simulation

QAOA Implementation

For our QAOA, constructing circuits for the unitaries that collectively realize the quantum tree generation $\mathcal{G}$ and thereby also the preparation of the desired initial state $|\mathrm{KP}\rangle$ is a major open point. Moreover, up to this point it not entirely clear how to decompose the (standard) phase separation unitaries and the (Grover) mixing unitaries into well-known single gates. These things are to be elaborated here.

6.1. State Preparation via QTG

As preparing the initial state, to which the phase separation and mixing unitaries are then applied alternately, is really the first part of any QAOA, the QTG circuit is the natural choice to start the discussion of the QAOA implementation. As described in Section 5.1, the whole QTG routine implementing $\mathcal{G}$ is composed of N structurally equal subcircuits corresponding to the $\mathcal{G}_n, n \in \{1, ..., N\}$. Eq. (5.1.1) then implies that each of these can itself be separated in two parts, one related to the item register I and the other to the capacity register W. As kind of a recap, the former shall put the I-qubits in a superposition of the next item being included and excluded based on the output of evaluating whether it is still affordable; the latter, on the other hand, is responsible for updating the residual capacity encoded across the W-qubits depending on whether we were able include the next item in the knapsack without breaking feasibility. Therefore, both parts mutually affect each other.

6.1.1. Digital Comparator

The first check deciding on including the next item and acting upon the output can be realized by exploiting a classical technique comparing two binary numbers that is called the *digital comparator*. We will start with working out the circuit for a single

$\mathcal{G}_n$ in the sense of Eq. (5.1.1) and connect them in series afterwards to obtain the full QTG circuit. Considering the n^{th} item, we need an additional register R to encode its weight w_n and make it comparable. This register R is, in fact, a classical register, as all the items weights are given as input to the respective KP instance at hand. In theory, $\lfloor \log w_n \rfloor + 1$ classical bits would be sufficient to encode the weight. However, due to assumption (iii) in Section 1.3, register R needs to be enlarged to size $\lfloor \log W_{\max} \rfloor + 1$ in order to be able to compare the binary representations of w_n and $w_{\max}$ for the latter occupying values in $\{1, ..., W_{\max}\}$.

The rationale behind the (classical) digital comparator is to start at the most significant bit and successively compare the bit values of the two numbers whose relation we want to figure out. Suppose we want to compare two, say, integer numbers $a, b \in \mathbb{N}$ given in binary representation $a = a_1 \cdots a_A, b = b_1 \cdots b_B$ with $A = \lfloor \log a \rfloor + 1, B = \lfloor \log b \rfloor + 1$. Now define $\kappa_j := a_j b_j + \overline{a}_j \overline{b}_j$ for $j \in \{1, ..., M\}$ with $M := \max\{A, B\}$. Then, $a = b$ if and only if $\kappa_j = 1 \ \forall j$. Using this and denoting $\overline{a}_j = (a_j + 1) \mod 2$, we can state a logical expression evaluating to 1 in case $a < b$ and to 0 otherwise as follows:

$$(a < b) = \overline{a}_1 b_1 + \kappa_1 \overline{a}_2 b_2 + \kappa_1 \kappa_2 \overline{a}_3 b_3 + \cdots + \kappa_1 x_2 \cdots x_{M-1} \overline{a}_M b_M = \sum_{j=1}^{M} \left(\prod_{i=1}^{j-1} \kappa_i \right) \overline{a}_j b_j. \tag{6.1.1}$$

In case that $a_j = b_j$, the corresponding term in Eq. (6.1.1) is zero and $\kappa_j = 1$; once the iterative procedure found an index $j^* \in \{1, ..., M\}$ for which $a_{j^*} = 0 < 1 = b_{j^*}$, this term evaluates to 1 and furthermore induces every following summand to vanish for always multiplying with $\kappa_{j^*} = 0$ afterwards. As terms for which $a_j = 1 > 0 = b_j$ will also be zero since $\overline{a}_j = 0 = b_j$ in this case, Eq. (6.1.1) cannot exceed a value of 1 and returns this if and only if $a > b$.

Before translating the digital comparator technique to our situation, we need to know what should be done in the different outcome cases. Since the QTG routine starts from state $|0\rangle_I |W_{\max}\rangle_W$, $\mathcal{G}_n$ acts on states of the form $|x_1 \cdots x_{n-1} 00 \cdots 0\rangle_I |w_{\max}\rangle_W$ as emphasized in Section 5.1. According to Eq. (5.1.1), the item register qubits shall be put in superposition as

$$\begin{aligned} |x\rangle_I = \ |x_1 \cdots x_{n-1} 00 \cdots 0\rangle_I &\mapsto \frac{1}{\sqrt{2}} \left(|x_1 \cdots z_{x-1} 00 \cdots 0\rangle_I + |x_1 \cdots x_{n-1} 10 \cdots 0\rangle_I \right) \\ &= \frac{1}{\sqrt{2}} \left(|x\rangle_I + |x + 2^{N-n}\rangle_I \right) \end{aligned}$$

in case that $w_n \leq w_{\max}$. This can trivially be achieved by applying a Hadamard gate on the n^{th} item-register qubit, see Eq. (2.2.6). The fact that we want to actually perform this transformation in case of "$\leq$" instead of "$<$" as in the digital comparator complicates things a little. We can overcome this by either checking for the opposite, i.e. $w_n > w_{\max}$, implying the need to flip all W-qubits and all classical R-bits before and after the comparison, or we could append an additional check for simultaneous equality

in all W and R positions. However, we choose a third option, namely first applying the Hadamard on the n^{th} I-qubit on the assumption that $w_n \leq w_{\max}$ holds, then checking for $w_n > w_{\max}$ via the digital comparator logic and finally revert the first operation in case this check evaluates to "true" by applying H once more, leveraging $H^2 = \mathbb{1}$. This way, we only need one additional single-qubit gate instead of $2\lfloor \log W_{\max} \rfloor + 1$ single-qubit gates or one extra $(\lfloor \log W_{\max} \rfloor + 2)$-qubit gate.

Based on that strategy we can now incorporate the logic of Eq. (6.1.1) to construct the quantum circuit for the I-register part of $\mathcal{G}_n$; the result is shown in Fig. 6.1.[1]

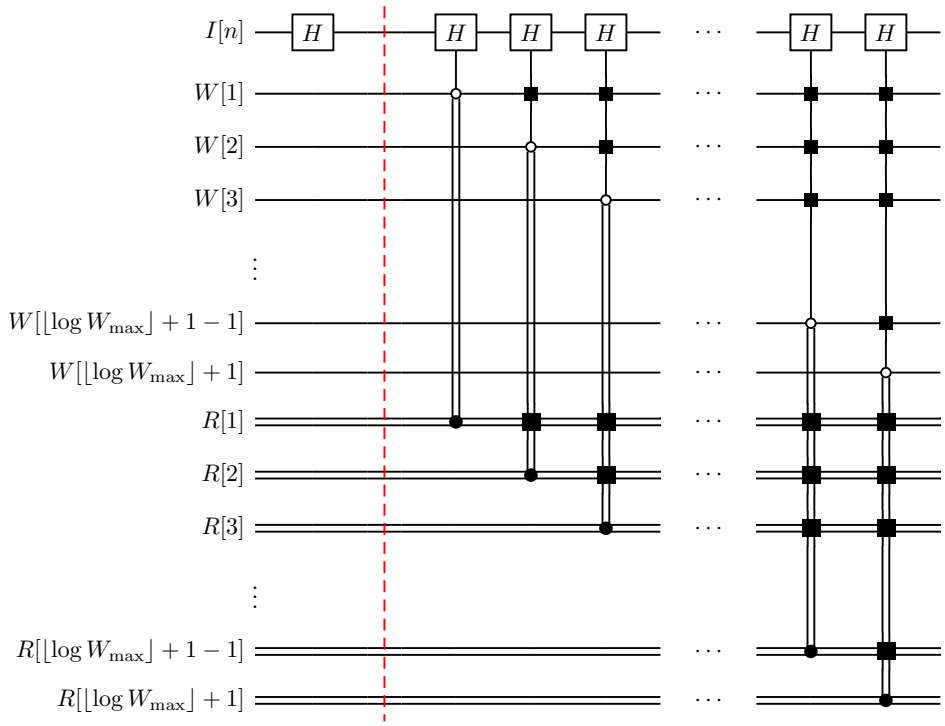


Figure 6.1.: Quantum circuit implementing the first part of $\mathcal{G}_n$, putting the n^{th} item-register qubit in a superposition of being included and excluded in the knapsack based on whether it is still affordable, inspired by the digital comparator logic. We spare showing all N qubits in register I, since $\mathcal{G}_n$ only acts on the n^{th} one by design (cf. Section 5.1, especially Eq. (5.1.1)). The red line separates the two phases where $w_n \leq w_{\max}$ is assumed first and $w_n > w_{\max}$ checked afterwards to potentially rollback. The black squares denote a control on the equality of the corresponding pair of W and R entries with the same position. Since R is a classical register, just retrieving the bit value of $(w_n)_j$ implies whether H is controlled on $W[j]$ in state $|0\rangle$ or $|1\rangle$. This is, the third H gate (counted from the left) is simultaneously controlled on qubit $W[1]$ being in state $|R[1]\rangle$ and $W[2]$ being in state $|0\rangle$ if $R[2] = 1$.

[1] In this thesis, I exclusively use the Quantikz package [Kay23] for drawing quantum circuits. Please consult Appendix C for a brief introduction into the common notation.

This circuit works as intended for the same reason the digital comparator defined via Eq. (6.1.1) obeys the desired behavior; there is especially at most one H gate being actually applied on $I[n]$ after the red line, since the different controls are indeed mutually exclusive. Concerning the classical controls on bit value 1 in the R-register, they translate to simple "If" conditions in the actual implementation: $R[j] = 0$ automatically means that the respective control failed and the corresponding Hadamard does not need to be applied at all. Summarizing, this first part of $\mathcal{G}_n$ can be implemented using at most $\lfloor \log W_{\max} \rfloor + 2$ gates.

6.1.2. Subtraction via Quantum Fourier Transform

With the I-register part being done, we can now discuss how (and in which cases) to update the residual capacity encoded in register W. After executing subcircuit Fig. 6.1 with input state $|x\rangle_I |w_{\max}\rangle_W$, the n^{th} item-register qubit is in an equal superposition of states $|0\rangle$ (meaning that it is exluded) and $|1\rangle$ (corresponding to it being included) if its weight does not exceed the remaining capacity. Only in this case, the value stored in register W needs to be adjusted at all. However, the update clearly just needs to be performed for the path where the item was selected. Therefore, we need to apply a subtraction controlled on qubit $I[n]$ being in state $|1\rangle$. Among different possibilities to implement addition (or subtraction) on a quantum computer, there is one standard approach by Draper [Dra00] that uses the *Quantum Fourier Transform (QFT)*.

To reconstruct the setting in [Dra00], let $|a\rangle = \bigotimes_{l=1}^{L} |a_n\rangle = |a_1\rangle \otimes \cdots \otimes |a_L\rangle$ be an L-qubit state where $a = a_1 \cdots a_L$ is the binary representation of $a \in \mathbb{N}$. Suppose we want to add a classically given number $b \in \mathbb{N}$ to the value encoded in the corresponding register of size L, for which we assume $b \leq a$ such that it can be encoded using L (classical) bits, yielding the binary representation $b = b_1 \cdots b_L$. In line with [Dra00], introduce the notation $e(x) = e^{2\pi i x}$. Moreover, let $|\phi_l(a)\rangle := \frac{1}{\sqrt{2}} \left(|0\rangle + e\left(a/2^l\right) |1\rangle \right)$ for $l \in \{1, ..., L\}$. The Quantum Fourier Transform of $|a\rangle$ is then defined as the map $|a_l\rangle \mapsto |\phi_l(a)\rangle$, or $|a\rangle \mapsto \bigotimes_{l=1}^{L} |\phi_l(a)\rangle =: |\phi(a)\rangle$ on an aggregated level, which can be achieved using the following circuit:

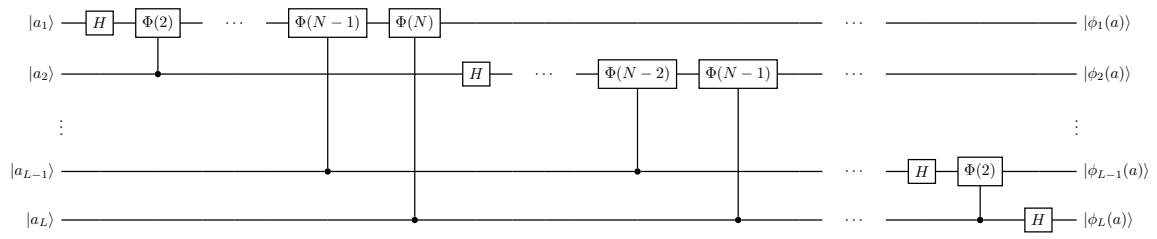


Figure 6.2.: Quantum circuit implementing QFT on a qubit register of size L.

Here, $\Phi(l) := |0\rangle\langle 0| + e\left(1/2^l\right)|1\rangle\langle 1|$ denote phase or rotation gates with $e\left(1/2^l\right)$ being the l^{th} root of unity. That the circuit shown in Fig. 6.2 indeed maps $|a_l\rangle \mapsto |\phi_l(a)\rangle \;\; \forall\, l \in \{1, ..., L\}$ can be seen by first expressing the controlled phase gates as

$$\begin{aligned}\left(C_j\Phi(k)\right)|a_l\rangle &= \begin{cases}\Phi(k)\,|a_l\rangle = \left(|0\rangle\langle 0| + e\left(1/2^k\right)|1\rangle\langle 1|\right)|a_l\rangle & \text{, if} \quad a_j = 1\\ \mathbb{1}\,|a_l\rangle = \left(|0\rangle\langle 0| + |1\rangle\langle 1|\right)|a_l\rangle & \text{, if} \quad a_j = 0\end{cases}\\ &= \left(|0\rangle\langle 0| + e\left(\frac{a_j}{2^k}\right)|1\rangle\langle 1|\right)|a_l\rangle\,, \qquad j > l,\; j,k \in \{2, ..., L\}, l \in \{1, .., L\},\end{aligned}$$

then, supposing $l < L$ in the following, realizing and exploiting the rewriting

$$\begin{aligned}\frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a_l}{2}\right)|1\rangle\right) &= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\pi i a_l}|1\rangle\right)\\ &= \begin{cases}\frac{1}{\sqrt{2}}(|0\rangle + e^0|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle & \text{, if} \quad a_l = 0\\ \frac{1}{\sqrt{2}}(|0\rangle + e^{\pi i}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle & \text{, if} \quad a_l = 1\end{cases}\\ &= H|a_l\rangle\,,\end{aligned}$$

such that the transformation of $|a_l\rangle$ may finally be comprehended step by step as

$$\begin{aligned}|a_l\rangle \mapsto H|a_l\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a_l}{2}\right)|1\rangle\right)\\ \mapsto \left(C_{l+1}\Phi(2)\right)H|a_l\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle\langle 0| + e\left(\frac{a_{l+1}}{2^2}\right)|1\rangle\langle 1|\right)\left(|0\rangle + e\left(\frac{a_l}{2}\right)|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a_l}{2} + \frac{a_{l+1}}{2^2}\right)|1\rangle\right)\\ \mapsto \quad &\cdots\\ \mapsto \left(C_L\Phi(L-l+1)\right)\cdots\left(C_{l+1}\Phi(2)\right)H|a_l\rangle\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a_l}{2} + \frac{a_{l+1}}{2^2} + \cdots + \frac{a_L}{2^{L-l+1}}\right)|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}}\right)|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}a_j 2^{-(j+1-l)}\right)|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e(0.a_l a_{l+1}\cdots a_L)|1\rangle\right)\\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a}{2^l}\right)|1\rangle\right) = |\phi_l(a)\rangle\,,\end{aligned}$$

where $0.a_l a_{l+1}\cdots a_L$ denotes the binary fraction, for which $e(0.a_l\cdots a_L) = e(a/2^l)$ holds as argued in [Dra00]. Hence, Fig. 6.2 indeed displays the QFT circuit, whose output

is

$$\begin{aligned}\mathrm{QFT}\,|a\rangle &= \bigotimes_{l=1}^{L} |\phi_l(a)\rangle = \bigotimes_{l=1}^{L} \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a}{2^l}\right)|1\rangle\right) \\ &= \frac{1}{\sqrt{2^L}} \sum_{z\in\{0,1\}^L} \left(\prod_{l=1}^{L} z_l e\left(\frac{a}{2^l}\right)\right)|z\rangle = \frac{1}{\sqrt{2^L}} \sum_{z\in\{0,1\}^L} e\left(a\sum_{l=1}^{L} z_l 2^{-l}\right)|z\rangle \\ &= \frac{1}{\sqrt{2^L}} \sum_{z\in\{0,1\}^L} e\left(a\, 0.z_1\cdots z_L\right)|z\rangle = \frac{1}{2^{L/2}} \sum_{z\in\{0,1\}^L} e\left(\frac{az}{2^L}\right)|z\rangle\,,\end{aligned}$$

from which it now becomes clear why the transformation $|a\rangle \mapsto |\phi\rangle$ is called Quantum Fourier Transform. One seemingly obvious but however non-irrelevant observation from Fig. 6.2 is that the Quantum Fourier Transform can be implemented using single- and two-qubit gates solely. By simple counting, the QFT circuit consists of

$$L + (L-1) + \cdots + 2 + 1 = \sum_{l=0}^{L-1}(L-l) = \sum_{l=0}^{L-1} L - \sum_{l=0}^{L-1} l = L^2 - \frac{L(L-1)}{2} = \frac{L(L+1)}{2}$$

gates, i.e. the amount of gates required to implement the Quantum Fourier Transform on a state that can be encoded using L qubits is of order $\mathcal{O}(L^2)$. In contrast, the best classical algorithms for computing the discrete Fourier Transform on the considered 2^L data points, such as the Fast Fourier Transform, feature a gate complexity of order $\mathcal{O}(2^L \log 2^L) = \mathcal{O}(L2^L)$, meaning that the implementation of the Quantum Fourier Transform in Fig. 6.2 provides an exponential speedup compared to its classical analogue. However, this success comes with a grain of salt, since accessing the probabilities of the final states requires numerous applications of the QFT circuit.

The circuit that performs the actual addition now highly resembles the QFT circuit:

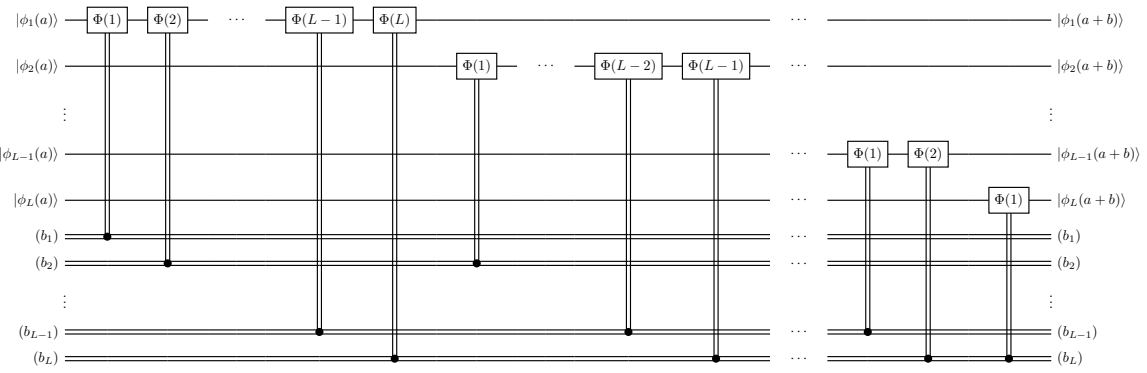


Figure 6.3.: Quantum circuit implementing the addition of a classically given number b to a QFT state $|\phi(a)\rangle$ on a qubit register of size L.

It takes $\phi_l(a) \mapsto \phi_l(a+b)$ and therefore overall implements the transformation $\phi(a) \mapsto \phi(a+b)$, following from an analogous calculation compared to the one we saw above for the QFT circuit behavior:

$$
\begin{aligned}
|\phi_l(a)\rangle \mapsto \mathrm{IF}_{b_l=1}(\Phi(1))\,|\phi_l(a)\rangle &= \mathrm{IF}_{b_l=1}(\Phi(1))\frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a}{2^l}\right)|1\rangle\right) \\
&= \mathrm{IF}_{b_l=1}(\Phi(1))\frac{1}{\sqrt{2}}\left(|0\rangle + e(0.a_l a_{l+1}\cdots a_L)\,|1\rangle\right) \\
&= \frac{1}{\sqrt{2}}\mathrm{IF}_{b_l=1}(\Phi(1))\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}}\right)|1\rangle\right) \\
&= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}} + \frac{b_l}{2^1}\right)|1\rangle\right) \\
\mapsto \mathrm{IF}_{b_{l+1}=1}(\Phi(2))\mathrm{IF}_{b_l=1}(\Phi(1))\,|\phi_l(a)\rangle & \\
&= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}} + \frac{b_l}{2^1} + \frac{b_{l+1}}{2^2}\right)|1\rangle\right) \\
\mapsto \quad \cdots & \\
\mapsto \mathrm{IF}_{b_L=1}(\Phi(L-l+1))\cdots\mathrm{IF}_{b_{l+1}=1}(\Phi(2))\mathrm{IF}_{b_l=1}(\Phi(1))\,|\phi_l(a)\rangle & \\
= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}} + \frac{b_l}{2^1} + \frac{b_{l+1}}{2^2} + \cdots + \frac{b_L}{2^{L-l+1}}\right)|1\rangle\right) & \\
= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\sum_{j=l}^{L}\frac{a_j}{2^{j+1-l}} + \sum_{j=l}^{L}\frac{b_j}{2^{j+1-l}}\right)|1\rangle\right) & \\
= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(0.a_l a_{l+1}\cdots a_L + 0.b_l b_{l+1}\cdots b_L\right)|1\rangle\right) & \\
= \frac{1}{\sqrt{2}}\left(|0\rangle + e\left(\frac{a}{2^l} + \frac{b}{2^l}\right)|1\rangle\right) = |\phi_l(a+b)\rangle\,. &
\end{aligned}
$$

Since the bits $b_1, ..., b_L$ are given classically, we here face the same situation as in the discussion of the digital comparator (cf. Section 6.1.1), namely that the controls on these classical bits simply evaluate to classical "If" statements, which are just a matter of coding. As a consequence, the circuit depicted in Fig. 6.3 may imply the application of $L(L+1)/2$ gates, but it does not have to. Notice that in contrast to the QFT circuit where the Hadamard gates require a certain computing order, all operations in the circuit in Fig. 6.3 commute. Therefore, multiple gates acting on different qubits may be collected in groups which can then be executed simultaneously, yielding a computing time speedup due to parallelism. Among several possibilities, we could, for instance, combine all phase gates controlled by the same classical bit b_l. The result is displayed in Fig. 6.4.

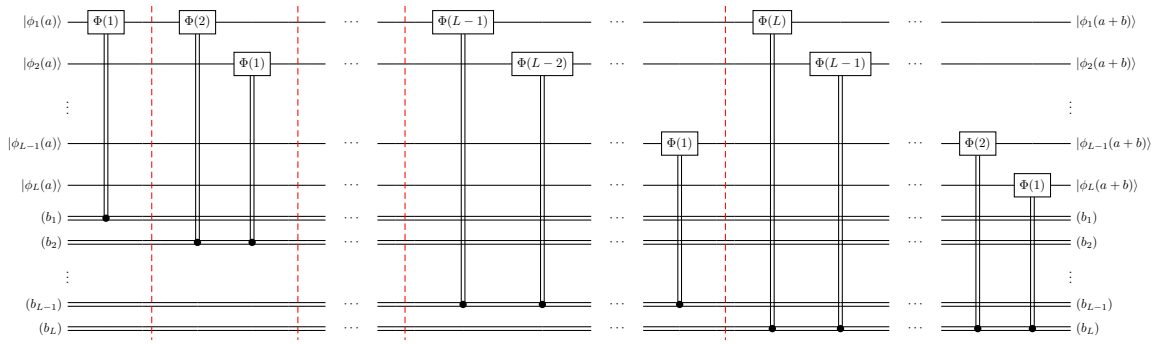


Figure 6.4.: Quantum circuit implementing the addition of a classically given number b to a QFT state $|\phi(a)\rangle$ on a qubit register of size L with operations being parallelized according to the classical bit on which they are controlled. The red lines separate the gate collections that are executed simultaneously.

Assuming a quantum computer to have the necessary capabilities to execute up to L single-qubit operations in parallel, Fig. 6.4 implies that the quantum addition transforming $|\phi(a)\rangle \mapsto |\phi(a+b)\rangle$ may be performed within a maximum L time slices.

However, the original motivation of adding a classically given number b to a quantum state $|a\rangle$ suggests to figure out how to realize the transformation $|a\rangle \mapsto |a+b\rangle$. Based on what we saw so far, and denoting the circuit in Fig. 6.4 by Add(b), this can now easily be achieved via the composed routine

$$|a\rangle \xmapsto{\text{QFT}} |\phi(a)\rangle \xmapsto{\text{Add}(b)} |\phi(a+b)\rangle \xmapsto{\text{QFT}^{-1}} |a+b\rangle\,.$$

Coming back to our original motivation of subtracting the weight w_n of the n^{th} item, given as input to our KP instance, from the state $|w_{\max}\rangle_W$ encoding the residual capacity in the $(\lfloor \log W_{\max} \rfloor + 1)$-qubit register W, we can now see that adapting the quantum addition routine by Draper [Dra00] to implement subtraction instead is indeed pretty simple - just reverting the signs in the phase gates applied in Figs. 6.3 and 6.4 yields that the b-terms get subtracted from the a-sum in the exponential throughout the above derivation following Fig. 6.3. This is, $\Phi(n)$ is replaced by its inverse $\Phi(n)^{-1} = \Phi(-n)$ in the adjusted circuit, which shall then accordingly be referred to as Sub(w_n). The parallelized version of this circuit is, for the sake of completeness, drawn in Fig. 6.5.

Combining Fig. 6.2 and a controlled version of the circuit in Fig. 6.5 now yields the final quantum circuit that implements the conditional subtraction of w_n from the capacity register based on whether the n^{th} item register qubit is in state $|1\rangle$, which is the case if and only if the check $w_n < w_{\max}$ evaluated to "true" (cf. Section 5.1). Consult Fig. 6.6 for a schematic summary.

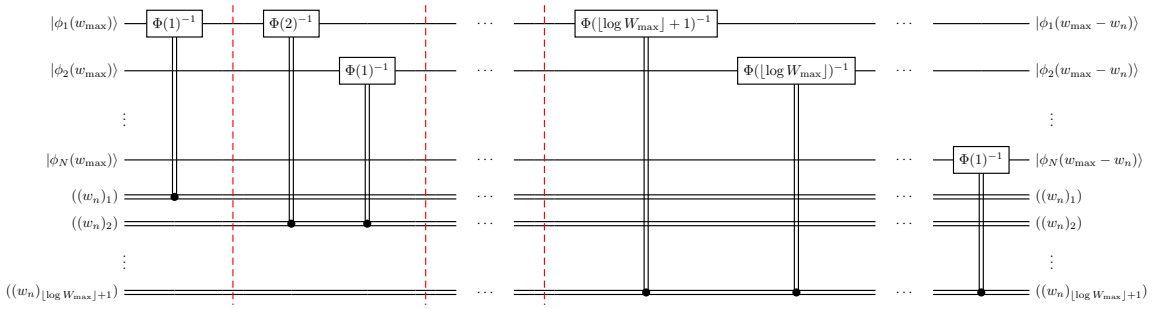


Figure 6.5.: Quantum circuit implementing $\mathrm{Sub}(w_n)$.

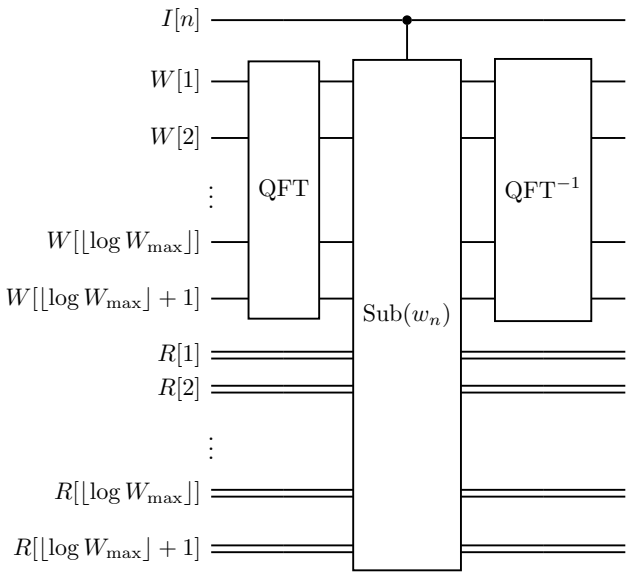


Figure 6.6.: Schematic quantum circuit for the full controlled quantum subtraction of w_n from the value encoded in the capacity register W, conditioned on the state of the n^{th} I-register qubit.

What shall be clarified is how the whole subtraction circuit (Fig. 6.5) is supposed to be controlled in Fig. 6.6. In this case, it is indeed straight forward, since the $\mathrm{Sub}(w_n)$ circuit only consists of single-qubit gates (controls on classical bits transform to classical "If" conditions): Each of them is controlled on $I[n]$ being in state $|1\rangle$ individually, making the inverse phase operations two-qubit control gates. Of course, the execution of the QFT circuit and its inverse on the W-register could also be controlled on $I[n]$ in the same way as the subtraction, since the Quantum Fourier Transform is only needed in case that the subtraction shall actually be performed. However, the resulting increase in gate complexity can be avoided, as QFT and QFT^{-1} cancel each other out anyway if $C_{I[n]}\mathrm{Sub}(w_n)$ acts as the identity.

Translating the gate counts we did for the circuits of QFT and $\mathrm{Sub}(w_n)$ to the case of $\lfloor \log W_{\max} \rfloor + 1$ qubits, the full controlled quantum subtraction shown in Fig. 6.6 necessitates the application of at most $3(\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 2)/2$ gates.

Taking into account parallelization implies that this can be achieved within a maximum of

$$2\frac{(\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 2)}{2} + (\lfloor \log W_{\max} \rfloor + 1) = (\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 3)$$

time slices. Both of these results still belong to the regime $\mathcal{O}(\log(W_{\max})^2)$.

6.1.3. Full Circuit

Now where both components - the item inclusion and the weight subtraction both based on its affordability - have been worked through, concatenating them yields the final circuit for $\mathcal{G}_n$:

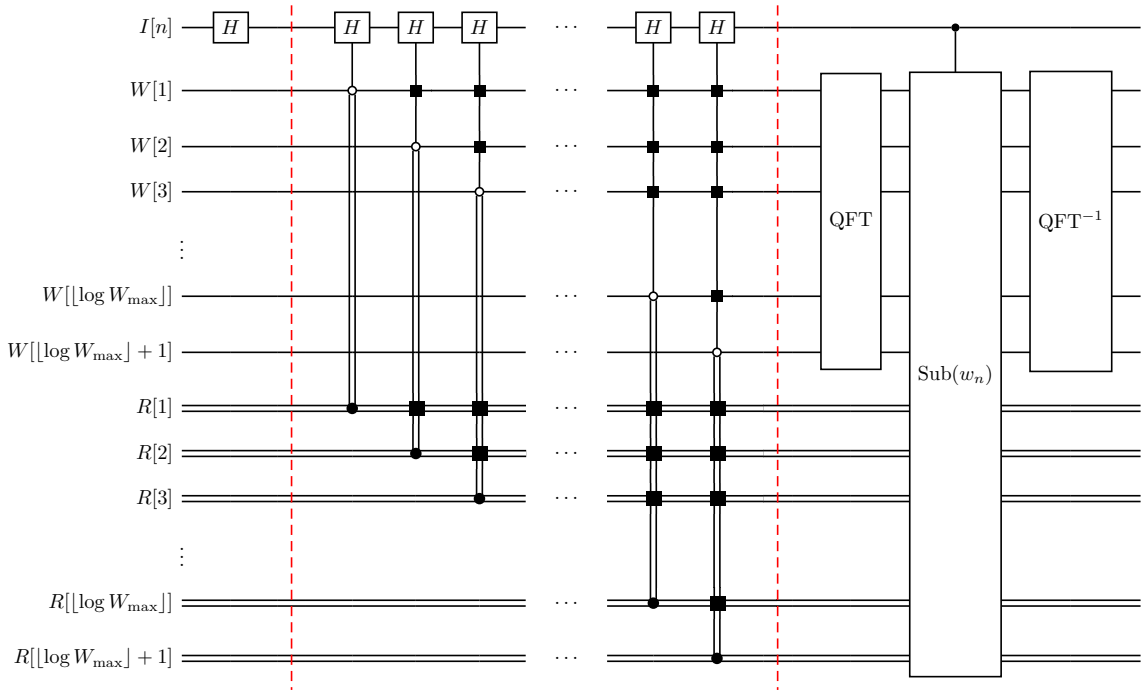


Figure 6.7.: Quantum circuit implementing $\mathcal{G}_n$.

Fig. 6.7 features a complexity evaluating to $\mathcal{O}(\log(W_{\max})^2)$ thanks to an upper limit of

$$\begin{aligned}(\lfloor \log W_{\max} \rfloor + 2) + (\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 3) \\ = (\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 4) + 1\end{aligned}$$

under the assumption of parallelization or

$$(\lfloor \log W_{\max} \rfloor + 2) + \tfrac{3}{2}(\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 2) \\ = (\lfloor \log W_{\max} \rfloor + 2) \left(\tfrac{3}{2} \lfloor \log W_{\max} \rfloor + \tfrac{7}{2} \right).$$

without. Fig. 6.7 is even sufficient to obtain the circuit for the full quantum tree generation procedure, as $\mathcal{G} = \prod_{n=1}^{N} \mathcal{G}_n$ implies that it is just given by cascading all $\mathcal{G}_n$ circuits. However, recall from Section 5.2 that the final state preparation of our Grover-mixer QAOA is not given by $\mathcal{G}$ but G instead, ensuring that the desired initial state is prepared from $|0\rangle_I |0\rangle_W$ (cf. Eqs. (5.1.2) and (5.2.2)). Also recall that we postponed the discussion of how to unitarily transform $|0\rangle_W \mapsto |W_{\max}\rangle_W$. Here is the appropriate place to elaborate on that. Since the capacity $W_{\max}$ is classically given as input to the respective KP instance at hand, we just need to retrieve its binary representation in the sense of Eq. (1.2.5) and flip all qubits whose position correspond to a bit value 1 in this binary representation of $W_{\max}$, see Fig. 6.8.

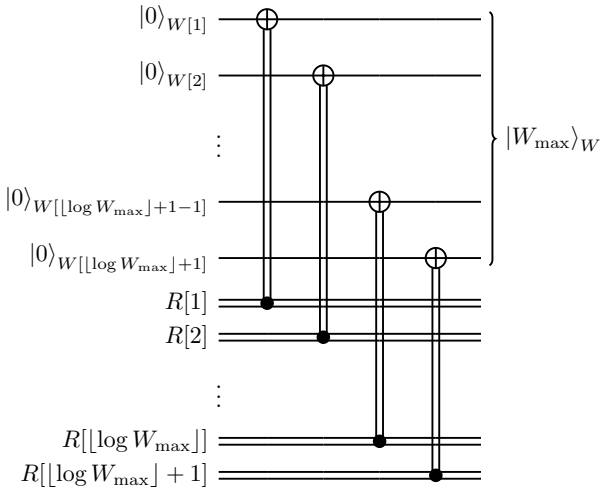


Figure 6.8.: Quantum circuit implementing the transformation $|0\rangle_W \mapsto |W_{\max}\rangle_W$. The classical register R is used to encode the capacity $W_{\max}$.

By the usual reasoning, Fig. 6.8 tells that the number of gates required to transform $|0\rangle_W \mapsto |W_{\max}\rangle_W$ is bounded from above by $\lfloor \log W_{\max} \rfloor + 1$. Thanks to parallelization, it may be executed in only one time step. For being composed of single-qubit unitary gates, the operation implemented by the circuit in Fig. 6.8 is clearly unitary.

Now we have everything together to provide the full quantum circuit for the state preparation G, see Fig. 6.9.

By simply adding up exact values and upper bounds for the gate cost of the single constituents displayed in Fig. 6.9, the whole state preparation procedure has a gate requirement smaller or equal to

$$(\lfloor \log W_{\max} \rfloor + 1) + N(\lfloor \log W_{\max} \rfloor + 2) \left(\tfrac{3}{2} \lfloor \log W_{\max} \rfloor + \tfrac{7}{2} \right).$$

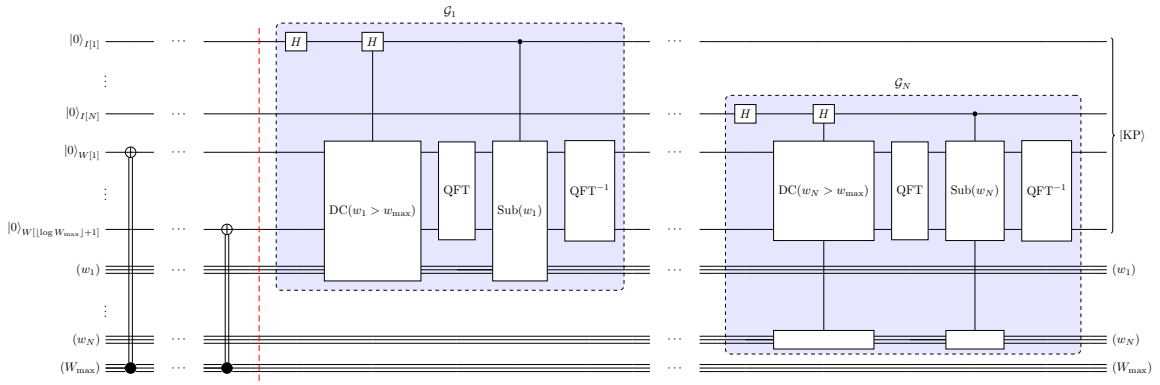


Figure 6.9.: Quantum circuit implementing G. The red line separate the initial transformation $|0\rangle_W \mapsto |W_{\max}\rangle_W$ and the application of $\mathcal{G}$. There is one classical register of size $\lfloor \log W_{\max} \rfloor + 1$ per $\mathcal{G}_n$ subcircuit, encoding the corresponding weight w_n, plus one of the same size for the capacity. The larger control symbols on this last classical register represent the decomposition shown in Fig. 6.8. The gate $\mathrm{DC}(w_n > w_{\max})$ abbreviates the Digital Comparator part of the circuit in Fig. 6.1. Empty gates in this context mean that the respective classical register used in the circuits in Fig. 6.1 or Fig. 6.5 is separated from the capacity register W by some other classical register(s).

By employing parallelization, the execution of G according to Fig. 6.9 may be streamlined to at most

$$1 + N\big((\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 4) + 1\big)$$

time slices. Both counts correspond to a complexity $\mathcal{O}(N \log(W_{\max})^2)$.

6.2. Grover Mixing and Phase Separation Unitaries

As discussed in Chapter 5, the state preparation represents the largest part in our Grover mixer approach for the Knapsack Problem in terms of circuit construction and actual implementation. What remains to be done in order to achieve a fully specified QAOA implementation is to elaborate on the mixing unitaries $U_B(\beta) = e^{-i\beta|\mathrm{KP}\rangle\langle\mathrm{KP}|}$, determined by the initial $|\mathrm{KP}\rangle$, which itself is obtained via the quantum tree generation according to Eq. (5.1.2), and the phase separation unitaries $U_P(\gamma) = e^{-i\gamma P}$ with objective Hamiltonian P (cf. Section 5.2). Recall that Eq. (5.2.3) provides a more concrete expression for $U_B(\beta)$, from which the corresponding circuit can be inferred to a large extent thanks to the already derived QTG circuit for $\mathcal{G}$ in Fig. 6.9. More specifically, we only need to consider the middle part of Eq. (5.2.3), for which the

following rewriting facilitates the intuition:

$$\begin{aligned}\mathbb{1} - \left(1 - e^{-i\beta}\right) |0\rangle_I \langle 0|_I &= \left(|0\rangle_I \langle 0|_I + \sum_{z=1}^{2^N-1} |z\rangle_I \langle z|_I \right) - \left(1 - e^{-i\beta}\right) |0\rangle_I \langle 0|_I \\ &= e^{-i\beta} |0\rangle_I \langle 0|_I + \sum_{z=1}^{2^N-1} |z\rangle_I \langle z|_I\end{aligned}$$

where $\mathbb{1}$ here denotes the identity on the N item-register qubits and $|0\rangle \equiv \bigotimes_{n=1}^{N} |0\rangle_{I[n]}$ via Eq. (1.2.5). This is, $U_B(\beta)$ acts as the identity unless all qubits are in state zero, in which case it introduces a phase factor $e^{-i\beta}$. Hence, $U_B(\beta)$ can be implemented by applying the single-qubit phase gate $P^{(0)}_{I[m]}(\theta) = e^{i\theta} |0\rangle_{I[m]} \langle 0|_{I[m]} + |1\rangle_{I[m]} \langle 1|_{I[m]}$ on an arbitrary but fixed position $m \in \{1, ..., N\}$ and controlling it on all the others qubits being in state zero. The concrete qubit register entry m is indeed irrelevant, as the phase factor may be pulled through the Kronecker product. We will, by arbitrary choice, select $m = N$, such that

$$\begin{aligned}C^{(0,...,0)}_{I[1,...,N-1]} P^{(0)}_{I[N]}(-\beta) &= \left(\bigotimes_{n=1}^{N-1} |0\rangle_{I[n]} \langle 0|_{I[n]} \right) \otimes P^{(0)}_{I[N]}(-\beta) + \sum_{z=1}^{2^{N-1}-1} |z\rangle_{I[1,...,N-1]} \langle z|_{I[1,...,N-1]} \otimes \mathbb{1}_{I[N]} \\ &= e^{-i\beta} \left(\bigotimes_{n=1}^{N} |0\rangle_{I[n]} \langle 0|_{I[n]} \right) + \left(\bigotimes_{n=1}^{N-1} |0\rangle_{I[n]} \langle 0|_{I[n]} \right) \otimes |1\rangle_{I[N]} \langle 1|_{I[N]} + \sum_{z=2}^{2^N-1} |z\rangle_I \langle z|_I \\ &= e^{-i\beta} |0\rangle_I \langle 0|_I + |1\rangle_I \langle 1|_I + \sum_{z=2}^{2^N-1} |z\rangle_I \langle z|_I = e^{-i\beta} |0\rangle_I \langle 0|_I + \sum_{z=1}^{2^N-1} |z\rangle_I \langle z|_I \,.\end{aligned}$$

Based on what we just learned, Eq. (5.2.3) implies that the quantum circuit for the mixing unitary $U_B(\beta)$ can be constructed by sandwiching the $(N-1)$-fold controlled phase gate with $G^\dagger$ and G, see also [BE20]. Fig. 6.10 displays the schematic result.

Note that Fig. 6.10 underpins again why Bärtschi and Eidenbenz [BE20] sell their approach as "shifting complexity from mixer design to state preparation" - once the state preparation (here given by the quantum tree generation) is known, the implementation of the mixing unitaries is rather trivial. Obviously, the gate count of $U_B(\beta)$ is twice that for G plus one, meaning that the complexity falls in the same regime $\mathcal{O}(N \log(W_{\max})^2)$ (independent of with or without the aid of parallelism).

Now recall that the phase separation unitaries are given by $U_P(\gamma) = e^{-i\gamma P}$ as in the unconstrained case (cf. Section 2.2.2), making their implementation a lot easier. U_P acts as $U_P(\gamma) |x\rangle_I = e^{-i\gamma P} |x\rangle_I = e^{-i\gamma P(x)} |x\rangle_I$ on a computational basis state $|x\rangle_I, x \in \{0,1\}^N$, and can thereby be linearly extended to the full underlying Hilbert space $\mathbb{q}_I^N$ modeling the N item-register qubits. Recalling Eq. (1.3.1), the objective function $P(x) = P(x_1 \cdots x_N)$ adds up all the profits p_n whose corresponding items are included in the knapsack, i.e. for which $x_n = 1$; in case $x_n = 0$ the objective function

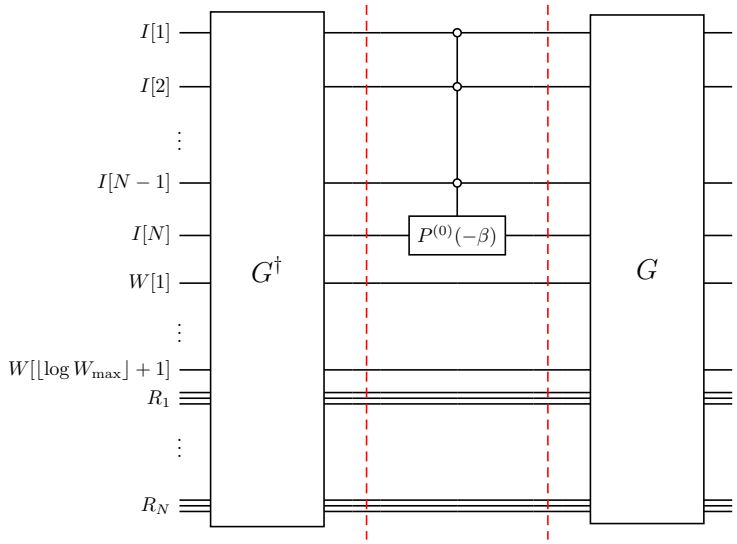


Figure 6.10.: Quantum circuit implementing the Grover mixing unitary $U_B(\beta)$ for the Knapsack Problem according to Eq. (5.2.3) with state preparation G as in Fig. 6.9. The red lines separate the three components of $U_B(\beta)$. As usual, each classical register R_n consists of $\lfloor \log W_{\max} \rfloor + 1$ classical bits.

remains unchanged. Thus, $U_P(\gamma)$ is implemented by ensuring to add a phase factor $e^{-i\gamma p_n}$ whenever $x_n = 1$ for $n \in \{1, ..., N\}$. Compare single-qubit phase gates:

$$P^{(1)}(\theta)\,|x_n\rangle = \left(|0\rangle\langle 0| + e^{i\theta}\,|1\rangle\langle 1|\right)|x_n\rangle = \begin{Bmatrix} |x_n\rangle & , \text{ if } & x_n = 0 \\ e^{i\theta}\,|x_n\rangle & , \text{ if } & x_n = 1 \end{Bmatrix} = e^{i\theta x_n}\,|x_n\rangle\,,$$

meaning that applying $P^{(1)}_{I[n]}(-\gamma p_n)$ to every item-register qubit n realizes $U_P(\gamma)$ as

$$\begin{aligned} U_P(\gamma)\,|x\rangle_I &= \left(\prod_{n=1}^{N} P^{(1)}_{I[n]}(-\gamma p_n)\right)|x_1 \cdots x_N\rangle_I = \bigotimes_{n=1}^{N} P^{(1)}_{I[n]}(-\gamma p_n)\,|x_n\rangle_{I[n]} \\ &= \bigotimes_{n=1}^{N} e^{-i\gamma p_n x_n}\,|x_n\rangle = \left(\prod_{n=1}^{N} e^{-i\gamma p_n x_n}\right)|x_1 \cdots x_N\rangle_I \\ &= e^{-i\gamma \sum_{n=1}^{N} p_n x_n}\,|x_1 \cdots x_N\rangle_I = e^{-i\gamma P(x)}\,|x\rangle_I = e^{-i\gamma P}\,|x\rangle_I\,. \end{aligned}$$

Therefore, the circuit $U_P(\gamma)$, consisting of N single-qubit gates, is as simple as:

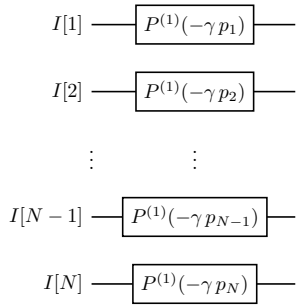


Figure 6.11.: Quantum circuit implementing $U_P(\gamma)$ for the Knapsack Problem.

Since each two phase gates in the $U_P(\gamma)$ circuit act on different qubits, every phase separation unitary may be applied in only one piece of time on a quantum computer with suitable capabilities. Joining Fig. 6.11 and Fig. 6.10 immediately yields the circuit for a single iteration $U_B(\beta_j)U_P(\gamma_j)$ with $j \in \{1, ..., p\}$. The full quasi-adiabatic evolution - described by Eq. (2.2.10) - is then implemented via cascading all these single iterations as shown in Fig. 6.12, and not forgetting about the initial state preparation procedure to obtain $|\mathrm{KP}\rangle$ from $|0\rangle_I |0\rangle_W$ in the first place.

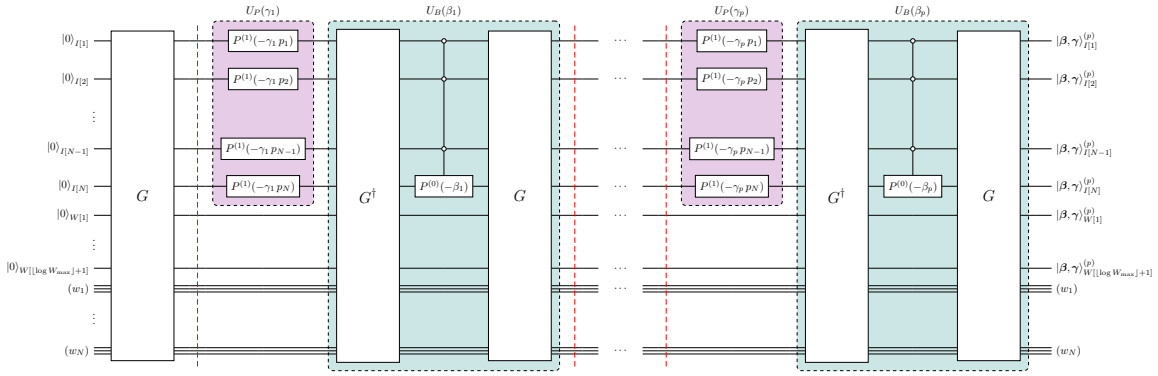


Figure 6.12.: Quantum circuit implementing the full quasi-adiabatic evolution of the Grover-mixer QAOA for the Knapsack Problem with depth p and state preparation represented by G as in Fig. 6.9. The red lines separate the different components of the quasi-adiabatic evolution, particularly the initial state preparation and the p rounds of applying $U_B(\beta)U_P(\gamma)$.

According to Fig. 6.12, the quasi-adiabatic evolution of the Grover-mixer QAOA for the Knapsack Problem obeys a gate cost of at most

$$\begin{aligned}
&\Big((\lfloor \log W_{\max} \rfloor + 1) + N(\lfloor \log W_{\max} \rfloor + 2) \left(\tfrac{3}{2} \lfloor \log W_{\max} \rfloor + \tfrac{7}{2} \right) \Big) \\
&\quad + p \Big(N + 2\big((\lfloor \log W_{\max} \rfloor + 1) + N(\lfloor \log W_{\max} \rfloor + 2)(\tfrac{3}{2} \lfloor \log W_{\max} \rfloor + \tfrac{7}{2}) \big) + 1 \Big) \\
&= (1 + 2p) \Big(\big(\lfloor \log W_{\max} \rfloor + 1 \big) + N \big(\lfloor \log W_{\max} \rfloor + 2 \big) \big(\tfrac{3}{2} \lfloor \log W_{\max} \rfloor + \tfrac{7}{2} \big) \Big) + (N + 1)p,
\end{aligned}$$

i.e. it features a gate complexity of $\mathcal{O}(pN \log(W_{\max})^2)$. Taking into account parallelization again, the quasi-adiabatic evolution can be achieved in not more than

$$\begin{aligned}
&\Big(1 + N\big((\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 4) + 1 \big) \Big) \\
&\quad + p \Big(1 + 2 \Big(1 + N\big((\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 4) + 1 \big) \Big) + 1 \Big) \\
&= (1 + 2p) \Big(1 + N\big((\lfloor \log W_{\max} \rfloor + 1)(\lfloor \log W_{\max} \rfloor + 4) + 1 \big) \Big) + 2p,
\end{aligned}$$

time steps; this complexity is still contained in $\mathcal{O}(pN\log(W_{\max})^2)$. One can argue that the overall gate complexity featuring our QAOA for the Knapsack Problem should rather be concluded to be $\mathcal{O}(N\log(W_{\max})^2)$, as the circuit depth p is not a problem-specific quantity.

6.3. Classical Angle Optimization

Without any doubt, the design of the phase separation and mixing unitaries as well as their realizations and implementations in terms of quantum circuits is - especially in the framework of this thesis - by far the most intriguing part of the QAOA. However, I do not want to sweep under the carpet the classical part in Algorithm 2 that is underlying the iteration procedure, turning the QAOA itself into a hybrid quantum-classical algorithm (cf. Section 2.2.2). More specifically, let us briefly take a look at the method we use to optimize the $2p$ angles $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)$ and $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_p)$, since this can indeed turn out as a bottleneck for the QAOA at the end of the day. Our QAOA shall employ the so-called *Nelder-Mead method*, which is a standard numerical routine to find the minimum of a function on a multidimensional domain, that was proposed by Nelder and Mead [NM65]. In a nutshell, the Nelder-Mead method for dimension n evolves a simplex consisting of $n+1$ test points according to some sophisticated rules in order to extrapolate the behavior of the objective function such that it is contracted in all directions once a valley has been found. Since the Knapsack Problem is a maximization problems (cf. Definition 1.14), we need to insert the expectation value E_p defined via Eq. (2.2.12) with reversed sign, denoted by $\overline{E}_p$, as input to the Nelder-Mead method to make it applicable here. The full procedure of this commonly used classical optimization routine can be found in Algorithm 12 in Appendix B. In my code, the Nelder-Mead method is selected out of a list of various options in the context of SciPy's built-in function `scipy.optimize.minimize` [Vir+20]. As initial conditions for the $2p$ angles, our QAOA uses random values from $[0, 2\pi)$ in order to avoid any bias influencing the optimization routine.

Simulation

After lots of work being done, we arrived at the final part of the thesis - the simulation of the algorithms we have put so much effort into. The testing and evaluation phase is in fact not less important than the construction or the implementation: it establishes comparability and enables to draw conclusions on the usability and potential of an algorithm. The largest portion of an algorithm's value only materializes in its results.

Chapter 3 was closed with the remark that there is - next to the full HQCBB - a standalone inherent motivation and scientific interest in designing a hardconstraint QAOA using the Quantum Tree Generation by Wilkening et al. [Wil+23]. Hence, we will not only evaluate Algorithm 8 but also investigate the performance of the QAOA implemented according to Chapter 6 on exemplary benchmark instances, underlining again this work's focus on the quantum physics. As the latter is embedded in the former, this is what we are going to start with. Two main types of results can here be examined in particular: the distribution of solution probabilities and the approximation ratio. In terms of the HQCBB we will in detail analyze the competing lower bounds as well as the number of explored nodes.

An important question for all of the upcoming simulations of course is how to achieve the largest possible representativity of the used KP instances. In that aim, all of them are generated randomly, based on a given number of items (the problem size), a desired ratio between the capacity and the sum of all weights and a maximum value for profits and weights. In Section 5.1 we learned that the total number of qubits required for QTG is composed of N qubits for N items and $\lfloor \log W_{\max} \rfloor + 1$ qubits to keep track the residual capacity in every step. Regarding the two parameters whose effect on the qubit requirement may not be obvious, an increasing value in both the capacity ratio and the maximum profit/weight leads to a larger capacity, which is equivalent to more qubits being necessary. Wherever meaningful, we will furthermore generate a whole bunch of equivalent random problem instances and average over them to allow more profound statements. In what follows I refrain from printing the obtained values for profits and weights as well as the capacity for the sake of readability - what will however be shown are the parameters used to generate the respective KP instance.

7.1. High-level Simulator

A major obstacle on our way to apply both our QTG-induced Grover-mixer QAOA purely and the full HQCBB where it is integrated to exemplary benchmark instances is that simulating the quantum circuit in Fig. 6.12 on a classical machine is computationally very expensive. This is however nothing that holds particularly for our QAOA, it rather refers to the general inefficiency in simulating arbitrary quantum circuits on classical computers. The simple reason for this is the number of complex amplitudes needed to be stored that is growing exponentially with the amount of qubits [Fey82].[1] Clearly, the available memory capacity of the used classical device naturally restricts the number qubits that can be simulated. Note that although a certain amount of qubits may be simulatable on your system, this does not necessarily mean that also your quantum circuit can be executed in a reasonable amount of time - whether an instance of the optimization problem in question is tractable depends, besides its qubit requirement, on the number of gates making up the respective quantum circuit. According to the survey undertaken by Smelyanskiy, Sawaya, and Aspuru-Guzik [SSA16], the best available supercomputers should by now be able to operate on 49 qubits.

Assuming such a hard limit, KP instances consisting of more than 49 items would be intractable. Even worse, $N + \lfloor \log W_{\max} \rfloor + 1 \leq 49$ would need to be satisfied, meaning that the upper bound of 49 items cannot be reached due to the capacity register. However, Wilkening et al. [Wil+23] impressively showed that we do not need to settle for that very restrictive limit. Inspired by them (and borrowing terminology), I followed a completely different approach: After setting up a high-level simulator for the QTG, I extended it by QAOA-based logic, making it also specifically tailored to the alternating application of the mixing and phase separation unitaries derived in Section 5.2. More specifically, as in [Wil+23], the QTG is simulated by generating the tree of feasible solutions using the same heuristic as in Section 5.1, i.e. respecting the branching rule given by Eq. (5.1.1), but doing that classically. This is, we apply a breadth-first search (cf. Section 2.1.4) starting from the (feasible) solution $0 \cdots 0$ where no item is selected, and store any found feasible solution together with its amplitude and the value it encodes via the binary representation. While building the tree, the amplitude of a stored feasible solution undergoes an update via being re-scaled by a factor of $1/\sqrt{2}$ whenever we find the solution with the next item being included to be feasible as well, see Fig. 5.1. The essence of the QTG is to iteratively rule out all infeasible solutions in the process of exploring the tree, which is why we do not need to store the full 2^N amplitudes for an instance consisting of N items. This classical procedure returns an array of f^N (yet) classical amplitudes with $f < 2$ and an array of

[1] Assuming a complex double precision where 8 bytes each encode the real part and the complex part, simulating n qubits necessitates to store 2^{n+4} bytes for the 2^n complex amplitudes [SSA16].

f^N binary values that enable to access the particular feasible solution via the index in these arrays.[2] So much for the simulation of the QTG. What is new compared to [Wil+23] is to classically pretend the p-fold application of the concatenated mixing and phase separation unitary for a QAOA circuit of depth p. Instead of blindly simulating the item and the capacity register, recall the actions of U_P and U_B. As described in Section 5.2 and elaborated on in Section 6.2, the phase separation unitary acts as $U_P(\gamma)\,|x\rangle = e^{-i\gamma P}\,|x\rangle = e^{-i\gamma P(x)}\,|x\rangle$ on a computational basis state $|x\rangle$. As indicated above, the array of binary values identifying the feasible solutions that was returned as part of our QTG simulation can now be used to create another array storing the profit of each feasible solution at the same index. By linear extension, this means that applying $U_P(\gamma)$ to the f^N-dimensional state vector can be imitated by re-scaling each entry with the according exponential factor corresponding to the respective feasible solution. A bit more involved is to find an analytical expression for how the mixing unitary acts on such a state vector. To figure that out, we will revisit the steps that led to Eq. (2.2.22), expressing the general Grover mixing unitaries in terms of the employed state preparation, which ultimately implied Eq. (5.2.1). Let $|z\rangle$ denote the state vector of (meanwhile) complex amplitudes throughout the quasi-adiabatic evolution; then a mixing unitary acts as

$$\begin{aligned} U_B(\beta)\,|z\rangle = e^{-i\beta|\mathrm{KP}\rangle\langle\mathrm{KP}|}\,|z\rangle &= \left(\sum_{k=0}^{\infty}\frac{(-i\beta)^k}{k!}(|\mathrm{KP}\rangle\langle\mathrm{KP}|)^k\right)|z\rangle \\ &= \left(\mathbb{1}+\sum_{k=1}^{\infty}\frac{(-i\beta)^k}{k!}\,|\mathrm{KP}\rangle\langle\mathrm{KP}|\right)|z\rangle = \left(\mathbb{1}-\left(1-e^{-i\beta}\right)|\mathrm{KP}\rangle\langle\mathrm{KP}|\right)|z\rangle \\ &= |z\rangle - \left(1-e^{-i\beta}\right)\langle\mathrm{KP}|z\rangle\,|\mathrm{KP}\rangle\,. \end{aligned}$$

Hence, when storing $|\mathrm{KP}\rangle$ as the QTG result separately besides a continuously updated f^N-dimensional array holding the complex amplitudes of $|z\rangle$, we see that simulating the application of U_B reduces to calculating the expectation value $\langle\mathrm{KP}|z\rangle$. This however does not present us with major issues: Thanks to the feasibility-preserving property of the Grover mixing unitaries (cf. Section 2.2.3) and due to U_P being diagonal in the computational basis (cf. Eq. (1.2.6)), we know that $|z\rangle$ can - at any stage of the quasi-adiabatic evolution - be written as $|z\rangle = \sum_{x\in\mathrm{feas(KP)}} z_x\,|x\rangle$, meaning that the scalar product can just be evaluated via

$$\langle\mathrm{KP}|z\rangle = \left(\sum_{y\in\mathrm{feas(KP)}}\varepsilon_y^*\,\langle y|\right)\left(\sum_{x\in\mathrm{feas(KP)}} z_x\,|x\rangle\right) = \sum_{x\in\mathrm{feas(KP)}}\varepsilon_x^* z_x\,|x\rangle$$

where ε_x denotes the amplitude that was collected by the feasible solution $x\in\mathrm{feas(KP)}$ in the course of the QTG.

[2] Assumption (ii) in Section 1.3 guarantees that the factor f characterizing the number of feasible solutions for a specific problem instance is indeed strictly smaller than 2, as otherwise any solution would be feasible and the problem thus trivially solvable.

To summarize that up, by storing four f^N-dimensional arrays - one for the (real) amplitudes of all feasible solutions after the QTG, one for their associated binary values, another one for the corresponding profits and a last one for the continuously updated (complex) amplitudes throughout the quasi-adiabatic evolution - we are able to emulate the full QAOA circuit displayed in Fig. 6.12. It is remarkable that this approach, compared to the usual execution based on simulated qubits, only needs to carry out the QTG once whereas the usual execution based on simulated qubits requires to apply it twice (one time inverted) in each lap as part of the Grover mixing unitary (see e.g. Fig. 6.10).

The name "high-level simulator" has a two-fold meaning: First and foremost, it reflects the fact that the simulation does not need to actually execute the circuits obtained in Sections 6.1 and 6.2 but is based on simple analytical formulae instead; on the other hand, this approach allows, as we will see, to tackle problem instances that are way too large for conventional simulation techniques. The success recipe for the performance of the high-level simulator can be discovered by recapping the qubit requirement of applying the QTG or, equivalently, the full QAOA on quantum device. Even though the classical QTG procedure takes $\mathcal{O}(2^N)$ steps for a KP instance consisting of N items, we can here save time thanks to not having to take into account the $\lfloor \log W_{\max} \rfloor + 1$ qubits in the capacity register. The operating dimension $2^{N+\lfloor \log W_{\max} \rfloor + 1}$, corresponding to the underlying Hilbert space, is, as discussed above, reduced to f^N with $f < 2$. Obviously, this provides an enormous speedup. Finally, I want to once more stress that our high-level simulator is developed exclusively for the Knapsack Problem and tailored explicitly to the designed Grover-mixer QAOA. Especially, we cannot expect to be able to transfer that idea and set up such an approach for different (or even arbitrary) circuits without further ado, as it badly depends on the structure of the investigated combinatorial optimization problem and the unitaries to be simulated.

The actual code is written in Python 3.9, although this is of course not obligatory. All simulations will be performed using an Intel Core i7 processor (2.0 GHz) with 40 GB of RAM. All of my code - including the classical framework of the HQCBB given by Algorithm 8, the above described high-level simulator for the QTG and the quasi-adiabatic evolution, the classical evaluation and optimization part of the QAOA as well as the different simulations together with the KP instances generated randomly for them and the obtained results - has been made publicly available at `https://github.com/PaulChr99/MasterThesis_Hybrid-Quantum-Classical-Branch-and-Bound/tree/main`.

7.2. Grover-Mixer QTG QAOA

In order to keep the significance of results growing with the reading flow, let us begin with taking a look at solution probabilities before evolving to approximation ratios.

7.2.1. Solution Probabilities

Recall that executing the Grover-mixer QAOA configured with QTG as state preparation according to Fig. 6.12 with depth p results in a final angle state of the form Eq. (2.2.10). To obtain the corresponding QAOA result this state would need to be evaluated a last time as in Eq. (2.2.12). However, here we are not interested in the optimal solution value found by the QAOA but instead in how the probabilities are distributed among the different solutions. Thanks to the chosen mixer, only feasible solutions can have a non-vanishing amplitude (a probability larger than zero) in the final angle state - the essence of our Quantum Alternating Operator Ansatz was to ensure the preservation of feasibility during the full QAOA run. On a quantum computer, a repeated measurement would provide the possibility to retrieve information about the probability distribution encoded in the final state.

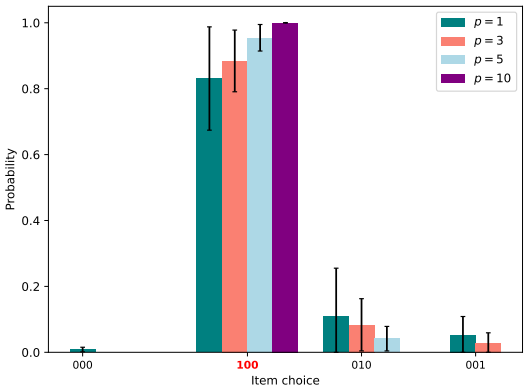
As the number of feasible solutions in general increases with the problem size, the KP instances generated for this first simulation part cannot be large (more feasible solutions imply more candidates with significant probability, making a visualization only hardly possible). As mentioned above, by increasing the capacity $W_{\max}$ we can however shift even small problem instances to the interesting regime in terms of qubits where simulations of quantum circuits are not trivial.

Since different problems generally feature different optimal solutions, the type of visualization chosen here is incompatible with considering multiple equivalent instances (meaning the same generation parameters). This is, there is only one random problem for each set of parameters. To investigate the behavior for different qubit requirements, each of the three chosen problem sizes is itself equipped with the same four values limiting profits and weights. The corresponding qubit numbers are then individually evaluated for each generated Knapsack Problem instance. That being said, the first complete set of parameters can be found in Table 7.1.

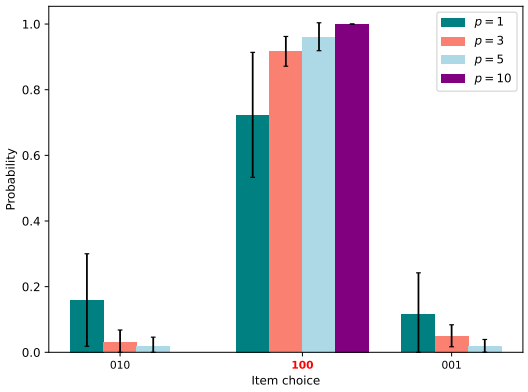
A quantity that is also well-suited varying and monitoring is the depth p of the QAOA, determining how many loops of applying the phase separation and the mixing unitaries are been taken. Its influence can be roughly studied in the first simulation results that correspond to Table 7.1.

Number of items	Capacity ratio	Maximum profit/weight	Qubit requirement
3	0.75	10^4	17
3	0.75	10^6	24
3	0.75	10^{10}	37
3	0.75	10^{18}	64

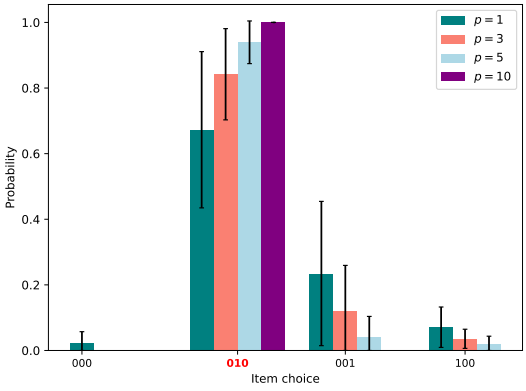
Table 7.1.: Parameters for random KP instances with 3 items and qubit requirements.



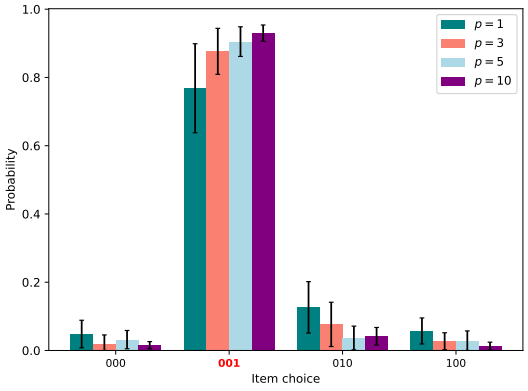
(a) Maximum profit/weight value of 10^4 and corresponding qubit requirement of 17.



(b) Maximum profit/weight value of 10^6 and corresponding qubit requirement of 24.



(c) Maximum profit/weight value of 10^{10} and corresponding qubit requirement of 37.



(d) Maximum profit/weight value of 10^{18} and corresponding qubit requirement of 64.

Figure 7.1.: Average distributions of solution probabilities for random KP instances of size 3 with a capacity ratio of 0.75 that are generated using the parameters in Table 7.1. For each instance and each depth $p \in \{1, 3, 5, 10\}$, the QAOA is executed 10 times. The error bars depict the resulting standard deviations. The optimal solutions of the problem instances are written in red.

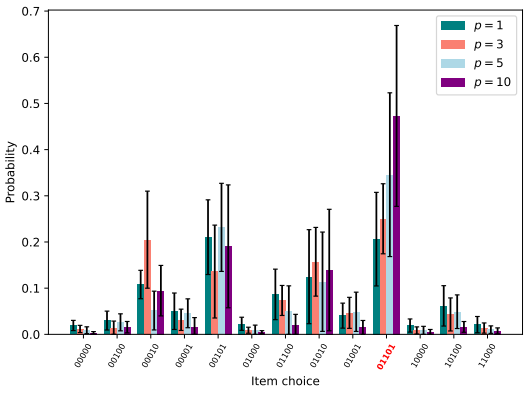
It can be considered a first validation of our Grover-mixer QAOA that the quasi-adiabatic evolution develops towards the optimal solution for any of the four problems of size 3. Moreover, we can beautifully see in Figs. 7.1a to 7.1d how the probabilities of the optimal solutions increase and approach the value 1 with growing depth. Both the slopes and the standard deviations generally seem to shrink the larger the depth gets with the standard deviation being negligible for $p = 10$, indicating that there is not much improvement to be expected for even higher depth values.

The next size is only marginally larger than the one before, making it possible to analyze the severity of changes that can be caused by a slight variation of the size:

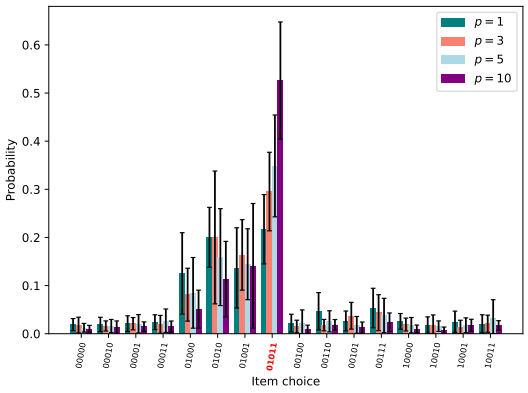
Number of items	Capacity ratio	Maximum profit/weight	Qubit requirement
5	0.5	10^4	19
5	0.5	10^6	26
5	0.5	10^{10}	39
5	0.5	10^{18}	66

Table 7.2.: Parameters for random KP instances with 5 items and qubit reuqirements.

Performing completely the same simulation that led to Fig. 7.1 for the problem instances based on Table 7.2 yields:

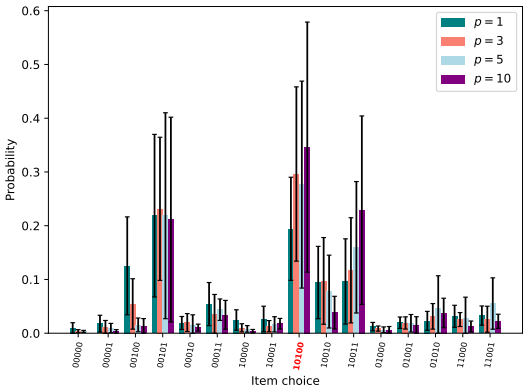


(a) Maximum profit/weight value of 10^4 and corresponding qubit requirement of 19.

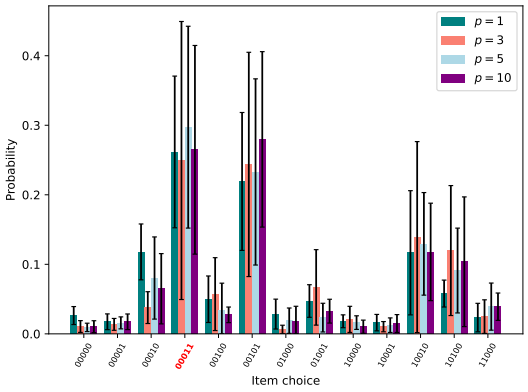


(b) Maximum profit/weight value of 10^6 and corresponding qubit requirement of 26.

Figure 7.2.: Average distributions of solution probabilities for random KP instances of size 5 with a capacity ratio of 0.5 that are generated using the parameters in Table 7.2. For each instance and each depth $p \in \{1, 3, 5, 10\}$, the QAOA is executed 10 times. The error bars depict the resulting standard deviations. The optimal solutions of the problem instances are written in red.



(c) Maximum profit/weight value of 10^4 and corresponding qubit requirement of 39.



(d) Maximum profit/weight value of 10^6 and corresponding qubit requirement of 64.

Figure 7.2.: Average distributions of solution probabilities for random KP instances of size 5 with a capacity ratio of 0.5 that are generated using the parameters in Table 7.2. For each instance and each depth $p \in \{1, 3, 5, 10\}$, the QAOA is executed 10 times. The error bars depict the resulting standard deviations. The optimal solutions of the problem instances are written in red.

And indeed, the difference between Figs. 7.1 and 7.2 is drastic. Each of the four generated instances of size 5 requires exactly two qubits more than its size-3 counterpart. It is good to see that the number of feasible solutions with non-vanishing probability has approximately quadrupled, in line with the exponential growth in dimension thanks to the increased qubit numbers. But what catches the eye most are the error bars - compared to those in Fig. 7.1 the standard deviations are fairly large relative to the corresponding average probability values; this holds for all Figs. 7.2a to 7.2d. This behavior can most probably be explained by the increased number of feasible solutions among which the probabilities are distributed while the number of QAOA executions has not grown accordingly. For the two smaller problems in terms of qubits (Figs. 7.2a and 7.2b) the respective optimal solution was nevertheless found with the largest probability, which is in turn again higher the larger the depth was chosen. For the two more qubit-expensive instances: While the optimal solution in Fig. 7.2c has a slight overweight compared to two other competing solutions, the QAOA ends up with two solutions of almost equal probabilities in Fig. 7.2d. Keep in mind that this analysis is based on average values with which large standard deviations are associated.

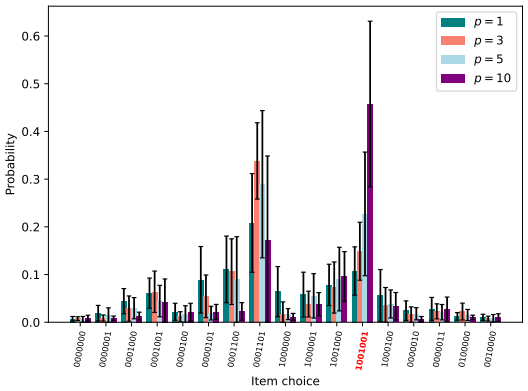
Let us now increase the number of items again by this amount of two in order to see whether that confirms our just made observations. The corresponding parameters are listed in Table 7.3.

Comparing the rightmost columns of Tables 7.2 and 7.3 we see that the size being again enlarged by two leads to a qubit requirement increased by one in the first two

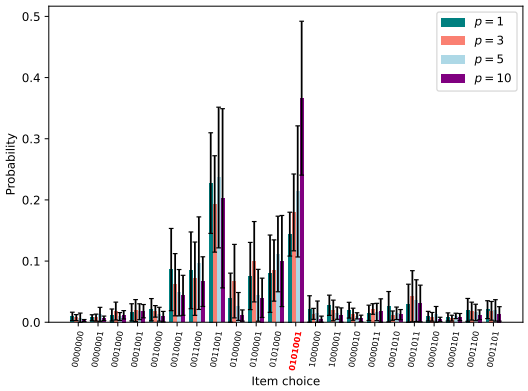
Number of items	Capacity ratio	Maximum profit/weight	Qubit requirement
7	0.25	10^4	20
7	0.25	10^6	27
7	0.25	10^{10}	41
7	0.25	10^{18}	67

Table 7.3.: Parameters for random KP instances with 7 items and qubit requirements.

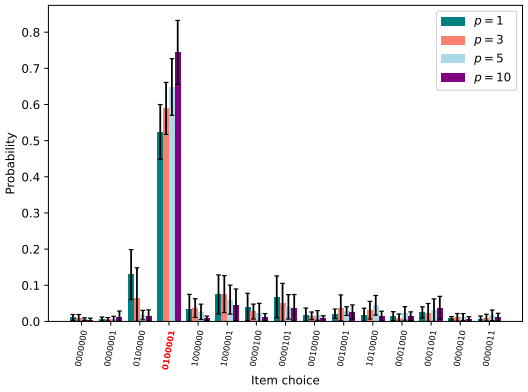
cases and by two in the second two instances. This is what the results look like:



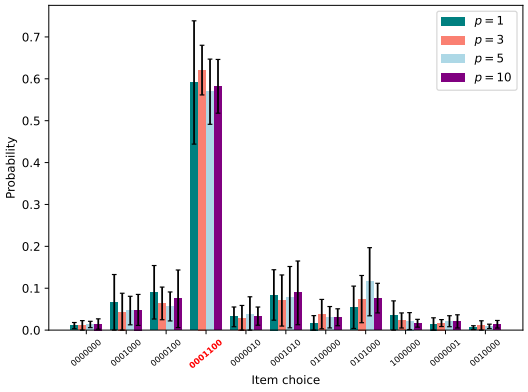
(a) Maximum profit/weight value of 10^4 and corresponding qubit requirement of 20.



(b) Maximum profit/weight value of 10^6 and corresponding qubit requirement of 27.



(c) Maximum profit/weight value of 10^{10} and corresponding qubit requirement of 41.



(d) Maximum profit/weight value of 10^{18} and corresponding qubit requirement of 67.

Figure 7.3.: Average distributions of solution probabilities for random KP instances of size 7 with a capacity ratio of 0.25 that are generated using the parameters in Table 7.3. For each instance and each depth $p \in \{1, 3, 5, 10\}$ the QAOA is executed 10 times. The error bars depict the resulting standard deviations. The optimal solutions of the problem instances are written in red.

The reader may wonder now why number of solutions with non-negligible probabilities in none of Figs. 7.3a to 7.3d has increased by a factor of two or four compared to its respective counterpart in Fig. 7.2. Normally, this is what we would expect. However, the ratio that determines how large the capacity can be relative to the sum of all weights has been halved in the transition from Table 7.2 to Table 7.3 just in order to prevent too large amount of feasible solutions to be printed. Reducing the capacity ratio has also been done in the first step of increasing the size (compare Tables 7.1 and 7.2) but it was there cut by only a third.

Apart from that, Figs. 7.3a and 7.3b structurally resemble all of the results in Fig. 7.2 - large error bars and an ambiguous balance in two alleged best solutions in each case. Even further, only at the highest depth 10 the respective optimal solution ends up with the top probability. The largest two KP instances in this simulation part show, in contrast, a different behavior. Their results in Figs. 7.3c and 7.3d look more like what we found for size 3 in Fig. 7.1: The optimal solutions are clearly hit with probabilities between 50% and 70% for any depth and standard deviations are back in reasonable orders of magnitude. This also shows that an enlarged set of feasible solutions does not always imply a rise in error bars.

All together, simulating the distributions of solution probabilities for different sizes, different qubit amounts and different depths gave us a good first impression of how to assess the outcome of our Grover-mixer QAOA for the Knapsack Problem. The results in Figs. 7.1 to 7.3 indicate the success of our approach. However, the chosen way of displaying the results is only feasible for very small problem instances where the number of feasible solutions with non-vanishing probability is low enough to resolve all of them for the different depths in one graphic. It is moreover not suited for quantitatively comparing the results obtained for different triples of parameters. Obviously, the sizes analyzed here do not justify to draw general conclusions about the performance of the QAOA, even though the qubit amount has been artificially up-scaled to make the problems worth investigating.

7.2.2. Approximation Ratios

Probably the most common quantity to analyze when constructing a QAOA is the so-called approximation ratio. Given a problem instance, the *approximation ratio* is defined as the share of the QAOA result by the optimal solution value. By construction, a QAOA provides a lower bound to the optimal solution value of a maximization problem (cf. Section 2.2.2), meaning the approximation ratio takes values between 0 and 1. Compared to the approach in Section 7.2.1, investigating the approximation ratio seems to be less informative, as it does not tell anything about the probability

distribution but only evaluates the expectation value of the final angle state according to Eq. (2.2.12), in which the single probabilities are aggregated. So why would we want to take this road? First and foremost, the format of bar charts in Figs. 7.1 to 7.3 makes it difficult to compare different problem sizes (in terms of items or qubits) and to quantitatively the dependency on the depth. But most importantly, the approximation ratio represents the relative version of the quantity that is actually used for the lower bound comparison in Algorithm 8 or, more precisely, in Algorithm 10. Also, even though the approximation does not directly contain any information about the probabilities of individual solutions, its height of its value tells us something about how close we are to the optimal solution, since that by definition corresponds to the maximum objective function value (while feasibility is preserved). Corollary 2.3 states that the solution found by the QAOA approaches the optimal solution value in the limit $p \to \infty$. From this we derive the expected behavior of the approximation ratio to increase with growing depth. Our expectation shall be verified for different qubit sizes, for which two parameters in the generation of KP instances will be varied: the maximum value of profits and weights for a fixed combination of size and capacity ratio and then the number of items with the other two being fixed.

Varying Maximum Profit/Weight Value

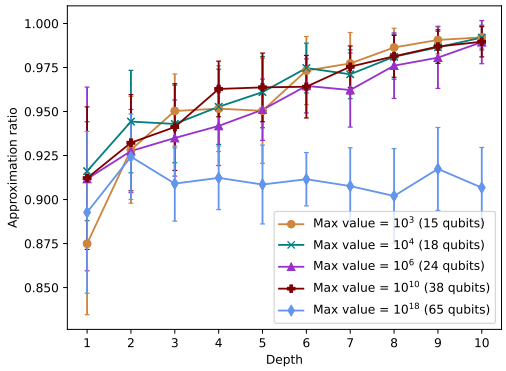
Starting with the maximum value for profits and weights, we will iterate it over a set of five numbers for four different problem sizes in order to not lose generality by restricting to only one number of items. The values by which profits and weights are to be limited are again chosen deliberately high, as in Section 7.2.1 to up-scale the qubit requirements, making the simulation even more attractive. The parameters used for the generation of problem instances are as follows:

Number of items	Capacity ratio	Maximum profit/weight	Qubit requirement
5	0.25	$\{10^3, 10^4, 10^6, 10^{10}, 10^{18}\}$	{15, 18, 24, 38, 65}
20	0.07	$\{10^3, 10^4, 10^6, 10^{10}, 10^{18}\}$	{30, 34, 40, 53, 80}
40	0.033	$\{10^3, 10^4, 10^6, 10^{10}, 10^{18}\}$	{50, 53, 60, 73, 100}
60	0.026	$\{10^3, 10^4, 10^6, 10^{10}, 10^{18}\}$	{70, 73, 80, 93, 120}

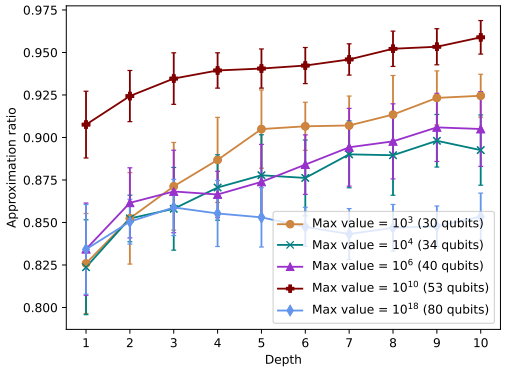
Table 7.4.: Parameters for random KP instances with varying maximum profit/weight value and corresponding qubit requirements.

As the focus is here on comparing the approximation ratio behavior for problem sizes

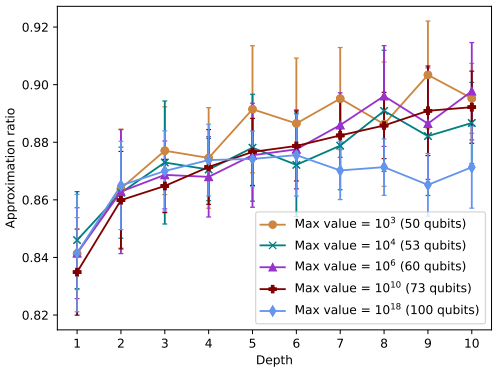
induced by different maximum profit/weight values, the simulation results are separated by the number of items, yielding four graphics:



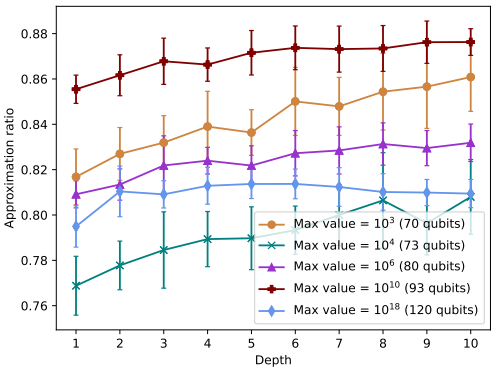
(a) Simulation results for a size of 5 and a capacity ratio of 0.25.



(b) Simulation results for a size of 20 and a capacity ratio of 0.07.



(c) Simulation results for a size of 40 and a capacity ratio of 0.033.



(d) Simulation results for a size of 60 and a capacity ratio of 0.026.

Figure 7.4.: Average approximation ratios for random KP instances generated using the parameters in Table 7.4. For each parameter combination the simulation is performed for five problem instances with the same specification, where it is ensured that all of them are of the same qubit size. For each of generated instance, the QAOA is executed five times. The error bars depict the averages of the resulting standard deviations for the equivalent KP instances.

The first good news is that the tendency of the approximation ratio to grow when the depth is increased can be recognized in all of Figs. 7.4a to 7.4d - confirming our above formulated expectation. While this is apparently definitely true for the data series corresponding to the four smaller problem instances in terms of qubits at each number of items, the data points associated with the largest maximum profit/weight values seem to not share this behavior - they experience a brief rise before dropping slightly

again or stagnating in every case. However, we must not forget the schema according to which the maximum values for profits and weights were chosen in Table 7.4: The exponent is doubled in every step and the qubit requirement shares this doubling procedure to a good approximation, which can be traced back to its logarithmic scaling with the capacity. Hence, while the three smallest instances have a somewhat close qubit need at each number of items, especially the last steps lead to big gaps in qubit size. A possible explanation for the non-conform course of the data series corresponding to those qubit-largest problems could therefore be that the chosen depths are simply too low to find the expected behavior for these strongly more expensive instances.

Also conspicuous is that, except for a maximum profit/weight value of 10^{18}, the data points for the different problem instances are confined to a small band in Figs. 7.4a and 7.4c whereas the whole data series are shifted horizontally and intersect each other only rarely in Figs. 7.4b and 7.4d. Beyond, the order in which the data series appear (started from top or bottom) is equal comparing the latter two graphics. It may indeed be not predictable which of these two options will actually apply when varying the maximum value for profits and weights for a given number of items - recall that the capacity ratio is kept unchanged in each of Figs. 7.4a to 7.4d.

A conclusion that can indubitably be drawn from Fig. 7.4 is that the quality of the approximations calculated by our QAOA is solid, as the results find themselves even for the most shallow depths to not be worse than 75%. Considering the actual values of the obtained approximation ratios, an average decrease can be identified for numbers of items getting larger. However, this is something for which an in detail analysis of different sizes is better suited.

Varying Number of Items

Vice versa, the number of items is now varied for four values limiting profits and weights of different magnitudes. In contrast to the procedure above, the size is not iterated over the same set of values - instead, the intervals out of which the numbers of items are chosen are shifted to larger regions. More specifically, we set up four sets in each of which four sizes are picked from a range of length 10. However, the capacity ratios for the four sections shall be assigned as in Table 7.4. The maximum value for profits and weights in turn is supposed to accompany the four regimes with a uniformly growing exponent. An overview of the parameter combinations is given by Table 7.5.

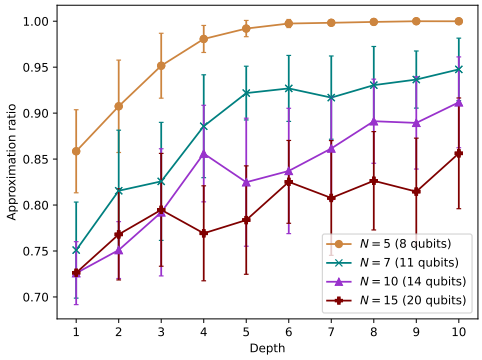
Except from the tiny regime (first row in Table 7.5) where three, four or five ancilla qubits are needed to store the capacity, the discrepancy between qubit requirement and number of items is always constant per regime for the chosen pairs of capacity ratio

Number of items	Capacity ratio	Maximum profit/weight	Qubit requirement
{5, 7, 10, 15}	0.25	10	{8, 11, 14, 20}
{20, 23, 26, 30}	0.07	10^6	{40, 43, 46, 50}
{35, 38, 40, 45}	0.038	10^{12}	{75, 78, 80, 85}
{50, 54, 57, 60}	0.027	10^{18}	{110, 114, 117, 120}

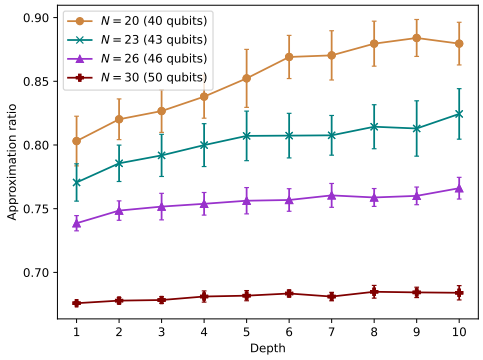
Table 7.5.: Parameters for random KP instances with varying number of items and corresponding qubit requirements.

and maximum profit/weight value. Even further, this difference increases by exactly 20 in the transition between the larger three regimes.

Generating analogously structured simulation results as in Fig. 7.4 with the roles of maximum profit/weight value and size being flipped yields:



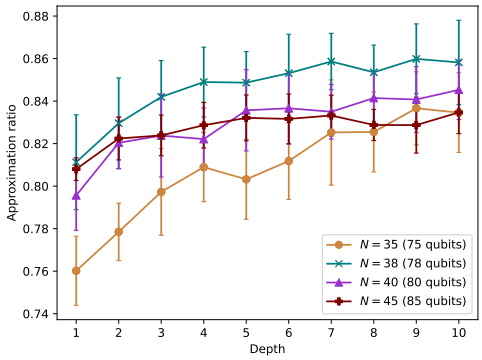
(a) Simulation results for the tiny size regime with a capacity ratio of 0.25 and a maximum profit/weight value of 10.



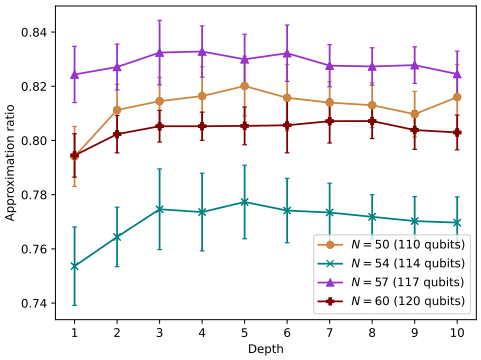
(b) Simulation results for the small size regime with a capacity ratio of 0.07 and a maximum profit/weight value of 10^6.

Figure 7.5.: Average approximation ratios for random KP instances generated using the parameters in Table 7.5. For each parameter combination the simulation is performed for five problem instances with the same specification, where it is ensured that all of them are of the same qubit size. For each generated instance, the QAOA is executed five times. The error bars depict the averages of the resulting standard deviations for the equivalent KP instances.

Considered over the full depth interval, the three smallest regimes in Figs. 7.5a to 7.5c meet our expectation of an approximation ratio that increases as the depth is enlarged. Our smallest set of instances consisting of $N = 5$ items shows in full beauty how



(c) Simulation results for the medium size regime with a capacity ratio of 0.038 and a maximum profit/weight value of 10^{12}.



(d) Simulation results for the large size regime with a capacity ratio of 0.027 and a maximum profit/weight value of 10^{18}.

Figure 7.5.: Average approximation ratios for random KP instances generated using the parameters in Table 7.5. For each parameter combination the simulation is performed for five problem instances with the same specification, where it is ensured that all of them are of the same qubit size. For each generated instance, the QAOA is executed five times. The error bars depict the averages of the resulting standard deviations for the equivalent KP instances.

the QAOA result can approximate the optimal solution value for a growing depth, including continuously shrinking standard deviations. The here found monotonicity cannot be discovered to the same ideal extent in the other data series. While the three other curves in Fig. 7.5a manage to overall overcome an absolute amount of approximately 15 percentage points, the KP instances in the small size regime expose a behavior that can be best described as something between slight growth and stagnation. The largest set of problems does not even reach the 70% in terms of approximation. The effect of the size is most impressively visible in Fig. 7.5b: The larger the problem instance in terms of variables or qubits the more complicated it obviously gets for the QAOA to evolve to the optimal solution at a constant depth due to the generally enlarged number of feasible solutions - having visible negative consequences for the approximation ratio even for slightly varied sizes. While the order is the same in Fig. 7.5a for the tiny size regime, there is not even a single intersection in Fig. 7.5b. Both is different for the larger two regimes: In Fig. 7.5c another than the smallest set of instances is achieving the best results for the first time; pretty untypically, the problems with $N = 35$ actually show the worst performance here. In the medium size regime it looks as if the approximation ratios would not approach 1 but a different value smaller than 0.86; this observation has not been made collectively in any of the other approximation ratio simulations in Figs. 7.4 and 7.5. Turning to the large size regime, we can recognize a similar behavior in Fig. 7.5d compared to the qubit-expensive problem sets in Fig. 7.4: For none of the sizes we find the average approximation ratio

at the largest depth of 10 to occupy a global maximum among its corresponding data series. The explanation given there could also apply here: It would simply be necessary to extend the depth range of the simulation by larger in order to observe the expected behavior of an approximation ratio that increases with growing depth. As we can see from Table 7.5, the qubit requirements for all of the four KP instance sets in the large regime are pretty high, meaning that the application of the QAOA becomes very expensive.

Nevertheless, the simulation results for varying the number of items underpin the conviction gained from varying the maximum value for profits and weights in Fig. 7.4 that our QAOA produces reliable results. We found only one set of KP instances in Fig. 7.5 whose approximations stay below 70% relative to the optimal solution value for all depths, namely those with $N = 30$ items in the medium size regime. Furthermore, our QAOA managed to evolve to the optimal solution with a probability bordering certainty of about and over 90% for smaller KP instances requiring 5, 11-14 and even 40 qubits. On the other hand, the maximum approximation ratios achieved in each of the regimes in turn support the hypothesis that the approximation provided by the QAOA can in general be expected to anti-correlate with the size of the problem in terms of items/qubits.

7.3. Full HQCBB

Now where we have sufficiently investigated the performance of the pure Grover-mixer QAOA, we can turn to evaluating the full HQCBB. Before analyzing global properties, let us first restrict to the single component of the algorithm that is affected by the quantum extension, namely the lower bounds.

7.3.1. Lower Bound Comparisons

Clearly, introducing an alternative quantum lower bound to the classical Branch and Bound happened in the hope and the good belief that it can turn out as an actual competition for its classical counterpart. That is, a HQCBB in which the QAOA outcomes are not even close to respective the Greedy lower bounds is essentially nothing else than a normal B&B. Fortunately, the simulation results Section 7.2 - especially the approximation ratios in Section 7.2.2 - give us cause for optimism.

Throughout the algorithm run, the size of the residual subproblem to compute (lower)

bounds for vary in terms of items and qubits, as different nodes correspond to different items being specified, inducing an according reduction in capacity (cf. Chapter 4). The idea now is to examine how the QAOA output compares to the Greedy lower bound for different sizes of subproblems that are associated with non-prunable nodes in the search tree. Thus, we will run the HQCBB and collect data about the obtained lower bound values in combination with the residual size whenever a lower bound is to be calculated. Afterwards we can average over the implied ratios of QAOA by Greedy for each residual size. In order to make that comparable for different KP instances, the size of a subproblem is going to be stored in relation to the input problem in question; for this, the qubit requirement will be consulted as measure.[3] As the number of required qubits is used as identifier here, the problems to be simulated shall, unlike in Section 7.2, not be determined by the number of items, a capacity ratio and a maximum value for profits and weights. Instead, the qubit amount will this time be the second fixed parameter alongside the number of items, based on which the other two are chosen to meet the specifications. As you will see, the chosen combinations in fact allow to only vary the limiting values for profits and weights and even keep the capacity ratios constant. Moreover, each number of items that was used in Section 7.2.2 when varying the maximum profit/weight value (cf. Table 7.4) is going to be revived here, sufficing for two sets of parameters. For the smallest sizes, this looks like:

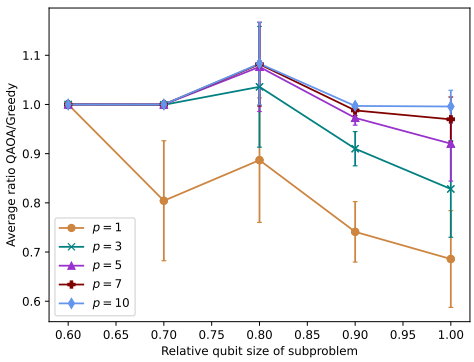
Number of items	Qubit requirement	Capacity ratio	Maximum profit/weight
5	10	0.75	10
5	20	0.75	10^4

Table 7.6.: Parameters for random KP instances of size 5.

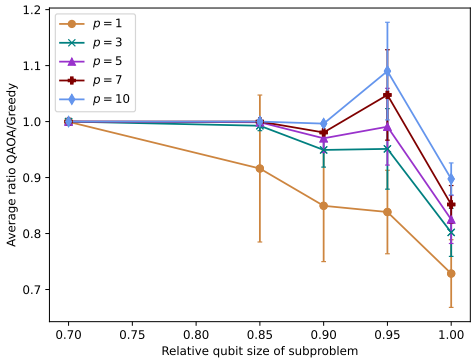
In order to properly face the slight probabilistic behavior that enters our HQCBB in Algorithm 8 via the node selection (cf. Algorithm 6) as part of the searching strategy, the algorithm shall be run four times on each generated instance. Compared to Section 7.2, the number of equivalent KP instances considered per parameter combination is up-scaled to 10. What shall - as always - furthermore be done is to analyze the influence the depth of the QAOA circuit has, which is here ranging over $\{1, 3, 5, 7, 10\}$. The results produced by the simulation for the small instances created based on Table 7.6 are displayed in Fig. 7.6.

As we are dividing the QAOA result by the respective Greedy outcome, data points

[3] In this setting there is no single correct direction of arranging the relative residual sizes on the horizontal axis. This is due to the employed depth-first search (DFS) employed in Algorithm 8, inheriting the tendency of shrinking the size of an explored subproblem to zero before continuing with more open items again to the HQCBB.



(a) Simulation results for 10 qubits, accompanied by a maximum profit/weight value of 10.



(b) Simulation results for 20 qubits, accompanied by a maximum profit/weight value of 10^4.

Figure 7.6.: Average ratios of QAOA by Greedy lower bound for random KP instances consisting of 5 items and featuring a capacity ratio of 0.75, generated using the parameters in Table 7.6. For each parameter combination the simulation is performed for 10 problem instances with the same specifications. For each generated instance, the HQCBB is run four times. The error bars depict the average of the resulting standard deviations for the equivalent KP instances.

with values larger than 1 correspond to the cases we are looking for - those where the QAOA managed to outperform the Greedy bound in the sense of Algorithm 4.

A first thing that catches the eye when considering Fig. 7.6 is that there seem to be some data points featuring a QAOA-Greedy ratio of exactly 1 and no error bars, meaning that quantum and classical lower bounds do completely match (even on average). How can that be, given that our QAOA implementation starts every run with a new set of random initial angles? The reason can be found in the input types of Algorithms 2 and 4: Since both our QAOA as well as the Greedy lower bound operate on a KP instance, the actual bound for a node in the search tree is not obtained by simply running either of these algorithms solely; instead, the outcome has to be added to the offset which is determined by the already specified items, together making a valid lower bound for the associated region of the search space, as shown in Algorithm 10 and described in Section 4.2. Now there is an edge case in the generation of a subproblem as a KP instance based on the bitstring corresponding to the current node: Right after the definition of the Knapsack Problem in Definition 1.14 in Section 1.3 we introduced three assumptions on the structure of the profits, weights and the capacity forming a valid KP instance - especially that there is no item with a weight larger than the capacity, as these items can otherwise not be contained in a feasible solution anyway. As already emphasized in Section 4.4, when creating a subproblem from a bitstring in our application, it is ensured that this assumption still holds for the subproblem

again. Since the subproblem generation reduces the capacity while simultaneously not altering single weights, the probability of a yet unspecified item whose weight exceeds the residual capacity drastically increases. In the extreme case, none of the remaining items is still affordable. When this happens, the resulting residual subproblem is empty, in which case both the Greedy lower bound and our QAOA naturally return a value of zero. As a consequence, both are just given by the calculated offset, which is why they turn out to fully agree.

Now where this question has been answered, we find four and two data points with a ratio greater than 1 in Figs. 7.6a and 7.6b, respectively, corresponding to the four and two largest depths and the same relative residual sizes of 80% and 95%. That represents an early confirmation for our QAOA being indeed capable of achieving better results compared to the Greedy lower bound; whether this persists for larger instances remains to be seen. In Fig. 7.6, the data series associated with $p = 1$ are the only ones that could never improve the classical lower bound. However, this is not surprising for such a small depth. It is even more impressive that a first slight improvement can already be found at $p = 3$ for KP instances small as consisting of only $N = 5$ items.

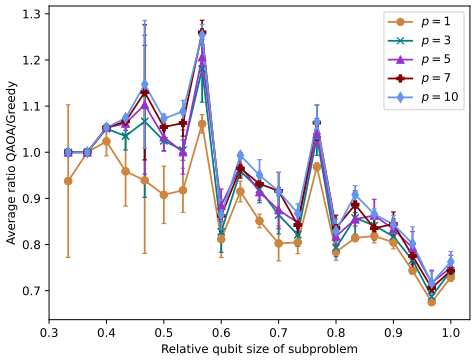
Apart from that, all curves for the same qubit sizes in Figs. 7.6a and 7.6b show pretty similar courses, especially if the lowest depth $p = 1$ is taken out of this consideration. Both figures also suggest that the QAOA results get better with growing depth - there is not even a single deviation from the expected vertical order of data points. On the other hand, the difference to the Greedy bound seems to be largest when running the bounding algorithms on the original problem instance. Due to the peaks in Figs. 7.6a and 7.6b, it is not possible to ascertain a falling tendency with larger becoming subproblems. Let us move on to the next sizes to investigate that further.

For the next sets of parameters we quadruple the number of items but want the qubit requirement to increase only by 10 first before making a larger step and adding 20. That is achieved by the following choices:

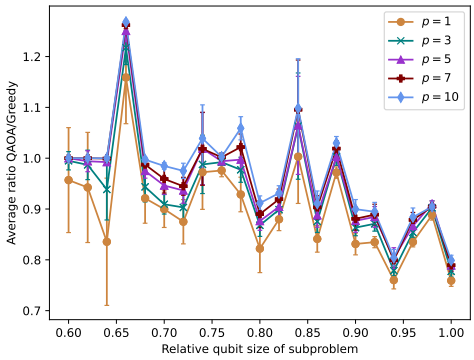
Number of items	Qubit requirement	Capacity ratio	Maximum profit/weight
20	30	0.1	10^3
20	50	0.1	10^9

Table 7.7.: Parameters for random KP instances of size 20.

Running the same simulation that led to Fig. 7.6 yields the results shown in Fig. 7.7 for the accordingly generated KP instances.



(a) Simulation results for 30 qubits, accompanied by a maximum profit/weight value of 10^3.



(b) Simulation results for 50 qubits, accompanied by a maximum profit/weight value of 10^9.

Figure 7.7.: Average ratios of QAOA by Greedy lower bound for random KP instances consisting of 20 items and featuring a capacity ratio of 0.1, generated using the parameters in Table 7.7. For each parameter combination the simulation is performed for 10 problem instances with the same specifications. For each generated instance, the HQCBB is run four times. The error bars depict the average of the resulting standard deviations for the equivalent KP instances.

Many things that could be observed for the instances of size 5 can also be recognized in the next larger regime. Maybe the most striking is the large degree to which the data series for the different depths do overlap. The course taken by the curve associated with $p = 1$ can be best identified individually; the corresponding values seem to be lower than the others - sometimes more, sometimes less. Especially in the peaks the other four data series match very well. Fig. 7.7a shows two larger peaks with top values slightly below 1.3 and 1.1 at about 57.5% and 77.5% qubit size, respectively. In Fig. 7.7b, in contrast, we can identify just one major peak at about 65% of the original qubit requirement with a maximum value above 1.2 as well as two similarly shaped smaller ones next to each other at larger relative residual qubit sizes. These peaks are proof that our QAOA was again able to beat the Greedy lower bound at certain, suggesting that the first small successes found for instances with 5 items in Fig. 7.6 were no coincidence. However, one has to admit that the QAOA returns better lower bounds only in selected situations - the Greedy value still represents the maximum in the majority of cases. Concerning the peaks, we observe more of rising and falling sections in Fig. 7.7 than in Fig. 7.6. Despite the larger peaks, Fig. 7.7a in particular justifies us to speak of a visible tendency of a decreasing QAOA output quality when the subproblem to which it is applied gets larger. This behavior can however only be weakly identified in Fig. 7.7b.

What has not been mentioned so far is the number of printed data points, which

directly translates to the number of differently sized subproblems for which our HQCBB calculated lower bounds during the simulation run. In fact, it is very unlikely for problem instances with the same number of items to induce a subproblem of a yet untouched size when running the HQCBB multiple times (which is done in our simulation to wipe out the slight probabilistic behavior in the node selection and, of course, due to different circuit depths being evaluated). This is why the data series normally all have values at the same relative residual qubit sizes. This is true both for Figs. 7.6 and 7.7, only the absolute number of data points has increased in the transition from 5 items to 20. To a good approximation, this number has quadrupled - just like the number of items, which is as expected since a four-times larger problem should in general induce an amount of (non-rejectable) subproblems with different sizes that is four-times higher.

The last thing I want to address before continuing with the next sizes is the error bars. In Figs. 7.6a and 7.6b the data points with value 1 are really the only ones that come with significant standard deviations. In contrast, large error bars can only be found in the left halves of Figs. 7.7a and 7.7b, but also not for every depth: The data points corresponding to $p = 1$ again form the outsider for being largely erroneous at small subproblems. However, we do not have analyzed enough data to draw any solid conclusions from these observations yet, so let us move on.

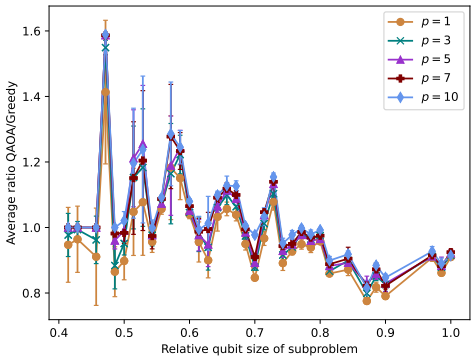
For the next parameter sets the factor by which the number of items is increased is not as large as in the last step, we are now doubling it:

Number of items	Qubit requirement	Capacity ratio	Maximum profit/weight
40	70	0.04	10^9
40	86	0.04	10^{14}

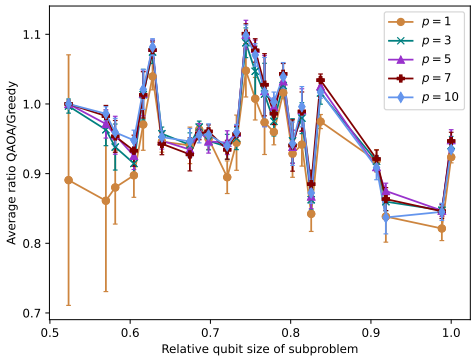
Table 7.8.: Parameters for random KP instances of size 40.

The analog simultation results for the random KP instances generated from these parameter specifications can be found in Fig. 7.8.

Almost every of the impressions we got from analyzing Figs. 7.6 and 7.7 can also be made when considering Fig. 7.8: the concordance between the data series for different depths with the data points corresponding to $p = 1$ deviating a bit downwards, an again increased number of peaks and data points and even comparably large error bars at the smaller subproblems, especially for the lowest depth. Only the tendency of poorer QAOA quality for growing subproblem qubit requirements can neither in Fig. 7.8a nor in Fig. 7.8b be recognized particularly well - not least because of the



(a) Simulation results for 70 qubits, accompanied by a maximum profit/weight value of 10^9.



(b) Simulation results for 86 qubits, accompanied by a maximum profit/weight value of 10^{14}.

Figure 7.8.: Average ratios of QAOA by Greedy lower bound for random KP instances consisting of 40 items and featuring a capacity ratio of 0.04, generated using the parameters in Table 7.8. For each parameter combination the simulation is performed for 10 problem instances with the same specifications. For each generated instance, the HQCBB is run four times. The error bars depict the average of the resulting standard deviations for the equivalent KP instances.

QAOA results obtained for the original problem sizes not forming the worst comparison values in either case this time.

What must definitely be highlighted here is the exceptionally good QAOA performance in Fig. 7.8a slightly below 50% of the original qubit size: The returned quantum lower bound exceeds its classical Greedy counterpart by almost 60%, thereby clearly representing the best result seen so far. Even the most shallow circuit depth $p = 1$ was sufficient in this case to reach a value of 1.4 or above. However, this success could apparently not be built on when increasing the maximum profit/weight value from 10^9 to 10^{14} - the top values achieved at two relative residual sizes in the right half of Fig. 7.8b, in contrast, are vertically located at a ratio of 1.1. In terms of the maximum values, The KP instances with the largest qubit cost thereby perform worse than those generated from Table 7.7 and approximately equal to those created from Table 7.6.

On the contrary, one thing is new in Fig. 7.8 compared to Figs. 7.6 and 7.7, namely that a significantly large portion of data points features a QAOA-Greedy ratio exceeding the threshold value of 1. In particular, for subproblems up to 80% of the original qubit requirement, Fig. 7.8a shows more than half the data points above the critical value (part of the full truth however also is that no improvement of the Greedy lower bound could be achieved for larger subproblems). This is not so pronounced in Fig. 7.8b, although there also sections in which the resulting ratio continuously stays above 1,

especially for the larger depths. Such observations are not valid for the smaller two regimes considered so far: For the generated KP instances consisting of 20 items, the cases of a better quantum bound are limited to the peaks in Figs. 7.7a and 7.8a, respectively, whereas an improvement on Greedy could only be made for the 5-items instances using a larger underlying depth at one certain subproblem qubit size each in Figs. 7.6a and 7.6b. Next to the maximum ratios achieved for a set of parameters, this growth in the share of data points for which the quantum lower bound outperformed its classical analogue can be considered confirmation for our idea of challenging the Greedy heuristic with our Grover-mixer QTG-induced QAOA likewise.

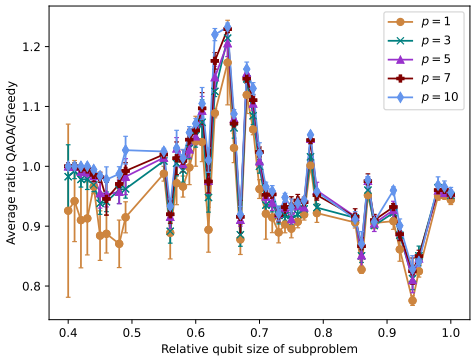
Let us figure out what of these two success measures can be found to hold at the fourth and largest regime to be analyzed here. The number of items is again increased by 20, corresponding to a minimum growth rate of 50% in this last step. Lastly we want to generate problem instances with the expressive and immense qubit costs of 100 and 120, respectively, for which the following parameter configurations are sufficient:

Number of items	Qubit requirement	Capacity ratio	Maximum profit/weight
60	100	0.025	10^{12}
60	120	0.025	10^{18}

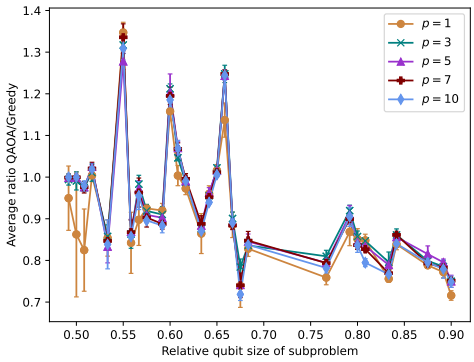
Table 7.9.: Parameters for random KP instances of size 60.

As we can see from Table 7.9, the high qubit requirements in this largest regime are mainly driven by the maximum profit/weight value next to the number of items. Since the capacity ratio has continuously decreased from Table 7.6 to Table 7.9, the upper limit for profits and weights naturally had to be up-scaled in order to reach the desired amounts of qubits. Our last simulation to investigate the relation between the two alternative lower bounds gave the results shown in Fig. 7.9.

A first thing catching the eye is the strongly increased density of data points in Fig. 7.9a. In some regions - especially when the QAOA-Greedy ratio is stagnating between 40% and 50% as well as between 70% and 80% of the original size in terms of qubits - they are compressed to a large degree whereas rising and falling sections manage to stretch them apart. On the other hand, the expected growth in the absolute number of data points due to more items failed to happen for our largest set of instances: This quantity seems rather unchanged in Fig. 7.9b compared to the 40-items regime in Fig. 7.8. Aggregated over all regimes, we notice that the amount of subproblems for which lower bounds shall be evaluated has - despite the partly strong visible growth, especially in the transition from Fig. 7.6 to Fig. 7.7 - not increased to the expected extent. Of course, randomly generated problem instances can and will by chance have more or less



(a) Simulation results for 100 qubits, accompanied by a maximum profit/weight value of 10^{12}.



(b) Simulation results for 120 qubits, accompanied by a maximum profit/weight value of 10^{18}.

Figure 7.9.: Average ratios of QAOA by Greedy lower bound for random KP instances consisting of 60 items and featuring a capacity ratio of 0.025, generated using the parameters in Table 7.9. For each parameter combination the simulation is performed for 10 problem instances with the same specifications. For each generated instance, the HQCBB is run four times. The error bars depict the average of the resulting standard deviations for the equivalent KP instances.

suited structures within the items, leading to more or less nodes that can be pruned off the search tree before coming to the lower bound calculation. Nevertheless, this can be best explained with the capacity ratio shrinking from regime to regime, generally implying a smaller relative number of feasible subproblems.

The usual findings - like a quite common course of curves for different depths with the data series for $p = 1$ deviating from that especially for smaller relative residual qubit sizes of subproblems and small standard deviations except for exactly these cases - can also be observed in Figs. 7.9a and 7.9b. With pretty much certainty we can state that these things seem to hold in general, irrespective of the chosen number of items, the specific capacity ratio and the particular limiting value for profits and weights. In Section 7.2.2 we found clear evidence for the correlation between a better QAOA output quality (there measured by the approximation ratio) and a larger circuit depth. Slight differences can also be recognized in Figs. 7.6 to 7.9, especially between $p = 1$ and the rest and probably best visible in Fig. 7.6b. The data series corresponding to largest depth of $p = 10$ also form an upper border for the others most of the time, particularly in Figs. 7.7a, 7.7b and 7.8a, while we have also seen exceptions to this rule, e.g. for the larger subproblems in Fig. 7.9b. Nevertheless, we have to conclude that further increases in the circuit depth after replacing $p = 1$ by $p = 3$ could not be found to achieve a significant improvement in how the QAOA performs in terms of bounding subproblems compared to the Greedy heuristic.

Compared to next smaller regime, Figs. 7.9a and 7.9b are again dominated by comparably large peaks. Each consists of three major peaks which are located closely together with maximum values between 1.1 and 1.2 and a bit more separated with top values between 1.2 and 1.4, respectively. Based solely on the largest lower bounds ratios achieved, our QAOA was not able to keep the level of its quality in relation to the Greedy lower bound that was found in Fig. 7.8a. Also, about neither of Fig. 7.9a nor Fig. 7.9b we can say that the large share of subproblem qubit sizes for which the quantum lower bound has beaten its classical counterpart that we have detected in the whole 40-items regime could be conserved: The critical value of 1 is just exceeded by the peaks and could not be reached anymore for subproblems larger than 80% and 65% of the original qubit cost, respectively. Hence, we need to infer that both - the maximum improvement reached by our QAOA and the portion of non-rejectable nodes in the search tree for which any improvement can be achieved - badly depends on the specific KP instance on which the HQCBB is applied. However, it is important to keep in mind that this Grover-mixer QTG-induced QAOA was capable of beating our classical lower bound in at least one case for each set of underlying parameters considered in this analysis - sometimes these cases even formed the majority.

7.3.2. Number of Explored Nodes

In Chapter 3, I went into the ultimate and superordinate motivation of our HQCBB setup: From extending the classical Branch and Bound for the Knapsack Problem with our QAOA we promised ourselves that the additional information may - in the ideal case - lead to a measurable improvement of the pure classical algorithm. Collecting and evaluating data for random benchmark instances to in detail investigate how the lower bound provided by the QTG-induced QAOA performs in comparison to the standard Greedy lower bound in Section 7.3.1 gave a close lookup into whether our QAOA represents an actual competitor for the Greedy heuristic. However, this analysis alone is like listing a set of arguments without being keen to draw a conclusion, as it tells nothing about whether the partly achieved success of the quantum bound over its classical counterpart is also sufficient to make a visible difference in the overall HQCBB run at the end of the day. This is a question we shall direct our attention to now.

In order to compare the HQCBB performance we first need a suiting measure. There are two natural choices offering themselves in general here: the computing time and the number of explored tree nodes required to find the optimal solution. The comparably high computational effort of our Grover-mixer QAOA induced by QTG makes that an easy choice for us. Compared to the classical B&B, the essence of our HQCBB is to execute the QAOA on top whenever a lower bound is to be calculated. This means that the only chance of the hybrid being faster than its counterpart is that less lower bounds

need to be evaluated. Even if non-optimal regions of the search space may be rejected earlier thanks to better lower bounds, we suspect the comparably high computational effort required to simulate our QTG-induced Grover-mixer QAOA on a classical machine to hinder us from extracting an actual run time improvement - especially in the double-digit range of items we are exploring. If at all, we expect to detect an improvement on the less-influenced level of explored nodes where the differences in computational expense between classical and quantum parts of the algorithm do not come into play. In line with Section 2.1 terminology, a node is called *explored* in our simulation once it is touched by the searching procedure, meaning that it ultimately gets removed from the stack (either due to infeasibility or because a backtracking has occurred or a new node is selected after a branching, see Algorithm 8).

Again, we will use our random generator in order to create KP instances of the greatest representativity. The maximum value for profits and weights affects the qubit requirements of the subproblems to which our QAOA is applied, as the latter depends logarithmically on the capacity, which itself is set according to the sum of all weights. However, this should in principle have no effect on the exploration of the search tree, as both profits/weights and the capacity are up-scaled by approximately the same factor when increasing their upper limit (assuming a fixed capacity ratio). In contrast, independent of the actual values, the capacity ratio - determining how large the capacity is compared to the sum of all weights - stands in direct correlation with the share of feasible solutions among all possible 2^N solutions (for a problem of size N). Since a larger number of feasible solutions generally leads to a higher the amount of tree nodes that cannot be pruned off, the capacity ratio, on the contrary, has an impact on the number of explored nodes. The parameters in this last simulation part then are:

Number of items	Capacity ratio	Maximum profit/weight
5	0.25	10^3
10	0.19	10^4
15	0.135	10^5
20	0.08	10^6
25	0.07	10^7
30	0.06	10^9
35	0.05	10^{10}
40	0.04	10^{11}
45	0.03625	10^{13}
50	0.0325	10^{15}
55	0.02875	10^{16}
60	0.025	10^{18}

Table 7.10.: Parameter for random KP instances.

While the number of items is iteratively increased in steps of 5 items, the exponent characterizing the maximum profit/weight value grows by one or two. The capacity ratio, on the other hand, simultaneously diminishes subject to three different slopes, aligned to the regimes we introduced in Section 7.3.1: In the small-size regime (5 to 20 items) it decreases by a value of 0.05 to 0.06 in every step, the medium-size regime (20 to 40 items) features a clear loss of 0.01 in each iteration and the large-size regime (40 to 60 items) comes along with with very small reductions of 0.00375 each. This time, the corresponding qubit requirements are not provided in Table 7.10 - simply because they are not relevant for our analysis of the HQCBB performance.

As in Section 7.3.1, 10 equivalent KP instances will be generated for each combination of parameters in Table 7.10 to promote generality. Completely analogous, the algorithm run shall be repeated four times for every created problem instance to take care of the probabilistic artifacts that have crept in our HQCBB via the node selection procedure (cf. Algorithm 6). Next to investigating the influence of different circuit depths, we here also want to compare the behavior of our classical framework where no QAOA is executed at all (lower bound then always given by Greedy). Storing the number of nodes necessary in order to arrive at the optimal solution yields the following results:

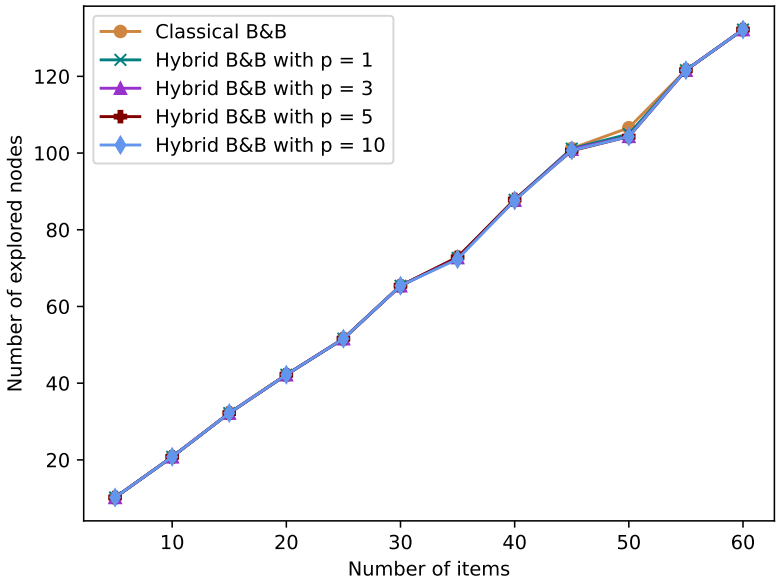


Figure 7.10.: Average number of explored nodes for random KP instances generated using the parameters in Table 7.10. For each parameter combination the simulation is performed for 10 problem instances with the same specifications. For each generated instance, the HQCBB is carried out four times. The error bars depicting the average of the resulting standard deviations for the equivalent KP instances are not visible for being too small.

For the capacity ratio and the maximum profit/weight value evolving as chosen in Table 7.10 and described afterwards, the number of explored needs seems to scale proportionally with the number of items. Fig. 7.10 shows a pretty accurate linear dependency between these two quantities with a slope that seems to be a bit larger than two. Without any further input we would rather expect an exponentially growth in the number of explored nodes, since that holds for the number of possible solutions. However, the continuously decreased capacity ratio in Table 7.10 can explain why its increase is reduced to a linear scaling.

Striking about Fig. 7.10 is that it apparently lacks the existence of any error bars and, in particular, that we can see only one curve of data points in the first place. The former can only be reasoned by the standard deviations from the average numbers of explored nodes being so small that they are not visible in our graphic. Concerning the latter, let us elaborate on that in two parts, namely the different QAOA circuit depths among each other and the comparison with the pure classical B&B configured by the strategies described in Chapter 4. An exact overlapping between the data series associated with the different depths is the confirmation that choosing a larger depth was indeed not successful in terms of the aim to improve the performance of the HQCBB, at least for the random sets of instances considered here. Even worse, a further overlap with the classical data series automatically tells us that none of the depths was able to induce a measurable effect. However, taking a closer look at Fig. 7.10 reveals a small splitting of the curve's course lasting from 45 items to 55 items. More precisely, the data point corresponding to the classical runs at 50 items is shifted slightly to the top compared to those associated with the depths $p \in \{1, 3, 5, 10\}$. Although there are no further splittings for the different depths, this represents a success, as our HQCBB was in this case indeed able to outperform its pure classical analogue in terms of the number of explored nodes. In Section 7.3.1 we saw that outstanding improvements on the Greedy lower bound in a magnitude of about 60% were only very rarely attained by the QAOA. The vast majority of the found improvements instead showed a plus smaller than 30%. Therefore, the prospects for achieving more significant reductions on a global scale like for the number of explored nodes were not very promising, meaning that the minor success in Fig. 7.10 could not be reasonably formulated as a clear expectation.

Back to the (Classical) Roots

Let me close the investigation of possible performance optimizations in terms of the number of explored nodes via our hybrid ansatz with a general heads-up. As emphasized above in the discussion to figure out the measure to apply, simulating the QAOA circuit seen in Fig. 6.12 on a classical device is computationally expensive, especially with growing depth. In line with our motivation in Section 7.1, emulating the circuit

execution with our high-level simulator may have enabled to tackle problem instances in the preceding analyses that would otherwise have been intractable. However, this does, of course, not mean that the problem sizes which our QAOA is capable of handling in a reasonable amount of time can compete with the sizes that a pure classical Branch and Bound built according to Chapter 4 is able to solve. To illustrate that, let us perform the same simulation examining the amount of nodes that is necessary to find the optimal solution for the classical B&B part of the HQCBB where the quantum extension is removed again. For the following parameters this classical simplification had no issues in solving the accordingly generated KP instances to optimality:

Number of items	Capacity ratio	Maximum profit/weight
500	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
1000	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
1500	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
2000	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
2500	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
3000	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
3500	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
4000	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
4500	{0.1, 0.2, 0.3, 0.4, 0.5}	1000
5000	{0.1, 0.2, 0.3, 0.4, 0.5}	1000

Table 7.11.: Parameters to generate random KP instances for simulating the number of explored nodes in the classical B&B.

For simplicity, the upper limit for profits and weights is chosen equally in any considered problem instance with a value of 1000. Since there is no circuit depth that can be varied in the classical setting, we create five sets of instances with different capacity ratios for any size. As described above, the share of the capacity by the sum of all weights has a major impact on the number of feasible solutions and - thereby - on the number of non-rejectable tree nodes that need a lower bound assigned. Compared to Table 7.10, Table 7.11 suggests that the instances handled in the classical setting are about a 100-times larger than those considered before where the QAOA was still in place. Decreasing the calls to compensate probabilistic misleadings, the generation of five equivalent KP instances for every combination of parameters as well as carrying out the classical B&B twice for each oft them results in Fig. 7.11.

On average, every data series in Fig. 7.11 shows - aggregated over the full range - the expected behavior for the number of explored nodes, namely that it grows with the amount of items. However, we apparently did not find the same almost perfect linear dependency that we saw in Fig. 7.10. Instead, the data points seem to approach an upper limit when the number of items per KP instance is increased, ignoring the

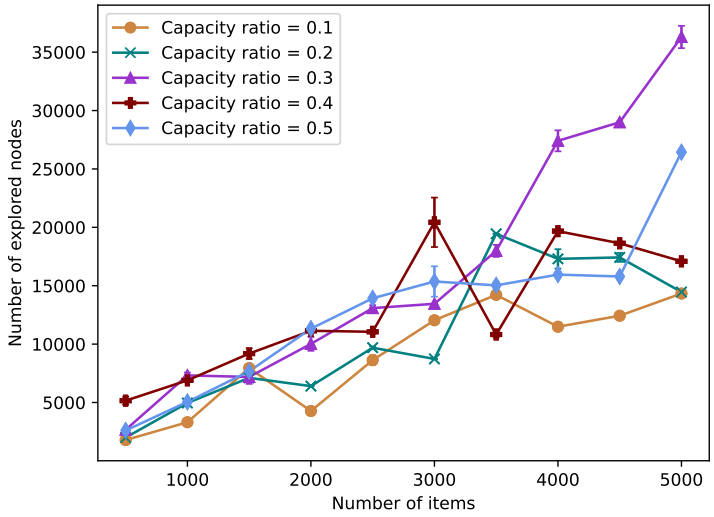


Figure 7.11.: Average number of explored nodes for random KP instances generated using the parameters in Table 7.11. For each parameter combination the simulation is performed for 5 problem instances with the same specifications. for each generated instance, the classical B&B is carried out twice.

strong rise of the data series corresponding to a capacity ratio of 0.3 in the right half of Fig. 7.11 and the jump in the last step at the largest ratio between capacity and total sum of weights of 50% for the moment. The sudden rise at 3000 items for a capacity ratio of 0.4 as well forms a moderate exception to that observation. In comparison to Fig. 7.10 where not a single averaged standard deviation was large enough to be visible, there are is a handful of data points featuring non-vanishing error bars here, most notably at 3000 items and a capacity ratio of 0.4 again. However, an in depth analysis and explanation of the behavior of the different data series in Fig. 7.11 is not of significance to us, it should just be demonstrated that the classical Branch and Bound in our HQCBB is capable of performing the same simulation of global algorithm properties for much larger problem instances.

Conclusions & Outlook

We have come to the end of the thesis, meaning that it is about time to draw an overall conclusion on what has been done in the past chapters and give an outlook on further research directs. This will provide the possibility to make a final condensed statement about the meaning and the value of the present work.

In retrospect, the raison d'être of this thesis is three-fold: First and foremost, the fundamental motivation was to utilize a quantum algorithm to enhance classical Branch and Bound techniques, here at the example of the Knapsack Problem. The B&B algorithm (which can actually better be understood as an algorithmic framework as we have seen) one of the most commonly used classical algorithms to solve combinatorial optimization problems. It owes its popularity to its generic and highly customizable formulation, making it applicable to pratically any ILP. The specific choice of a quantum algorithm is then connected to the second point of interest: The Quantum Tree Generation developed by Wilkening et al. [Wil+23] as part of their quantum algorithm for the Knapsack Problem creates a superposition of all feasible states for a given KP instance. Thereby, it brings itself into the pole position for being extracted and afterwards recycled in the shape of a state preparation in a Quantum Alternating Operator Ansatz following the Grover-mixer approach [BE20] that shifts the (usually very large) effort needed to construct a suiting mixer to designing a procedure for preparing the highly non-trivial initial state. For a given problem instance consisting of N items and a capacity of $W_{\max}$, we need $N + \lfloor \log W_{\max} \rfloor + 1$ qubits to implement the resulting QAOA, as was explained in Chapter 5. The concept of extending the Branch and Bound with this QAOA by introducing it as an alternative (quantum) lower bound was based on the consideration that adding information to the algorithm will only increase the lower bound quality, hopefully inducing an improved performance of the algorithm by helping to avoid the exploration of non-optimal search space regions. In Sections 6.1 and 6.2 we derived that our QAOA features a complexity $\mathcal{O}(N \log(W_{\max})^2)$ of more or less elementary gates of which the most are one or two-qubit unitaries. The third component which I meant above then came into play when actually simulating the final QAOA circuit we arrived at with Fig. 6.12: Inspired by Wilkening et al. [Wil+23], a high-level simulator was developed that allowed us to investigate KP instances featuring qubit requirements of up to 120 in Chapter 7. This simulator extended the classical emulation of the QTG by simple analytical formulae for the

action of the mixing and the phase separation unitaries; as discussed in Section 7.1, it was capable of outperforming its conventional counterpart of actually simulating the qubits on the classical machine thanks to the reduction in dimension, as the capacity register could be completely eliminated. On the considered problem instances, which were randomly generated based on certain parameters to guarantee representativity, we could indeed observe cases in Section 7.3.1 where our QTG-induced Grover-mixer QAOA managed to beat the Greedy lower bound - and not too few of them. Even further, there was no set of KP instances among the different parameter configurations for which we could not find any improvement on the Greedy heuristic throughout the runs of the HQCBB; for certain subproblems we even achieved a QAOA-surplus of 60%. This is an excellent result taking into account the effectiveness of the Greedy lower bound. What could, on the contrary, not be verified is a significant impact of the circuit depth. The mainly small relative exceedings of the Greedy threshold seen in Figs. 7.6 to 7.9 did not really give us reason to expect performance improvements of the whole HQCBB to be detectable on global scale. The corresponding simulation results in Fig. 7.10 are in line with this suspicion. Nevertheless, we found one set of KP instances - each composed of 50 items - where the number of explored nodes is on average slightly smaller when adding the quantum lower bound as an alternative to the classical one. This is a minor success, but represents a true enhancement.

Fig. 7.11 impressively illustrates that we must not succumb to the illusion of already being in a state where the approach pursued in this thesis can lead to a general improvement even on the large scale, especially not with the local resources available here. The magnitude of the number of items forming the simulation basis for Fig. 7.11 implies that emulating the designed QAOA circuit on a classical device can by no chance compete with the Greedy routine in terms of computing time. However, this was clear from the offset and has never been anywhere near a declared goal of this thesis. The essential learning here is that it took very much effort to build a QAOA that is capable of tackling medium-size problem instances and beating a well-established classical method which could hardly be simpler. Based on these drawbacks, let me state the real heritage of my work: Instead of just handling the two extreme cases where the quantum extension is either present and integrated as desired or removed completely (which means that we are back in the pure classical setting), we now have the ability to merge these two configurations by only applying the QAOA if the currently explored (non-prunable) subproblem is small enough and calculating the Greedy lower bound solely otherwise. Thereby we do not have to restrict to medium-size problems, meaning that there is no instance solvable by the classical B&B configured according to Chapter 4 that cannot be tackled by the HQCBB in a reasonable amount of time, while the advantage of possible performance improvements (in terms of explored nodes) is maintained. The criteria determining whether a subproblem is small enough for the QAOA application can then be specified according to the available hardware and resources; usually, the qubit cost will be the limiting factor with which that rule must

be aligned. Based on the experiences we made in Chapter 7, the capacity ratio as the critical quantity would be the better choice when having our high-level simulator at hand. This final step combines the best of both worlds; the underlying concept can, as emphasized in the introduction, be considered one of the most promising paths for quantum computing to become relevant and make a difference on the industry scale. Nevertheless, I decided to conduct the simulations of the lower bounds and the numbers of explored nodes in Section 7.3 not on this unified level, but in the stage where the QAOA is either applied or not, to keep the visualization and the analysis of the results straight-forward (for the lower bound comparison this ultimate refinement is not even relevant). With the idea of making large problems tractable for state-of-the-art quantum algorithms despite the current omnipresent hardware constraints by reducing the qubit costs we are, of course, not alone: Ponce et al. [Pon+23] for example demonstrated recently how large instances of the so-called Max Cut Problem[1] - as an exemplary QUBO (quadratic unconstrained binary optimization) problem - may be solved to a good approximation using a QAOA by employing a preprocessing that decomposes the input graph into smaller graphs with smaller qubit requirements.

The last anecdote brings us to the following question: What about other problems than the Knapsack Problem? By design, the concept of enhancing Branch and Bound techniques by quantum approximate optimization is not specifically tailored to the Knapsack Problem. It may even be desirable to consider combinatorial optimization problems with a more complex structure - as emphasized in Section 1.3, the Knapsack Problem is a standard example for which the existing classical algorithms are so advanced that they can solve instances consisting of up to 100,000 items within a few seconds [MT90, S.2.10].[2] Instead of challenging the whole problem, one may also take a step back and extend the simulations and the analyses of their results to different structures among the items: While our generator exclusively picked values for profits and weights according to an even probability distribution when randomly creating KP instances in Chapter 7, this is by far not the only possibility. More specifically, there are three common correlations between the profits and the weights studied across the literature: In the simplest case where both profits and weights are chosen as random integers not exceeding a specified maximum value (as in our case) the items are also called *uncorrelated*; when each weight is picked uniformly random from this interval, but the profit can then only be chosen randomly in a certain distance from the corresponding weight, one says the items are *weakly correlated*; we talk about *strongly correlated* items in case that the randomly drawn weight also determines the

[1] *Max Cut* is an NP-hard problem whose task is, given an input graph consisting of vertices and edges, to find that partition of the vertices in two disjoint sets where the number of edges connecting differently assigned vertices occupies its maximum value.

[2] Recall from Definition 1.14 that its objective function is linear in the input binary variables and only one constraint is needed to formulate it.

associated profit which is just shifted by a positive constant. How our simulation results change when generating instances with weakly or even strongly correlated items instead of uncorrelated ones could be investigated further in a follow-up. Increasing the correlation generally corresponds to a shrinking difference between the largest and the smallest effective profit[3], which in turn can be expected to lead to a more complex problem [MT90]. Despite these artificially increased levels of difficulty, the state-of-the-art classical algorithm with name COMBO could be shown to be capable of solving each benchmark instance from the three correlation types with up to 10,000 items in less than 0.2 seconds [MPT99]. It is not without reason that Pisinger [Pis05] consequently pursues the question "Where are the the hard knapsack problems?". In terms of other combinatorial optimization problems, one could e.g. move in a more graph-theoretical direction and set up a classical Branch and Bound for the Graph Coloring Problem[4]. Since Graph Coloring is a minimization problem - in contrast to the Knapsack Problem - the output of a QAOA can, by an analog reasoning as in Section 2.2.2 with flipped sign, directly be used as an upper bound in the B&B algorithm. The rich variety of interesting graph theoretical quantities even allows to also introduce a quantum lower bound: It is easy to verify that the outcome of the Max Clique Problem[5] bounds the chromatic number of the graph at hand from below (there are at least as many colors needed for a graph as there are vertices in its maximum clique), meaning that a QAOA for Max Clique would automatically provide a quantum lower bound for the Graph Coloring B&B. A further refinement could be made by additionally considering the Max Independent Set Problem[6], which would serve as an alternative to applying the Max Clique QAOA, based on whether the complement graph can be formulated with less variables, implying a fewer number of qubits to simulate. Lucas [Luc14] shows for each of these three graph theoretical problems - among a whole bunch of NP-complete and NP-hard problems - how their constraints may be wrapped inside the objective function and how to derive an Ising formulation from that by using spins instead of variables, which can afterwards be transformed to

[3] The effective profit is the ratio of an items profit and its weight; we introduced it when sorting the items as a preparation for calculating the Greedy bounds in Section 4.2.

[4] *Graph Coloring* is an NP-hard optimization problem that asks, given a graph, to find the smallest number of different colors required to dye every vertex such that no two adjacent vertices (i.e. vertices that are connected by an edge) have the same color assigned. The minimum value is called the *chromatic number*.

[5] *Max Clique* is another NP-hard optimization problem which, as the name suggests, aims at finding the largest clique in a graph. A *clique* in turn denotes a complete subgraph, i.e. a subgraph in which any vertex is adjacent to any other.

[6] Like the other two, *Max Independent Set* is provably NP-hard; given a graph, the object of desire here is the largest set of independent vertices or, more precisely, its size. A vertex subset is called *independent* if no two of the contained vertices are adjacent. It is not difficult to convince yourself that Max Clique transforms to Max Independent Set in the transition from a graph to its complement graph where a pair of vertices is connected by an edge if that was not the case in the original graph and vice versa.

its quantum version by replacing the classical spins with the associated Pauli matrices. This process is an example of softcoding constraints (cf. Section 2.2.2) and leads to phase separation operators in the sense of the Quantum Ising model [Dzi05], which in combination with the standard mixer from Eq. (2.2.4) make Softconstraint QAOAs. For Graph Coloring and Max Independent Set Hadfield et al. [Had+19, Ss.4.2&4.4] discusses design criteria for suitable mixers in line with Section 2.2.3 that enable to treat them as hardconstrained problems and guarantee the preservation of feasibility throughout the respective quasi-adiabatic evolution. Max Clique is there handled via its relation to Max Independent Set [Had+19, A.2.2].[7] However, it cannot be guaranteed that a comparable high-level simulator can also be set up for these three graph theoretical problems. The hope to be able to extend the simulation concept developed by Wilkening et al. [Wil+23] for the QTG in a similar straight-forward fashion to the residual Grover-mixer QAOA logic (which we could confirm) as well as the prospect to investigate a yet unevaluated mixer formed the motivation to consider the Knapsack Problem of all things. Although COMBO performs that well on every of the analyzed correlation types [MPT99], the findings of Pisinger [Pis05] indicate that there is still legitimate research interest in the Knapsack Problem. On the quantum side, a huge step has just been made by Wilkening et al. [Wil+23], who where able to be the first to test a quantum algorithm on realistic benchmark instances[8]. Their results suggest an advantage of the quantum algorithm over COMBO in terms of computing time starting at instances with 600 variables. The memory savings however are even more drastic: While the number of required logical qubits in their algorithm scales proportionally with the number of items, 10^{10} involved bits is the magnitude in which COMBO operates. The full consequences of the these results remain to be seen. For a good reason we were not able to simulate such large KP instances in Chapter 7: An important ingredient of the QTG in [Wil+23] is to not only discard infeasible solutions but to also iteratively sort out all (possibly feasible) solutions whose associated profit does not exceed a threshold determined by COMBO. This additional selection procedure reduces the final number of states to keep to an extent that allowed them to consider up to 1600 variables. Introducing such a threshold in our QTG is prevented by the structure of the employed Grover mixer, given by Eq. (5.2.1), that is based on the superposition of all feasible states and not only the ones that pass some quality check. The threshold in [Wil+23] furthermore made the use of biased Hadamard gates possible in the QTG implementation, pushing an item register qubit from state $|0\rangle$ into superposition $|0\rangle + |1\rangle$ with a lower or larger probability depending on whether the corresponding variable has value 0 or 1 in the COMBO solution, respectively. In a further refinement of our implementation (cf. Section 6.1) one could try to utilize

[7] Note that proposing a mixer for Max Independent Set that does not require a softcoding was even already part of the original QAOA work by Farhi, Goldstone, and Gutmann [FGG14].

[8] A true quantum advantage can obviously never be possible as long as quantum algorithms are not even capable of simulating those benchmark instances that are used to assess the performances of aspiring classical algorithms. This would typically require about 100 to 10,000 qubits.

the Greedy lower bound obtained at a subproblem immediately before the QAOA application in Algorithm 10 as a bias for the Hadamard gates.

What has been completely disregarded so far is the disadvantages that come along with QAOA as a quantum algorithm in general. In a first step, we moved from a Softconstraint Quantum Approximate Optimization Algorithm to a more general Quantum Alternating Operator Ansatz due to the inability of the original QAOA to properly handle constraints. However, there are still inherent things following from the way how the QAOA is defined that might make it not the ideal choice. A prominent example are the so-called barren plateaus - regions where the gradient of the objective function vanishes exponentially in the number of qubits - which naturally impose an impediment on the success for a derivative-based optimization routine. The occurrence of barren plateaus is however not QAOA-specific, but can be rather understood as a phenomenon that is related to parameterized quantum circuits in general, especially in the deep case where many layers are run by variational quantum algorithms [McC+18]. As emphasized in the introduction, VQAs provide our best chance to achieve a quantum advantage in the medium term on NISQ devices for leveraging classical computation power in the outsourced the parameter optimization and thereby keeping the qubit requirement as low as possible [Cer+21]. Hence, it is preferable to try overcoming - or at least mitigating - the obstacle of barren plateaus in VQAs instead of fully changing the course. Binkowski et al. [Bin+23] recently proposed a strategy in this regard: By extending a given VQA with a routine of non-unitary operations, implemented via the LCU (linear combination of unitaries) method [CW12; Cha23], which is applied whenever the gradient falls below a certain threshold, one can trigger the classical optimization to jump out of the corresponding barren plateau. This additional logic comes at the cost of mid-circuit measurements and a rather small register of ancilla qubits. It is, in fact, promoted to just be the start of a larger framework called *quantum conic programming* (QCP). The promising results obtained for Max Cut suggest that extending our QAOA as demonstrated in [Bin+23] could make our quantum setup more robust against typical pitfalls like barren plateaus on the one hand and even increase the approximation quality of its solutions on the other.

Last but not least, we should question the classical components of our HQCBB in Algorithm 8. Although Branch and Bound as an algorithmic family may be configurable and therefore applicable to any ILP, this does not necessarily mean that it is also the best algorithm for every problem. Even in case of the Knapsack Problem, the best available classical solver - COMBO - is based on Dynamic Programming (DP) instead of Branch and Bound. As derived in Appendix A, DP provides a reliable pseudo-polynomial time bound $\mathcal{O}(NW_{\max})$ which cannot be stated in a similar fashion that easily for B&B as discussed in Section 2.1.1. Future directions of research could thus be to examine other well-performing classical algorithms - applied to KP or a different COP - in the regard of possible enhancements via VQAs. A final fair point

can be made by asking whether the Greedy heuristic is too simple to send it as the classical representative into the ring where the better lower bound is fought out. It is actually pretty simple to come up with a KP instance for which the plain method in Algorithm 4 underlying the Greedy lower bound fails badly in returning a satisfying approximation to the optimal solution: Let $N = 2$ and $W_{\max} = M$ with $M \in \mathbb{N}$; let the profits then be $\boldsymbol{p} = (2, M)$ and choose the weights as $\boldsymbol{w} = (1, M)$. In this case the procedure sorting the items according to their efficiency (cf. Section 4.2.1) does not change the order of the two elements. Hence, applying Algorithm 4 here yields a profit value of 2 while the optimal solution consists of packing the second item, featuring a value of M. Choosing M arbitrarily large shows that no non-vanishing performance guarantee can be associated with Algorithm 4. However, a one-liner trick can be shown push the performance guarantee to $\frac{1}{2}$: Instead of returning the profit collected throughout the iteration in Line 10, output that value or the largest profit depending on what is greater [KPP04, Thm.2.5.4]. Since a lower bound which in the worst-case reaches only half the optimal solution value is still not very impressive, this slightly extended Greedy heuristic can be pushed to an ε-approximation scheme ($\varepsilon \leq \frac{1}{2}$) with a performance guarantee of $1 - \varepsilon$ by first searching for the best value among all sets of less than $l = \min\{\lceil \frac{1}{\varepsilon} \rceil - 2, N\}$ selected items and afterwards proceeding with the solutions of Hamming weight[9] l which are filled up by the remaining items of smaller profits via the improved Greedy in case of a non-zero residual capacity [KPP04, Thm.2.6.2].[10] Depending, of course, on the actual number of items, it would be interesting to see whether our QAOA is also able to beat an improved Greedy heuristic once the capacity is (assuming a small ε) such that $N^{l(\varepsilon)-1}$ and $\log(W_{\max})^2$ are of the same magnitude, meaning that both quantum and classical lower bound would in principle share comparable computing times.

In the face of the open questions and various directions in which this work could be extended in future research that I have elaborated on above, let me conclude with picking up on an introductory statement by highlighting the intangible value inherent in this thesis: showcasing a proof of concept for how well-established and far-developed classical algorithms applied to NP-hard combinatorial optimization problems may be enhanced by the use of VQAs. The crucial feature that the extent to which they are employed can be adjusted to the available hardware resources makes this approach interesting for the industry scale even in the NISQ-dominated medium term.

[9] The *Hamming weight* denotes the number of entries in a bitstring with value 1 (or, more specifically, with values different from 0).

[10] Obviously, this enhancement of the Greedy lower bound comes with an increase in computing time to $\mathcal{O}(N^l)$ [KPP04, Ss.2.6&6.1]. Not to forget are also the even stronger growing memory requirements; this issue is e.g. addressed by Kellerer and Pferschy [KP99].

Appendix

A. Dynamic Programming

The general solving technique called *Dynamic Programming (DP)* was developed by Bellman [Bel72] in a time where classical computers were approximately what quantum computers are nowadays. DP is frequently used in the context of the Knapsack Problem (cf. Section 1.3) and represents the strongest contender to Branch and Bound (cf. Section 2.1). In principle, it is applicable to any kind of optimization problem with an optimal substructure and overlapping subproblems, meaning that the optimal solution to the full problem can be obtained by combining the optimal solutions to the subproblems generated for it and that an algorithm solving the full problem also solves the same subproblems again and again. In a nutshell, DP tries to find a recursion relation for the specific problem at hand and store intermediate results in order to not start from scratch again in every iteration step as it is the case in a brute-force search approach. Applied to the Knapsack Problem, let $\hat{P}(w_{\max}, n)$ for $w_{\max} \in \{0, ..., W_{\max}\}, n \in \{0, ..., N\}$ denote the highest total profit that can be achieved using items $1, ..., n$ with the cost being limited to at most $w_{\max}$ for a given KP instance consisting of N items and a capacity $W_{\max}$ in the sense of Definition 1.14, i.e.

$$\hat{P}(w_{\max}, n) = \max \sum_{j=1}^{n} p_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \leq w_{\max} \quad \text{for } x_j \in \{0, 1\}.$$

The Knapsack Problem in this notation thus asks for finding $\hat{P}(W_{\max}, N)$. By explicitly writing out a table with rows indicated by $n = 0, ..., N$ and columns labeled by $w_{\max} = 0, ..., W_{\max}$ for a small KP instance with first increasing columns than rows you can comprehend that $\hat{P}(w_{\max}, n)$ is given by

$$\hat{P}(w_{\max}, n) = \begin{cases} \hat{P}(w_{\max}, n-1) & , \text{ if } \quad w_n > w_{\max} \\ \max\left\{\hat{P}(w_{\max}, n-1),\ \hat{P}(w_{\max} - w_n, n-1) + p_n\right\} & , \text{ otherwise} \end{cases} \tag{A.1}$$

with initial values $\hat{P}(w_{\max}, 0) = 0 \ \ \forall\, w_{\max} \in \{0, ..., W_{\max}\}$ as it is obviously not possible to afford any profit if no item is allowed to be included in the knapsack. This is called the *Bellman recursion* for KP. The first case in Eq. (A.1) in the context means that item n is too expensive for the current cost bound $w_{\max}$. The second case covers the other two possibilities, namely that item n is affordable but does not improve the best value for $\hat{P}$ found in the previous iteration and that it is affordable and indeed yields an improvement; the latter implies both that the permitted cost $w_{\max}$ needs to be reduced by the weight of item n and that the corresponding value for $\hat{P}$ can be increased by its profit. We can then obtain $\hat{P}(W_{\max}, N)$ via the iterative procedure in Algorithm 11.

One can show via induction that Algorithm 11 indeed produces the optimal solution value to the given KP instance. Furthermore, Algorithm 11 implies that the Knapsack

Algorithm 11: Dynamic Programming (KP)

for $w_{\max} \in \{0, ..., W_{max}\}$ **do**
  Set $\hat{P}(w_{\max}, 0) = 0$
for $n \in \{1, ..., N\}$ **do**
  for $w_{\max} \in \{0, ..., w_n - 1\}$ **do**
    Set $\hat{P}(w_{\max}, n) = \hat{P}(w_{\max}, n-1)$
  for $w_{\max} \in \{w_n, ..., W_{max}\}$ **do**
    if $\hat{P}(w_{\max} - w_n, n-1) + p_n > \hat{P}(w_{\max}, n-1)$ **then**
      Set $\hat{P}(w_{\max}, n) = \hat{P}(w_{\max} - w_n, n-1) + p_n$
    else
      Set $\hat{P}(w_{\max}, n) = \hat{P}(w_{\max}, n-1)$
return $\hat{P}(W_{\max}, N)$

Problem can be solved to optimality within $\mathcal{O}(NW_{\max})$ complexity. Therefore, KP is said to be solvable in *pseudo-polynomial time*, as the required computing time is a polynomial in the numeric value of the inputs but not necessarily in the length of the inputs (the number of bits needed to represent them) - the capacity $W_{\max}$ may actually grow exponentially in the problem size, i.e. the number of items. Due to the existence of such an algorithm, the decision version of the Knapsack Problem is called *weakly NP-complete*.

B. Nelder-Mead Method

Algorithm 12: Nelder-Mead

Initialize $2p+1$ test points $\boldsymbol{\chi}_1, ..., \boldsymbol{\chi}_{2p+1}$
while *termination condition not satisfied* **do**
  Find a permutation π of $\{1, ..., 2p+1\}$ such that the test points are sorted in ascending order as $\overline{E}_p(\boldsymbol{\chi}_{\pi(1)}) \leq \overline{E}_p(\boldsymbol{\chi}_{\pi(2}) \leq ... \leq \overline{E}_p(\boldsymbol{\chi}_{\pi(2p+1)})$
  Calculate the geometric center $\boldsymbol{\chi}_0$ of the point set $\{\boldsymbol{\chi}_{\pi(1)}, ..., \boldsymbol{\chi}_{\pi(2p)}\}$
  Calculate the reflected point $\boldsymbol{\chi}_r = \boldsymbol{\chi}_0 + a(\boldsymbol{\chi}_0 - \boldsymbol{\chi}_{\pi(2p+1)})$ with $a > 0$
  if $\overline{E}_p(\boldsymbol{\chi}_{\pi(1)}) \leq \overline{E}_p(\boldsymbol{\chi}_r) < \overline{E}_p(\boldsymbol{\chi}_{\pi(2p)})$ **then**
    $\boldsymbol{\chi}_{\pi(2p+1)} \leftarrow \boldsymbol{\chi}_r$
    continue
  else if $\overline{E}_p(\boldsymbol{\chi}_r) < \overline{E}_p(\boldsymbol{\chi}_1)$, *i.e.* $\boldsymbol{\chi}_r$ *is the best point so far* **then**
    Calculate the expanded point $\boldsymbol{\chi}_e = \boldsymbol{\chi}_0 + b(\boldsymbol{\chi}_r - \boldsymbol{\chi}_0)$ with $b > 1$
    if $\overline{E}_p(\boldsymbol{\chi}_r) \leq \overline{E}_p(\boldsymbol{\chi}_e)$, *i.e. not even* $\boldsymbol{\chi}_e$ *is better than* $\boldsymbol{\chi}_r$ **then**
      $\boldsymbol{\chi}_{\pi(2p+1)} \leftarrow \boldsymbol{\chi}_r$
    else
      $\boldsymbol{\chi}_{\pi(2p+1)} \leftarrow \boldsymbol{\chi}_e$
    continue
  else
    if $\overline{E}_p(\boldsymbol{\chi}_r) < \overline{E}_p(\boldsymbol{\chi}_{\pi(2p+1)})$ **then**
      Calculate the contracted point $\boldsymbol{\chi}_c^{(o)} = \boldsymbol{\chi}_0 + c(\boldsymbol{\chi}_r - \boldsymbol{\chi}_0)$ on the outside with $0 < c \leq 1/2$
      if $\overline{E}_p(\boldsymbol{\chi}_c^{(o)}) < \overline{E}_p(\boldsymbol{\chi}_r)$ **then**
        $\boldsymbol{\chi}_{\pi(2p+1)} \leftarrow \boldsymbol{\chi}_c^{(o)}$
        continue
    else
      Calculate the contracted point $\boldsymbol{\chi}_c^{(i)} = \boldsymbol{\chi}_0 + c(\boldsymbol{\chi}_{\pi(2p+1)} - \boldsymbol{\chi}_0)$ on the inside with $0 < c \leq 1/2$
      if $\overline{E}_p(\boldsymbol{\chi}_c^{(i)}) < \overline{E}_p(\boldsymbol{\chi}_c^{(i)})$ **then**
        $\boldsymbol{\chi}_{\pi(2p+1)} \leftarrow \boldsymbol{\chi}_c^{(i)}$
        continue
  $\boldsymbol{\chi}_{\pi(j)} \leftarrow \boldsymbol{\chi}_{\pi(1)} + d(\boldsymbol{\chi}_{\pi(j)} - \boldsymbol{\chi}_{\pi(1)}) \;\; \forall\, j \in \{2, ..., 2p+1\}$ with $0 < d \leq 1/2$
  $\boldsymbol{\chi}_j \leftarrow \boldsymbol{\chi}_{\pi(j)} \;\; \forall\, j \in \{1, ..., 2p+1\}$ (reversal of the permutation)
return $\underset{j \in \{1,...,2p+1\}}{\operatorname{argmin}} \; \overline{E}_p(\boldsymbol{\chi}_j)$

Algorithm 12 depicts the general Nelder-Mead method where we compactly write $\boldsymbol{\chi} = (\boldsymbol{\beta}, \boldsymbol{\gamma}) \in [0, \pi)^p \times [0, 2\pi)^p$ for the parameters to be optimized by the routine. What should be noted is that the heuristic inducing the termination condition in Line 2 is an important ingredient of Algorithm 12 that influences the quality of the returned solution. Nelder and Mead [NM65] used a certain tolerance below which the standard deviation of function values corresponding to the points of the current simplex needs to fall for the iterative procedure to break. Analogously, the effect of the initialization in Line 1 should not be underestimated: An initial simplex whose volume is, for instance, too small may lead to a search that is confined in a local region.

C. Quantum Circuit Notation

Let me provide a brief overview of the typical notation for drawing quantum circuits in order to provide clarity and facilitate the comprehension of Chapter 6 where the implementation of the QTG and the Grover mixing and phase separation unitaries is showcased. However, there is no claim to completeness; a more extensive introduction to the quantum circuit model is e.g. given by Nielsen and Chuang [NC10, Ch.4].

A class of simple diagrams following certain construction guidelines is mostly used to depict quantum circuits - the term "circuit" even refers to this type of visualization, inspired by electric circuits. In this model, a horizontal line - also called *wire* - represents a single qubit. A unitary operation (a gate) U is then usually denoted by a rectangle with an according label. This is everything we need for the simplest imaginable (non-trivial) quantum circuit:

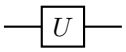
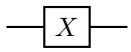


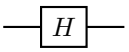
Figure 1.: Simplest possible non-trivial quantum circuit consisting of only one gate U acting on a single qubit.

Special single-qubit gates are the Pauli-x operator $X \equiv \sigma^x := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and the Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ (cf. Section 2.2.1), represented as



(a) First representation of the Pauli-x operator, referring to its index in the set of Pauli matrices.

(b) Second representation of the Pauli-x operator, reflecting its action given by $|0\rangle \mapsto |1\rangle$ and $|1\rangle \mapsto |0\rangle$.

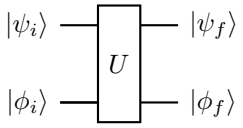


(c) Representation of the Hadamard transformation.

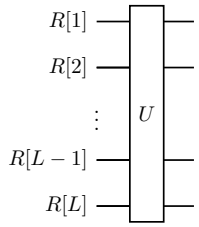
Figure 2.: Elementary quantum circuits holding only standard single-qubit gates.

In case of more than just one gate, the diagrams are to be read in the usual flow from left to right. Note that the order of application cannot be ignored, as two quantum operators generally do not commute. If the quantum circuit operates on more than one qubit (which is the case most of the time), labels on the left hand side of the wires are often added to distinguish the involved qubits. These can either refer to the index of a qubit in a certain register (typically written as a capital letter abbreviating the register and square brackets holding an index, e.g. $R[i]$) - a quantum circuit may be composed of multiple qubit registers - or to indicate the input to the circuit, i.e. the initial state of the respective qubit before the circuit is applied. On the contrary, the right hand side of a wire is usually only non-empty in the latter case where it is used

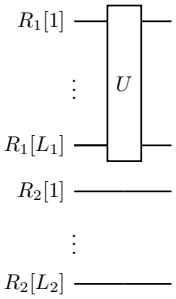
to depict the qubit's output state (the index of a register does not change in the course of the circuit application and does not need to be written twice). The trivial example for more than one employed qubit is the occurrence of a two-qubit (or multiple-qubit) gate. This is displayed via rectangles spanned over the wires of all affected qubits. In case that the involved qubits are separated by other qubits, an empty gate and a vertical connection are - at least in this thesis - used to emphasize that. The following should cover the important cases:



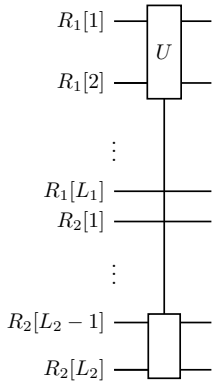
(a) Two-qubit unitary U acting on initial states $|\psi_i\rangle, |\phi_i\rangle$; yielding output states $|\psi_f\rangle, |\phi_f\rangle$.



(b) Unitary U acting on all L register-R qubits.



(c) Unitary U acting on all L_1 qubits in the register R_1 while none of the qubits in the register R_2 with size L_2 is affected.

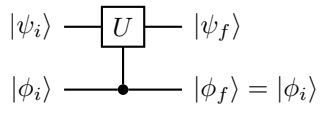


(d) Unitary U acting on some qubits contained in register R_1 and some from register R_2.

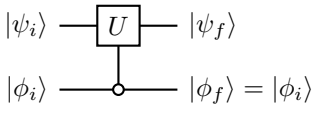
Figure 3.: Elementary quantum circuits for different variants of multiple-qubit gates U.

An important special case of multiple-qubit operations is a controlled gate where a single-qubit operation U shall only be applied if one more other qubits are in certain states, either $|1\rangle$ (denoted by a filled dot) or $|0\rangle$ (empty dot). For the simplest case of only two qubits where the first is w.l.o.g. assumed to be the target, see Fig. 4.

A quantum circuit may, in some sense, also depend on classical bits. Their non-quantum character is incorporated via doubled wires. If an operation is controlled on a classical bit, the vertical line used to denote a control in Fig. 4 is drawn twice analogously. Inputs to classical bits are embedded in round brackets instead of ket symbols; register indices however do not have a different notation. In principle, the circuit in Fig. 4a thus transforms to Fig. 5 when the second entry is of classical nature instead.



(a) Control on state $|1\rangle$.



(b) Control on state $|0\rangle$.

Figure 4.: Elementary quantum circuits with input states $|\psi_i\rangle, |\phi_i\rangle$ and output (final) states $|\psi_f\rangle, |\phi_f\rangle$ for a unitary U controlled on the second qubit. The input and output states on the second register entry are equal, since a control has no effect on the qubit's state.

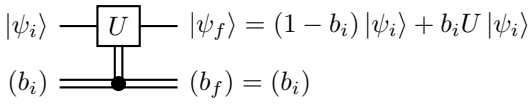


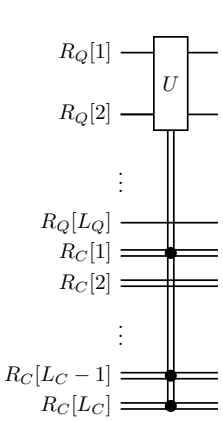
Figure 5.: Elementary quantum circuit with input state $|\psi_i\rangle$ on the first entry and value (b_i) on the second as well as output (final) states $|\psi_f\rangle, (b_f)$ for a unitary U controlled on the classical bit having value 1. The input and output values of the classical bit are equal, since a classical control has no effect on the qubit's state.

Analogously, we obtain the circuit for a gate U controlled on a classical bit with value 0. Note that controlling on a classical bit is nothing but a conventional "If" statement. Of course, a gate might also be controlled on more than one qubit or bit; the circuit diagrams reflect this by extended vertical lines with breakpoints at the control (qu-)bits. Interesting in particular is how to denote controls that include both qubits and classical bits. Exemplary circuits can be viewed in Fig. 6.[11] Moreover, a bundle of classical wires (three closely spaced horizontal lines) stands for a collection of classical bits, e.g. to collapse multiple bits encoding a classical quantity.

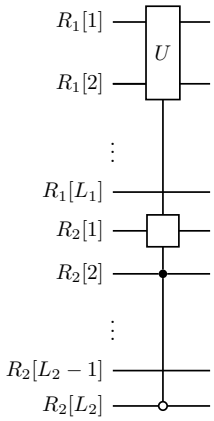
One custom notation needed to be introduced in this thesis: At some point, we wish to control an operation on another qubit and an additional classical bit where the explicit state of on which the gate is controlled depends on the bit value. For example, a unitary U could be applied to the first qubit in a quantum register R_Q when a second qubit in that register is in state $|1\rangle$ if a classical bit carries value 1 or when it is in state $|0\rangle$ if the classical bit has value 0. This generalization is pretty useful in situations where the classical bit represents a circuit parameter. We will use black squares on both the quantum and the classical control wire to indicate such a dependency.[12] In order to keep the circuit diagrams as simple and lucid as possible, we establish the rule that qubits and classical bits belonging together share the same index in their quantum and classical registers in case that a gate features multiple unspecified controls. This

[11] In this work, quantum registers will always be printed above classical ones.

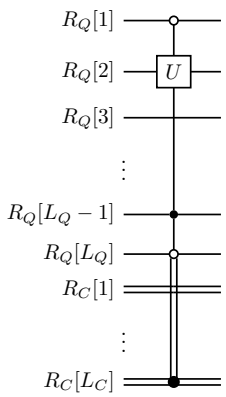
[12] Beware that there are, in contrast to the usual control symbols seen above, no empty squares - the different shape alone allows to promote the switch to an unspecified control.



(a) Unitary U acting on two positions in the quantum register R_Q, controlled on three bits with value 1 from the classical register R_C.



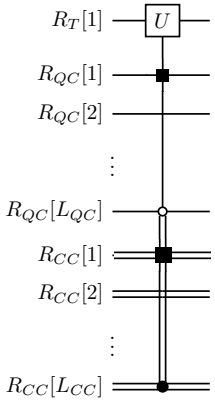
(b) Unitary U acting on two positions in the first and one position in the second register, controlled on two qubits from the second register, one each on being in state $|1\rangle$ and $|0\rangle$.



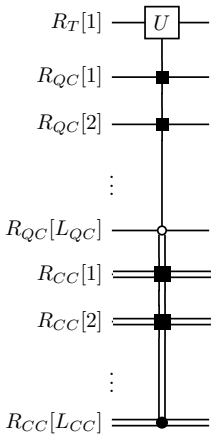
(c) Single-qubit unitary U being controlled on three qubits from the quantum register R_Q (once on state $|1\rangle$ and twice on state $|1\rangle$) and one bit in the classical register R_C.

Figure 6.: Elementary quantum circuits for different variants of multiple-controlled gates U.

should become clearer with Fig. 7 as an example. Note that, as suggested by Fig. 7, this logic can be arbitrarily combined with usual controls, multiple-qubit gates and everything else.



(a) Single-qubit unitary U, controlled i.a. on the first R_{QC}-qubit being in state $|R_{CC}[1]\rangle$.



(b) Single-qubit unitary U, controlled i.a. on the first R_{QC}-qubit being in state $|R_{CC}[1]\rangle$ and the second in state $|R_{CC}[2]\rangle$.

Figure 7.: Elementary quantum circuits for unspecified controls with target registers R_T, quantum-controlling registers R_{QC} and classical-controlling registers R_{CC}.

The notation elaborated on here should be sufficient to avoid ambiguity in Chapter 6.

Bibliography

[AHU83] A V Aho, J E Hopcroft, and J D Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983. ISBN: 9780201000238. URL: https://books.google.de/books?id=k8pQAAAAMAAJ.

[Amb04] A Ambainis. "Quantum Search Algorithms". In: *SIGACT News* 35 (2 June 2004), pp. 22–35. ISSN: 0163-5700. DOI: 10.1145/992287.992296. URL: https://doi.org/10.1145/992287.992296.

[Aru+19] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574 (7779 Oct. 2019), pp. 505–510. ISSN: 14764687. DOI: 10.1038/s41586-019-1666-5.

[Bar+98] Cynthia Barnhart et al. "Branch-and-Price: Column Generation for Solving Huge Integer Programs". In: *Operations Research* 46 (3 1998), pp. 316–329. DOI: 10.1287/opre.46.3.316. URL: https://doi.org/10.1287/opre.46.3.316.

[BE20] Andreas Bärtschi and Stephan Eidenbenz. "Grover Mixers for QAOA: Shifting Complexity from Mixer Design to State Preparation". In: (May 2020). DOI: 10.1109/QCE49297.2020.00020. URL: http://arxiv.org/abs/2006.00354%20http://dx.doi.org/10.1109/QCE49297.2020.00020.

[Bel72] Richard Bellman. *Dynamic programming*. Princeton University Press, 1972, p. 342. ISBN: 069107951X.

[Ben80] Paul Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". In: *Journal of Statistical Physics* 22 (5 May 1980), pp. 563–591. ISSN: 00224715. DOI: 10.1007/BF01011339.

[BF28] M Born and V Fock. *Beweis des Adiabatensatzes*. 1928.

[Bil23] Bundesministerium für Bildung und Forschung. *Handlungskonzept Quantentechnologie beschlossen*. Apr. 2023. URL: https://www.bmbf.de/bmbf/shareddocs/kurzmeldungen/de/2023/04/230425-handlungskonzept-quantentechnologien.html.

[Bin+23] Lennart Binkowski et al. *From barren plateaus through fertile valleys: Conic extensions of parameterised quantum circuits*. 2023.

[Bin22] Lennart Binkowski. *Constraint Graph Model Analysis of the Quantum Alternating Operator Ansatz*. 2022.

[BM10] Stephen Boyd and Jacob Mattingley. *Branch and Bound Methods*. 2010.

[Boi+18] Sergio Boixo et al. "Characterizing quantum supremacy in near-term devices". In: *Nature Physics* 14 (6 June 2018), pp. 595–600. ISSN: 17452481. DOI: 10.1038/s41567-018-0124-x.

[Bra+00] Gilles Brassard et al. *Quantum Amplitude Amplification and Estimation*. 2000.

[Cer+21] M. Cerezo et al. *Variational quantum algorithms*. Sept. 2021. DOI: 10.1038/s42254-021-00348-9.

[Cha23] Shantanav Chakraborty. "Implementing any Linear Combination of Unitaries on Intermediate-term Quantum Computers". In: (Feb. 2023). URL: http://arxiv.org/abs/2302.13555.

[Che52] Herman Chernoff. *A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations*. 1952.

[Cla99] Jens Clausen. *Branch and Bound Algorithms-Principles and Examples*. 1999.

[CN22] Hugh Collins and Chris Nay. *IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two*. Nov. 2022. URL: https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two?mhsrc=ibmsearch_a&mhq=osprey.

[Coo+] William J Cook et al. *Algorithms and Combinatorics*. URL: http://www.springer.com/series/13.

[Coo71] Stephen A Cook. "The Complexity of Theorem-Proving Procedures". In: Association for Computing Machinery, 1971, pp. 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047. URL: https://doi.org/10.1145/800157.805047.

[Cro+14] Elizabeth Crosson et al. "Different Strategies for Optimization Using the Quantum Adiabatic Algorithm". In: (Jan. 2014). URL: http://arxiv.org/abs/1401.7320.

[CW12] Andrew M Childs and Nathan Wiebe. *Hamiltonian Simulation Using Linear Combinations of Unitary Operations*. 2012.

[Dan08] Emilie R Danna. *Performance variability in mixed integer programming*. 2008.

[Deu85] David Deutsch. *Quantum theory, the Church-Turing principle and the universal quantum computer*. 1985.

[DH99] Christoph Dürr and Peter Høyer. *A quantum algorithm for finding the minimum*. 1999.

[Dia22] Oliver Dial. *Eagle's quantum performance progress*. Mar. 2022. URL: Eagle%E2%80%99s%20quantum%20performance%20progress.

[DJ92] David Deutsch and Richard Jozsa. *Rapid Solution of Problems by Quantum Computation*. 1992. URL: https://about.jstor.org/terms.

[Dra00] Thomas G Draper. *Addition on a Quantum Computer*. 2000. URL: http://xxx.lanl.gov/quant-ph..

[Dzi05] Jacek Dziarmaga. "Dynamics of a quantum phase transition: Exact solution of the quantum ising model". In: *Physical Review Letters* 95 (24 Dec. 2005). ISSN: 10797114. DOI: 10.1103/PhysRevLett.95.245701.

[Far+00] Edward Farhi et al. *Quantum Computation by Adiabatic Evolution*. 2000.

[Fey82] Richard P Feynman. *Simulating Physics with Computers*. 1982.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A Quantum Approximate Optimization Algorithm". In: (Nov. 2014). URL: http://arxiv.org/abs/1411.4028.

[FM14] Matteo Fischetti and Michele Monaci. "Exploiting erraticism in search". In: *Operations Research* 62 (1 Jan. 2014), pp. 114–122. ISSN: 0030364X. DOI: 10.1287/opre.2013.1231.

[For09] Lance Fortnow. *The status of the P versus NP problem*. Sept. 2009. DOI: 10.1145/1562164.1562186.

[GE21] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (2021), pp. 1–31. ISSN: 2521327X. DOI: 10.22331/Q-2021-04-15-433.

[GH19] Pierre Dupuy de la Grand'rive and Jean-Francois Hullo. "Knapsack Problem variants of QAOA for battery revenue optimisation". In: (Aug. 2019). URL: http://arxiv.org/abs/1908.02210.

[Glo65] Fred Glover. *A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem*. 1965. URL: https://about.jstor.org/terms.

[GM19] G. G. Guerreschi and A. Y. Matsuura. "QAOA for Max-Cut requires hundreds of qubits for quantum speed-up". In: *Scientific Reports* 9 (1 Dec. 2019). ISSN: 20452322. DOI: 10.1038/s41598-019-43176-9.

[Gro05] Lov K. Grover. "Fixed-point quantum search". In: *Physical Review Letters* 95 (15 Oct. 2005). ISSN: 00319007. DOI: 10.1103/PhysRevLett.95.150501.

[Gro96] Lov K Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[Gro98] Lov K Grover. *A framework for fast quantum mechanical algorithms*. 1998.

[Had+19] Stuart Hadfield et al. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz". In: (Sept. 2019). DOI: 10.3390/a12020034. URL: http://arxiv.org/abs/1709.03489%20http://dx.doi.org/10.3390/a12020034.

[Had18] Stuart Hadfield. "Quantum Algorithms for Scientific Computing and Approximate Optimization". In: (May 2018). URL: http://arxiv.org/abs/1805.03265.

[Hal15] Brian C Hall. *Lie Groups, Lie Algebras, and Representations - An Elementary Introduction*. 2015. URL: http://www.springer.com/series/136.

[Har+21] Matthew P. Harrigan et al. "Quantum approximate optimization of non-planar graph problems on a planar superconducting processor". In: *Nature Physics* 17 (3 Mar. 2021), pp. 332–336. ISSN: 17452481. DOI: 10.1038/s41567-020-01105-y.

[Hol22] Jan Rasmus Holst. "QAOA for the Knapsack Problem". In: (Nov. 2022).

[IBM23] Quantum IBM. *The IBM Quantum Development Roadmap*. 2023. URL: https://www.ibm.com/quantum/roadmap.

[Joh22] Blake Johnson. *Bringing the full power of dynamic circuits to Qiskit Runtime*. Nov. 2022. URL: https://research.ibm.com/blog/quantum-dynamic-circuits.

[Jor22] Stephen Jordan. *Quantum Algorithm Zoo*. June 2022. URL: https://quantumalgorithmzoo.org/.

[Kar72] Richard Karp. "Reducibility Among Combinatorial Problems". In: vol. 40. Apr. 1972, pp. 85–103. ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_8.

[Kay23] Alastair Kay. *Tutorial on the Quantikz Package*. 2023. DOI: 10.17637/rh.7000520.

[Koß+22] Gereon Koßmann et al. *Deep-Circuit QAOA*. 2022.

[KP99] Hans Kellerer and Ulrich Pferschy. *A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem*. 1999.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Apr. 2004. ISBN: 978-3-540-40286-2. DOI: 10.1007/978-3-540-24777-7.

[Lad+10] T. D. Ladd et al. *Quantum computers*. 2010. DOI: 10.1038/nature08812.

[LD60] A H Land and A G Doig. *An Automatic Method of Solving Discrete Programming Problems*. 1960. URL: https://about.jstor.org/terms.

[Lit+63] John D C Little et al. "An Algorithm for the Traveling Salesman Problem". In: *Operations Research* 11 (6 1963), pp. 972–989. DOI: 10.1287/opre.11.6.972. URL: https://doi.org/10.1287/opre.11.6.972.

[Llo18] Seth Lloyd. "Quantum approximate optimization is computationally universal". In: (Dec. 2018). URL: http://arxiv.org/abs/1812.11075.

[Llo96] Seth Lloyd. "Universal Quantum Simulators". In: *Science* 273 (5278 1996), pp. 1073–1078. DOI: 10.1126/science.273.5278.1073. URL: https://www.science.org/doi/abs/10.1126/science.273.5278.1073.

[Luc14] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (2014), pp. 1–14. ISSN: 2296424X. DOI: 10.3389/fphy.2014.00005.

[Man21] Ryan Mandelbaum. *Five years ago today, we put the first quantum computer on the cloud. Here's how we did it.* May 2021. URL: https://research.ibm.com/blog/quantum-five-years.

[MBZ19] Mauro E. S. Morales, Jacob Biamonte, and Zoltán Zimborás. "On the Universality of the Quantum Approximate Optimization Algorithm". In: (Sept. 2019). DOI: 10.1007/s11128-020-02748-9. URL: http://arxiv.org/abs/1909.03123%20http://dx.doi.org/10.1007/s11128-020-02748-9.

[McC+18] Jarrod R. McClean et al. "Barren plateaus in quantum neural network training landscapes". In: *Nature Communications* 9 (1 Dec. 2018). ISSN: 20411723. DOI: 10.1038/s41467-018-07090-4.

[Mon15] Ashley Montanaro. "Quantum algorithms: an overview". In: (Nov. 2015). DOI: 10.1038/npjqi.2015.23. URL: http://arxiv.org/abs/1511.04206%20http://dx.doi.org/10.1038/npjqi.2015.23.

[Mor+16] David R. Morrison et al. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19 (Feb. 2016), pp. 79–102. ISSN: 15725286. DOI: 10.1016/j.disopt.2016.01.005.

[Mor+17] David R. Morrison et al. "Cyclic best first search: Using contours to guide branch-and-bound algorithms". In: *Naval Research Logistics* 64 (1 Feb. 2017), pp. 64–82. ISSN: 15206750. DOI: 10.1002/nav.21732.

[MPT99] Silvano Martello, David Pisinger, and Paolo Toth. "Dynamic programming and strong bounds for the 0-1 Knapsack Problem". In: *Management Science* 45 (3 1999), pp. 414–424. ISSN: 00251909. DOI: 10.1287/mnsc.45.3.414.

[MT90] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990. ISBN: 0471924202.

[MW19] S. Marsh and J. B. Wang. "A quantum walk-assisted approximate algorithm for bounded NP optimisation problems". In: *Quantum Information Processing* 18 (3 Mar. 2019). ISSN: 15700755. DOI: 10.1007/s11128-019-2171-3.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010, p. 676. ISBN: 9781107002173.

[Nei+18] C Neill et al. *A blueprint for demonstrating quantum supremacy with superconducting qubits*. 2018. URL: https://www.science.org.

[NM65] John A Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *Comput. J.* 7 (1965), pp. 308–313.

[Pag+20] Guido Pagano et al. "Quantum approximate optimization of the long-range Ising model with a trapped-ion quantum simulator". In: 117 (41 2020), pp. 25396–25401. DOI: 10.1073/pnas.2006373117/-/DCSupplemental.y.

[Pis05] David Pisinger. "Where are the hard knapsack problems?" In: *Computers and Operations Research* 32 (9 2005), pp. 2271–2284. ISSN: 03050548. DOI: 10.1016/j.cor.2004.03.002.

[Pon+23] Moises Ponce et al. "Graph decomposition techniques for solving combinatorial optimization problems with variational quantum algorithms". In: (June 2023). URL: http://arxiv.org/abs/2306.00494.

[PR91] Manfred Padberg and Giovanni Rinaldi. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems". In: *SIAM Review* 33 (1 1991), pp. 60–100. DOI: 10.1137/1033004. URL: https://doi.org/10.1137/1033004.

[Pre18] John Preskill. "Quantum Computing in the NISQ era and beyond". In: (Jan. 2018). DOI: 10.22331/q-2018-08-06-79. URL: http://arxiv.org/abs/1801.00862%20http://dx.doi.org/10.22331/q-2018-08-06-79.

[Qia+18] Xiaogang Qiang et al. "Large-scale silicon quantum photonics implementing arbitrary two-qubit processing". In: *Nature Photonics* 12 (9 Sept. 2018), pp. 534–539. ISSN: 17494893. DOI: 10.1038/s41566-018-0236-y.

[Qua23] BMBF Quantentechnologien. *Handlungskonzept Quantentechnologien in der Bundesregierung*. Apr. 2023. URL: https://www.quantentechnologien.de/artikel/handlungskonzept-quantentechnologien-der-bundesregierung.html.

[RMO22] Jonathan Ruane, Andrew McAfee, and William D. Oliver. *Quantum Computing for Business Leaders*. Jan. 2022. URL: https://hbr.org/2022/01/quantum-computing-for-business-leaders.

[Roc+20] Christoph Roch et al. "Cross Entropy Hyperparameter Optimization for Constrained Problem Hamiltonians Applied to QAOA". In: (Mar. 2020). URL: http://arxiv.org/abs/2003.05292.

[RSA78] R L Rivest, A Shamir, and L Adleman. *Programming Techniques A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. 1978.

[Sch05] Maximilian Schlosshauer. *Decoherence, the measurement problem, and interpretations of quantum mechanics*. 2005.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In: IEEE Computer Society, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[Sho95] Peter W Shor. *Scheme for reducing decoherence in quantum computer memory*. 1995.

[SSA16] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alán Aspuru-Guzik. "qHiPSTER: The Quantum High Performance Software Testing Environment". In: (Jan. 2016). URL: http://arxiv.org/abs/1601.07195.

[Sve+21] Marika Svensson et al. "A Hybrid Quantum-Classical Heuristic to solve large-scale Integer Linear Programs". In: (Mar. 2021). URL: http://arxiv.org/abs/2103.15433.

[Vir+20] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[Wei73] Franz Weinberg. "Branch and Bound: Eine Einführung". In: 1973.

[Wil+23] Sören Wilkening et al. "A quantum algorithm for the solution of the 0-1 Knapsack problem". In: (Oct. 2023). URL: http://arxiv.org/abs/2310.06623.

[Zho+18] Leo Zhou et al. "Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices". In: (Dec. 2018). DOI: 10.1103/PhysRevX.10.021067. URL: http://arxiv.org/abs/1812.01041%20http://dx.doi.org/10.1103/PhysRevX.10.021067.

[ZJ22] David Zierler and Stephen Jordan. *Interview with Stephen Jordan*. Jan. 2022. URL: https://heritageproject.caltech.edu/interviews-updates/stephen-jordan.

Declaration of Ownership

I, Paul Johann Christiansen, hereby declare that this thesis is the result of my independent work and that no sources, auxiliary materials or means other than those indicated were used. Furthermore, all passages of the work that make reference to other sources, whether through direct quotation or paraphrasing, have been indicated accordingly. This thesis does not contain material which has already been used to any substantial extent for a comparable purpose.

Hanover, November 20, 2023 Paul Johann Christiansen

Acknowledgements

Lastly, I would like to thank people without whose kind support this thesis would not have been possible. First and foremost, special thanks to Prof. Tobias Osborne for the opportunity to write this thesis in his research group. Highly acknowledged are the fruitful discussions with Lennart Binkowski, Sören Wilkening, Timo Ziegler and Andreea Lefterovici as well as extensive proofreading. Finally, I want to thank my parents who originally paved my way with their unwavering encouragement.