Gottfried Wilhelm Leibniz University Hanover
Institute for Theoretical Physics

QAOA for the Knapsack Problem

Bachelor's Thesis

by
Jan Rasmus Holst
(10019521)

submitted on
November 04, 2022

Supervisor:
Prof. Dr. Tobias J. Osborne

Abstract

The knapsack problem is a well-known optimization problem with many applications and a wide range of closely related problems. There are advanced classical algorithms which allow for solving large instances of it in short time, but these are ultimately limited by a superpolynomially scaling worst-case runtime. Quantum computing might allow improving performance beyond this possibly fundamental limit. A widely discussed approach for optimization on the quantum devices of the near future is the quantum approximate optimization algorithm (QAOA) [FGG14]. In this thesis, we give an overview of different approaches to applying QAOA to the knapsack problem, including detailed descriptions of the quantum circuits and some improvements upon the current literature on this topic. We implement these approaches using the Qiskit library [Ani+21] and explore their behavior applied to small instances of the knapsack problem by means of numerical simulation.

Contents

Chapter 1

Introduction

The knapsack problem is an optimization problem with a multitude of different applications. It has been researched for over a hundred years due to its simple structure and its close connection to a wide variety of related problems [KPP04, Ch. 1]. This research has led to a range of different algorithms for solving the knapsack problem, and together with a drastic increase of computing power it continuously pushed the boundary of which problems were considered solvable in a feasible amount of time [MPT00; JLD22]. Despite this, there is no known algorithm for which the worst case runtime scales polynomially. This is widely believed to be a fundamental limit of classical computation, although it remains unproven [Coo00]. For very large instances of the knapsack problem or also for smaller instances of related problems, this is an issue.

A relatively recent contender for computational improvements beyond this possibly fundamental limit is quantum computing. Since its theoretical conception in the 1980s [Ben80; Deu85; Fey81], different quantum algorithms have been found which promise a scaling behavior that is not believed to be possible or which is even provenly impossible to reach with classical algorithms. The two most prominent examples of this are Shor's algorithm for factorizing integers [Sho94] and Grover's algorithm for unstructured search [Gro96]. Considerable progress has been made in developing the required computational hardware and large improvements in both the number of qubits as well as their quality have been announced for the relatively near future (e.g. [IBM22], see also [Had+19, Section 2]).

In recent years, it has been debated what quantum computers may be used for until large numbers of error corrected qubits are available [Pre18]. A widely discussed approach for this era of NISQ (Noisy Intermediate Scale Quantum) devices is the quantum approximate optimization algorithm (QAOA) [FGG14]. This is a quantum algorithm for finding approximate solutions to combinatoric optimization problems, such as the knapsack problem. So far, only few papers exist which discuss how QAOA might be applied to the knapsack problem. It is more than questionable whether for the knapsack problem a quantum algorithm on a NISQ device will outperform the very advanced classical algorithms and hardware currently available, as this would require solving problem instances with thousands of items. Nonetheless, exploring the application of QAOA to the knapsack problem is interesting, as results obtained in the research on algorithms for NISQ devices might be useful in the era of large scale error corrected quantum computing. Further, the knapsack problem is the simplest instance of a large class of closely related problems, so that results for the knapsack problem have a certain generality and can often be adapted to the other problems.

The goal of this thesis is to give an overview of different approaches to applying QAOA to the knapsack problem and to explore their behavior for small problem instances by means of numerical simulation. A key concern is the detailed presentation of the necessary quantum circuits in a manner, that is independent of the specific problem instance.

This thesis is structured as follows. In Chapter 2, we start by introducing the knapsack problem and giving a definition of which particular type of knapsack problem is discussed here. We give an overview of different variants of the problem, to which the results of this thesis might be linked, after which we discuss the problem's complexity and briefly present different classical approaches to solving it. In Chapter 3, we describe the conceptual framework of QAOA and motivate it by sketching its connection to quantum computing by adiabatic evolution. We discuss how QAOA may be applied to both unconstrained as well as constrained optimization problems. For the latter, different approaches to enforcing the constraints are presented. These form the basis for Chapter 4, in which three different approaches to applying QAOA to the knapsack problem are presented. For each approach a detailed description of the quantum circuits is given. For two of the three presented approaches, improvements compared to the current literature on the subject have been made. Chapter 5 is concerned with studying these approaches numerically. They are implemented using the Qiskit library [Ani+21] and applied to small instances of the knapsack problem. The dependence of their performance on different parameters is explored by means of simulation. Finally, in Chapter 6, we summarize the key results of this thesis and give an outlook on what next steps or further questions might be.

Prior knowledge of quantum computing is assumed. For an introduction to the field, we refer to the monograph by Nielsen and Chuang [NC10].

Chapter 2

The Knapsack Problem

In this chapter we present the Knapsack Problem, providing an overview of some of its variants and generalizations. We discuss both its complexity and different classical approaches to solving it.

First we give a definition of the knapsack problem and introduce some terminology and naming conventions which are used in the rest of this thesis (Section 2.1). This is especially important as the name "knapsack problem" is used for a wide range of similar but distinct problems. We discuss some of those variants in Section 2.2 to give a brief overview of closely related problems. In Section 2.3, we introduce some central concepts of computational complexity theory to discuss the computational complexity of the knapsack problem, which we later refer to in order to motivate a quantum approach. Finally, we give a brief overview of some of the main classical approaches used to solve the knapsack problem and discuss their limitations (Section 2.4).

This chapter mainly summarizes canonical knowledge. It is heavily based on Chapters 1, 2 and Appendix A of [KPP04], with slight changes in nomenclature. As only a very brief summary of the most central concepts is possible in the scope of this thesis, it is not possible to be rigorous. Instead we refer to the according sections of [KPP04] for a more detailed and comprehensive discussion.

2.1 Definition

The term "knapsack problem" refers to a range of different combinatorial optimization problems concerned with finding an optimal combination of items from a set, while satisfying one or more constraints which limit the possible item choices. We use the term to refer to a problem often called the "0-1 knapsack problem". This is the main topic of this thesis. We will continue with a brief description of the problem.

An instance of the knapsack problem contains $N \in \mathbb{N}$ items $j = 0, \ldots, N-1$. Each item j has an associated value $v_j \in \mathbb{N}$ (also referred to as profit in some sources, such as [KPP04]), and weight $w_j \in \mathbb{N}$. Every item may either be chosen or not, represented by a binary variable $x_j \in \{0, 1\}$. We refer to the resulting vector $x = (x_0, \ldots, x_{N-1})$ as a choice. For each choice x there are an associated value $v(x) = \sum_{j=0}^{N-1} v_j x_j$ and an associated weight $w(x) = \sum_{j=0}^{N-1} w_j x_j$. Which choices of items are allowed is determined by a maximum weight or capacity $W \in \mathbb{N}$ and we refer to a choice as valid or feasible if the weight constraint $w(x) \leq W$ is satisfied. The objective is to find a feasible choice x

maximizing the value $v(x)$. Expressed in the typical form of an integer linear program one might write this compactly as

$$\text{maximize } v(x) = \sum_{j=0}^{N-1} v_j x_j \tag{2.1}$$

$$\text{subject to } w(x) = \sum_{j=0}^{N-1} w_j x_j \leq W, \tag{2.2}$$

$$x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N . \tag{2.3}$$

In this definition of the knapsack problem, and in this thesis in general, we assume a few things. Firstly, we assume that $v_j, w_j \in \mathbb{N}\ \forall j \in \{0, \ldots, N-1\}$ and $W \in \mathbb{N}$. As explained in [KPP04, p.10], one may assume $v_j > 0, w_j > 0$ without loss of generality. Also cases of $v_j, w_j \in \mathbb{Q}_{>0}$ may be converted to the domain of natural numbers by multiplication with a sufficiently large integer, but this may cause a large overhead for some algorithms (e.g. dynamic programming), making the generalization to $\mathbb{Q}_{>0}$ a non-trivial task. Ultimately this assumption is motivated by the practical reason that algorithms are easier to design for $v_j, w_j \in \mathbb{N}\ \forall j \in \{0, \ldots, N-1\}$ and $W \in \mathbb{N}$. Secondly, we assume that $w_j \leq W\ \forall j \in \{0, \ldots, N-1\}$, as all other items may never be chosen without violating the weight constraint and are thus irrelevant to solving the problem. Finally, we assume that $w_{\text{total}} = \sum_{j=0}^{N-1} w_j > W$, as otherwise the weight constraint is always satisfied and thus the whole problem becomes trivially solvable by choosing all items.

While this is one of the most basic integer linear programs, and the possible values of x_j, v_j, w_j may seem quite restricted, there are abundantly many use cases of the knapsack problem: in logistics, financial investment, or anywhere where limited goods need to be partitioned in the most profitable way. Also, as the knapsack problem is basic, it occurs as a subproblem of many other optimization problems, so that findings for the knapsack problem may be beneficial for solving other optimization problems.

2.2 Variants

As mentioned before, there are many problems that are typically labeled as a knapsack problem, or at least closely related to the knapsack problem defined in Section 2.1. In this section, we will give a brief overview of some variants to give an impression of the variety of closely related problems which may be of interest.

One of the most basic variants of the knapsack problem is the *subset sum problem*. This is the special case of $w_j = v_j\ \forall j \in \{0, \ldots, N-1\}$ which is useful when the values and weights are proportional to each other, as in this case the structure of the knapsack problem collapses to this more simple structure.

Another variant is the *bounded knapsack problem*. Here we look at instances where many items are identical, and incorporate this into the formulation of the problem. Instead of an item j being chosen or not, it is allowed to be chosen up to b_j times, i.e. $0 \leq x_j \leq b_j$. The b_j are called upper bounds, hence the name of the variant. Rephrasing a problem in this manner might be useful, because this allows for choosing a more efficient data structure for representing a choice. Closely related is the *unbounded knapsack problem*, where each item may be chosen arbitrarily many times. This is a variant of the bounded knapsack

problem with upper bounds so large that they are not actually relevant anymore, i.e. they may be assumed infinite.

When there is more than one property of the items resulting in constraints, e.g. weight and volume instead of just weight, this may be expressed as a *d-dimensional knapsack problem*, where d refers to the number of constraining properties. This is basically the knapsack problem with d independent weight constraints. Similarly, the *multiple knapsack problem* is the knapsack problem with multiple knapsack which may each have their own capacities and which are filled using one set of items, i.e. each item may only occur in one of the knapsacks at once. Finally, the *multiple choice knapsack problem* is a knapsack problem with different categories of items, where only one item from each category may be chosen.

Combinations of these different variants are also possible and many real world problems may only be representable as such.

In this thesis we will only discuss the knapsack problem defined in Section 2.1, being in some sense the most basic of them, such that the underlying principles of the presented approaches may be demonstrated more clearly and the approaches themselves may be more easily adapted to other variants.

2.3 Complexity

In this section we discuss the complexity of the knapsack problem, first summarizing the main concepts from the field of computational complexity theory (Section 2.3.1) in order to classify the knapsack problem in these terms (Section 2.3.2).

2.3.1 Summary of Complexity-Theoretic Concepts

The field of computational complexity theory explores how problems may be classified based on a specific notion of complexity, and how these different classes or problems within these classes relate to each other. Two important complexity classes are P and NP, which classify decision problems, i.e. problems that pose a "yes"/"no" question. Roughly speaking, P refers to those decision problems that are *efficiently solvable*, i.e. those for which an efficient solving algorithm exists. In contrast, NP refers to those which are *efficiently verifiable*, i.e. provided an answer to the decision problem as well as some proof for the validity of the decision (also called a certificate), there is an algorithm which efficiently checks the validity of the decision. In particular all problems in P are part of NP.

For a function $f : \mathbb{R} \to \mathbb{R}$, we define

$$\mathcal{O}(f) = \{g : \mathbb{R} \to \mathbb{R} \mid \exists\, c, x_0 > 0 \text{ s.t. } 0 \leq g(x) \leq cf(x)\ \forall x \geq x_0\} \tag{2.4}$$

Let $f(n)$ be the runtime of an algorithm for a specific size n of the given problem instance. If $\exists\, k \in \mathbb{N} : f \in \mathcal{O}(n^k)$, the algorithm has *polynomial runtime*. This is considered efficient in this context, highlighting the contrast to "inefficient" superpolynomial, e.g. exponential, runtime, and expressing its desirable properties, while not every polynomial runtime algorithm is efficient in practical terms.

A decision problem Q may be *reduced* to another decision problem R if there exists a polynomial time mapping between the inputs of Q and R, s.t. deciding R on the mapped input also decides Q and vice versa. This is denoted by $Q \leq R$. We use this to define the class of NP-complete problems: A decision problem Q is NP-complete, if $Q \in$ NP and

for all $R \in \text{NP} : R \leq Q$. These NP-complete problems may in some sense be seen as the hardest NP problems. They are especially interesting, as by finding an efficient algorithm for deciding an NP-complete problem, all NP problems would be efficiently decidable and P = NP. A lot of energy has been put into finding such an algorithm and so far none has been found, such that it is widely believed (although unproven) that P $\neq$ NP, i.e. no efficient algorithms for deciding NP-complete problems exist.

2.3.2 Complexity of the Knapsack Problem

The knapsack problem has an associated decision problem which we denote as KP-DECISION. For some given $t \in \mathbb{N}$ it is to be decided whether there exists a choice x with $v(x) \geq t$ and $w(x) \leq W$. This problem is NP, as for a given certificate x it is efficiently verifiable whether the conditions are satisfied. Even more, KP-DECISION is NP-complete.

The optimal value of a knapsack problem may be found by using binary search combined with solving KP-DECISION for each of the considered values. Further, by solving the knapsack problem, it is trivial to solve KP-DECISION. Therefore both problems may be mapped to each other with only a polynomial overhead. By this, the knapsack problem is classified as NP-hard. If the widely believed P $\neq$ NP [Coo00] turns out to be true, this means that neither for KP-DECISION nor for the knapsack problem an efficient algorithm exists.

2.4 Classical Approaches

For solving the knapsack problem, a variety of different classical algorithms has been developed. In this section we give a brief review of the most important ones. More details may be found in [KPP04, chapter 2].

2.4.1 Basic Approaches and Common Methods

One of the most basic approaches is the *Greedy algorithm*. The items are sorted by their ratio of value to weight (also called efficiency of the items). Then they are packed in descending order of efficiency. Items that are too heavy to be added, i.e. any items that would lead to exceeding the weight limit if they were added to the knapsack, are skipped. This procedure is carried out until the weight limit is reached and no more items can be added or until all items were considered. This method always produces a solution and does so in polynomial time: In the case of N items, the runtime of sorting the items is in $\mathcal{O}(N \log N)$, and the runtime of choosing the items is in $\mathcal{O}(N)$. For these two properties the solution quality is sacrificed, i.e. there is no guarantee on how good the solution will be and in fact it might be arbitrarily bad. Nonetheless, this algorithm is very useful, as it efficiently generates a lower bound for the value of the optimal solution.

Closely related is the *Greedy algorithm for the relaxed knapsack problem*. The relaxed knapsack problem is the linear programming relaxation of the knapsack problem, i.e. instead of a binary choice variable we allow $x_j \in [0, 1] \ \forall j \in \{0, \ldots, N-1\}$. As before, we sort the items and pack them in descending order of efficiency, but as soon as an item is too heavy, only the largest possible fraction of it is packed, so that the maximum weight is reached but not exceeded. All the subsequent items are ignored. As before, this always gives a solution in $\mathcal{O}(N \log N)$ (limited by the sorting), but in this case also reliably produces an optimal solution (the relaxed knapsack problem is P). Solving an instance of the relaxed knapsack problem gives an upper bound for the corresponding (integral) knapsack problem.

A more sophisticated method is *Dynamic Programming*. The method itself is quite general and may be applied to many problems. The idea is to solve small subproblems and iteratively build on those solutions to finally solve the whole problem. This saves time by considering every subproblem only once, in contrast to brute force approaches. For the knapsack problem the algorithm may be roughly described as follows: First, choose one item; Calculate the optimal solutions for all capacities up to W; Add another item; Check whether the optimal solutions for the different capacities need to be updated (which can be done very easily); Continue adding more items, and repeat this process until all items are added and you have found the optimal solution. This algorithm runs in $\mathcal{O}(NW)$ (pseudopolynomial time, not polynomial time) and always produces the optimal solution. For high values of W, using the algorithm becomes infeasible, even for small numbers of items.

Another quite sophisticated method is the *Branch-and-Bound* approach. It is also quite general, but very different from the dynamic programming approach. A brute force approach may consist of traversing all possible solutions and picking the best one (2^N possibilities). Branch-and-Bound does something similar, but doesn't look at every possibility explicitly. By using heuristics for finding upper and lower bounds of the best possible solutions in different parts of the state space, it is possible to exclude some parts of the state space, specifically those for which we can infer that they won't contain the optimal solution. This is the case if, e.g., the upper bound on the solution quality of a certain part of the state space is lower than the lower bound on the solution quality of the total state space. For the knapsack problem, we may use the Greedy algorithm to efficiently generate lower bounds and the Greedy algorithm for the relaxed problem to efficiently generate upper bounds. In some cases this approach can lead to drastic speedups as only a small part of the state space is explicitly considered and the heuristics themselves are very efficient. In other cases this may not give any improvement and it may be required to traverse all possibilities, resulting in a worst case runtime of $\mathcal{O}\big(f(N)2^N\big)$, where $f(N)$ is determined by the efficiency of the heuristics and the decision process.

2.4.2 Approximation Algorithms

Since Knapsack is an NP-hard problem, no exact algorithms with polynomial runtime are known and it is widely believed that such algorithms do not exist. Branch-and-Bound, Dynamic Programming, and brute force approaches opt for being exact at the cost of not having polynomial runtime. Approximation algorithms act complementarily by opting for polynomial runtime at the cost of sacrificing accuracy. The Greedy algorithm is an example of an approximation algorithm.

An important question lies in the quality of the approximation. E.g., the Greedy algorithm's solutions may be arbitrarily bad, depending on the specific instance of the knapsack problem. There are other approximation algorithms which have a lower bound on the quality of the solution and a typical way of expressing this is as a *relative performance guarantee*. Let I be an instance of a knapsack problem, $z^*(I)$ the value of an optimal solution, and $z^A(I)$ the value of the solution obtained by an approximation algorithm A. A has a relative performance guarantee of $k \in (0,1)$ if $\frac{z^A(I)}{z^*(I)} \geq k$ for all instances I. In this case A is called a k-approximation algorithm. An example is the *Ext-Greedy algorithm*, a slightly modified version of the Greedy algorithm: After running the Greedy algorithm on a given instance I, a solution z^G is aquired and Ext-Greedy returns $\max\left\{z^G, \max\{v_0, \ldots, v_{N-1}\}\right\}$. Ext-Greedy is a $\frac{1}{2}$-approximation algorithm and runs in $\mathcal{O}(N \log N)$, limited by the speed

of sorting the items.

Some types of algorithms allow for arbitrary accuracy: an algorithm A is an ϵ-approximation scheme if for every input $\epsilon \in (0, 1)$ the algorithm acts as an $(1 - \epsilon)$-approximation algorithm. If such an algorithm runs in polynomial time for every ϵ, it is called a Polynomial Time Approximation Scheme (PTAS). The cost for increasing accuracy (reducing ϵ) often is an exponential increase in runtime. Some ϵ-approximation schemes avoid this and run in time polynomial in N and $\frac{1}{\epsilon}$ (Fully Polynomial Time Approximation Scheme, FPTAS). This comes at the cost of an increase in space requirements, rendering the approach impractical in many cases. For the knapsack problem, such PTASs and FPTASs exist, but they are not discussed here.

2.4.3 Current State of the Art

In 2004, Kellerer et al. write in the context of describing more advanced exact algorithms of the knapsack problem: "From practical experience it is known that many [knapsack problem] instances of considerable size can be solved within reasonable time by exact solution methods." [KPP04, page 118]. The authors refer to [MPT00], where different approaches for exactly solving the knapsack problem are presented and compared for differrent types of knapsack problem instances. There, algorithms are reported to solve problems with up to 100,000 items within seconds on what they classify as "easy" instances and similarly impressive results for harder instances. The combo algorithm [MPT99], a combination of different algorithms employing more advanced algorithmic concepts than we can cover here, is concluded to be superior to the other presented approaches, solving all compared types of instances with up to 10,000 items in less than 0.2 seconds. In [Pis05], new types of harder knapsack problem instances are presented. On these, the combo algorithm is shown to solve problems of 10,000 items on an Intel Pentium IV processor at a clock frequency of 3 GHz and with 1 GB of RAM in a matter of a few seconds. These numbers have to be taken with a grain of salt. On the one hand, CPU performance has continued to improve over the last few years, while on the other hand the types of knapsack problem instances used for benchmarking have been further developed. Nonetheless, as of 2022, the combo algorithm is still regarded as state of the art and designing knapsack problem instances which are specifically challenging for the combo algorithm still poses a challenging task [JLD22].

Chapter 3

QAOA

The Quantum Approximate Optimization Algorithm (QAOA) is a quantum algorithm for finding approximate solutions to combinatorial optimization problems. It was introduced in 2014 by Farhi et al. [FGG14] and is widely discussed as an approach for optimization on Noisy Intermediate Scale Quantum (NISQ) devices [Pre18; Had+19].

In this chapter we present the QAOA ansatz by providing a description of the general framework and its application to unconstrained as well as constrained optimization problems. In Section 3.1, we give an overview of the QAOA, describing the approach for unconstrained optimization problems. In Section 3.2, we describe how that approach can be altered to solve constrained optimization problems.

For each ansatz a derivation is outlined. The derivations are not mathematically rigorous but are intended as an overview of an argument already presented in literature.

3.1 QAOA for Unconstrained Optimization Problems

In this section we present the QAOA approach for unconstrained optimization problems, as described in [FGG14]. We discuss the relation of QAOA and the Quantum Adiabatic Algorithm (QAA) [Far+00] and present an operator oriented perspective on QAOA. Finally we discuss QAOA as a variational quantum algorithm and briefly explore classical preprocessing.

Let $Z(N) = \{0,1\}^N$ be the space of length N bitstrings. Let $c : Z(N) \to \mathbb{N}$ be the objective function. We consider the optimization problem of maximizing c over $Z(N)$. We will only discuss maximization problems in this thesis, but the approaches can be easily adapted for minimization ones. In this problem, all $x \in Z(N)$ are feasible solutions, i.e. there are no constraints that restrict the possible values of x, but any $x \in Z(N)$ might possibly be a solution to the problem. An optimal solution of this problem is a bitstring $x^* \in Z(N)$ such that $c(x^*) = \max_{x \in Z(N)} c(x)$, that is c assumes its maximum value. Similarly an approximate solution of this problems is a bitstring $x^* \in Z(N)$ such that $c(x^*) \approx \max_{x \in Z(N)} c(x)$. The exact meaning of $\approx$ depends on the context.

Many problems can be described as such an optimization problem. We will briefly discuss the MaxCut problem, which is the focus of [FGG14], to exemplify the general approach. Given a graph $G = (V, E)$, the objective is to find a subset $S \subset V$ such that the number of edges between S and $V \setminus S$ is maximized. We number all N vertices $v_0, \ldots, v_{N-1}$. Each

vertex $v_j, j = 0, \ldots, N-1$ is either in S or not. We represent this using the variables x_j, $j = 0, \ldots, N-1$, defined by

$$x_j = \begin{cases} 1, & \text{if } v_j \in S \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Then each subset S is identified with a bitstring $x = x_0 \ldots x_{N-1}$. For each edge $e = \{v_i, v_j\} \in E$ we define $c_e : Z(N) \to \mathbb{N}$ with

$$c_e(x) = x_i x_j + (1 - x_i)(1 - x_j) = \begin{cases} 1, & \text{if } x_i \neq x_j \\ 0, & \text{if } x_i = x_j \end{cases}, \tag{3.2}$$

indicating whether the edge is between S_x and $V \setminus S_x$ or within one of them. We use this to define $c : Z(N) \to \mathbb{N}$ as

$$c(x) = \sum_{e \in E} c_e(x), \tag{3.3}$$

corresponding to the number of edges between S_x and $V \setminus S_x$. The task of finding an optimal subset S is thereby converted to the optimization problem of maximizing c. In [FGG14], an approach to finding approximate solutions for this type of optimization problem is described. We will now discuss the details of this approach.

The Hilbert space of a single qubit is $\mathcal{H} = \mathbb{C}^2$, with basis vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Therefore the Hilbert space of an N-qubit system (e.g. a quantum computer) is $\mathcal{H} = \mathbb{C}^{2^N}$ with basis vectors $|x_0, \ldots, x_{N-1}\rangle = |x_0\rangle \otimes \cdots \otimes |x_{N-1}\rangle$, $x_j \in \{0, 1\}$, $j = 0, \ldots, N-1$ forming the computational basis. We identify each basis vector $|x\rangle = |x_0 \ldots x_{N-1}\rangle$ with the corresponding bitstring $x = x_0 \ldots x_{N-1} \in Z(N)$.

Let the operator $C : \mathcal{H} \to \mathcal{H}$ be defined by $C|x\rangle = c(x)|x\rangle$ on the computational basis vectors $|x\rangle, x \in Z(N)$, and by linearity on the entire Hilbert space. An optimal solution $x^* \in Z(N)$ of c corresponds to a highest-value eigenstate $|x^*\rangle$ of C. QAOA aims to create such a highest value eigenstate (or at least a high valued eigenstate) and measure it in the computational basis to obtain a bitstring giving an optimal (or at least approximate) solution to the optimization problem.

The approach of QAOA is heavily based on the Quantum Adiabatic Algorithm (QAA) [Far+00]. We will sketch the connection between these two to motivate QAOA. For a rigorous discussion of their connection we refer to [Bin22, Chapter 3].

QAA is an algorithm for preparing the highest valued eigenstate of an operator like C by evolving into this state adiabatically from another, easily constructable state. The initial state is the highest energy eigenstate $|s\rangle$ of a different Hamiltonian B for which the highest energy eigenstate is easily constructable. We define

$$B = \sum_{j=0}^{N-1} \sigma_j^x, \tag{3.4}$$

where σ_j^x is the Pauli X matrix, applied to the j-th qubit. The highest energy eigenstate of B is

$$|s\rangle = \frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} |x\rangle \tag{3.5}$$

in computational basis. This state can be constructed by taking $|0\rangle$ as an initial state and applying a Hadamard gate to each qubit. We then gradually alter the Hamiltonian over the span of a time T such that it becomes the objective Hamiltonian C, e.g. by interpolating linearly between the two.

$$H(t) = \left(1 - \frac{t}{T}\right) B + \left(\frac{t}{T}\right) C \tag{3.6}$$

For sufficiently large T the evolution of the system's state is adiabatic and (assuming the conditions for applying the adiabatic theorem hold) $|s\rangle \mapsto |x^*\rangle$, where $|x^*\rangle$ is a highest value eigenstate of C. By measuring $|x^*\rangle$ in the computational basis, an optimal solution $x^* \in Z(N)$ for the problem of maximizing c is obtained.

We will now describe the connection of QAA and QAOA.

The time evolution generated by (3.6) is described by the unitary operator

$$U\left(t\right) = \lim_{n\to\infty} \prod_{j=0}^{n} \mathrm{e}^{-iH(j\frac{t}{n})\frac{t}{n}} \tag{3.7}$$

We recall that H is the weighted sum of B and C, allowing us to apply the Trotter product formula. For two same-size quadratic matrices A and B we can rewrite

$$\mathrm{e}^{A+B} = \lim_{n\to\infty} \left(\mathrm{e}^{A/n}\ \mathrm{e}^{B/n}\right)^n \tag{3.8}$$

Thus

$$U(t) = \lim_{n\to\infty} \prod_{j=0}^{n} \lim_{m\to\infty} \left(\mathrm{e}^{-i\left(1-\frac{jt}{nT}\right)B\frac{t}{nm}}\ \mathrm{e}^{-i\left(\frac{jt}{nT}\right)C\frac{t}{nm}}\right)^m \tag{3.9}$$

We are interested in the unitary mapping $U(T) : |s\rangle \mapsto |x^*\rangle$. We can write this as

$$U(T) = \lim_{n\to\infty} \prod_{j=0}^{n} \lim_{m\to\infty} \left(\mathrm{e}^{-i\left(1-\frac{j}{n}\right)B\frac{T}{nm}}\ \mathrm{e}^{-i\left(\frac{j}{n}\right)C\frac{T}{nm}}\right)^m \tag{3.10}$$

$$= \lim_{n\to\infty} \prod_{j=1}^{n} \left(\mathrm{e}^{-i\beta_j B}\ \mathrm{e}^{-i\gamma_j C}\right) \tag{3.11}$$

$$= \lim_{n\to\infty} \prod_{j=1}^{n} U_B(\beta_j)\ U_C(\gamma_j) \tag{3.12}$$

for suiting β_j, γ_j and $U_B(\beta_j) = \mathrm{e}^{-i\beta_j B}$, $U_C(\gamma_j) = \mathrm{e}^{-i\gamma_j C}$. We refer to U_B as mixing operator and U_C as phase separation operator. As B, C have integer eigenvalues, we can restrict $\gamma_j \in [0, 2\pi)$, $\beta_j \in [0, \pi)$ $\forall j$ and refer to them as angles.

For $n = p < \infty$ this equation still holds approximately, with the quality of this approximation depending on the size of p.

$$U\left(T\right) \approx \prod_{j=1}^{p} U_B\left(\beta_j\right) U_C\left(\gamma_j\right) \tag{3.13}$$

By applying this unitary to our initial state we obtain a state

$$|\beta, \gamma\rangle = \prod_{j=1}^{p} U_B\left(\beta_j\right) U_C\left(\gamma_j\right) |s\rangle \tag{3.14}$$

defined by the $2p$ angles $\beta = (\beta_1, \ldots, \beta_p)$ and $\gamma = (\gamma_1, \ldots, \gamma_p)$.

In QAOA, a quantum computer is used to create a N-qubit system in the state $|\beta, \gamma\rangle$. Repeatedly measuring this state in computational basis yields a distribution of binary strings. For a sufficiently good approximation to $U(T)$ this distribution contains an approximate or even optimal solution x^* to maximizing c.

The quality of this distribution may be improved by changing β and γ. Especially for small p, where the approximation of $U(T)$ is not very good, the choice of angles heavily influences the distribution and is of great significance to the quality of the results.

This gives two new approaches to solving combinatorial optimization problems. Firstly, for sufficiently large p, QAOA can be seen as a method of simulating QAA on a quantum computer using $2p$ unitaries. Secondly, and more important for NISQ technology, for small p, QAOA is a (possibly shallow-depth) approach to approximate optimization. In [FGG14] this second approach is emphasized.

In the regime of small p (e.g. $p = 1$), the QAA-based perspective presented above is questionable, as the approximation to $U(T)$ might be quite bad. Instead, an operator centered description is more adequate. A nice example of such a description can be found in [Had+19]. The focus lies on how exactly U_B and U_C act on the computational basis states. The phase separation operator U_C applies a phase to every basis state $|x\rangle$, depending on the value $c(x)$ of the objective function. For $x \in Z(N)$,

$$U_C(\gamma) |x\rangle = \mathrm{e}^{-i\gamma c(x)} |x\rangle . \tag{3.15}$$

The mixing operator U_B rotates each qubit around the x-axis (on its Bloch sphere), putting it in a superposition of $|0\rangle$ and $|1\rangle$ state, creating a superposition of all basis states. The behavior of the mixing operator is discussed in more detail in Section 3.2.2. The alternating application of phase separation and mixing operators, i.e. alternatingly applying phases based on the value of the objective function and creating superpositions of those states with different phases, leads to interference. This causes the amplitudes of some states to be larger than those of others. The choice of parameters β, γ influences the interference patterns (the amplitudes of the states) and thus the measured bitstring distribution. The quality of the bitstring distribution heavily depends on the choice of β, γ, i.e. optimizing c is translated to the task of optimizing β, γ to obtain a bitstring distribution in which good or optimal solutions are dominant.

Two methods for finding good β, γ are presented in [FGG14]. The more general method is using QAOA in the context of a hybrid quantum-classical variational algorithm. For a given set of parameters β, γ, the QAOA circuit is run n times, resulting in a distribution of bitstrings $x \in Z(N)$, each occurring n_x times. The quality of the distribution is judged using a metric, usually

$$\frac{1}{n} \sum_{x \in Z(N)} n_x c(x). \tag{3.16}$$

A classical optimizer is used to update β, γ. This process is repeated until it converges to a (local) optimum. The initial values of β, γ, the type of classical optimizer, the chosen metric, the construction of the QAOA circuit, and the choice of n influence the runtime and the quality of the results. Furthermore, depending on the problem, there might be a way to make an informed choice of parameters, narrowing down the size of the relevant parameter space and reducing the parameter optimization's runtime.

The other presented method is using classical preprocessing to determine the optimal values for β, γ without running the circuit. This method is only applicable in some cases. It requires an efficient way of computing the expectation value of the metric from β, γ, as a direct simulation of a QAOA circuit would scale exponentially. This is not trivial to find and requires exploiting the structure of the problem. An example of a case where this is possible is given in [FGG14, Section 2]. Here, the structure of MaxCut is exploited and a way of classically computing the expectancy $\langle \beta, \gamma | \, C \, | \beta, \gamma \rangle$ for $p = 1$ in polynomial runtime is found ($\langle \beta, \gamma | \, C \, | \beta, \gamma \rangle$ is the expectancy of $c(x)$ when sampling from a QAOA circuit, which is typically used as a metric for parameter optimization).

This thesis focuses on the QAOA circuits and does not discuss parameter optimization.

3.2 QAOA for Constrained Optimization Problems

So far the unconstrained optimization of a function $c : Z(N) \to \mathbb{N}$ has been considered. Now we consider the case that constraints are added, which restrict the feasible solutions to a subset $F \subset Z(N)$. In this case an optimal solution is a bitstring $x^* \in F$ such that $c(x^*) = \max_{x \in F} c(x)$, that is c assumes the maximum value possible without violating any constraints. An approximate solution is defined accordingly.

In this Section we present two general approaches to this problem. In Section 3.2.1, we discuss converting the hard constraint $x \in F$ to soft constraints. This allows for a direct application of the approach presented in Section 3.1. In Section 3.2.2, we discuss an approach to enforcing the hard constraints based on altering the mixing operator.

3.2.1 Conversion to Soft Constraints

The central idea of this approach is to not strictly enforce $x \in F$, but to modify c such that $x \in F$ are more favorable and optimization will result in bitstring distributions containing an (approximate) solution $x^* \in F$.

Let

$$\widetilde{c} : Z(N) \to \mathbb{Z}, \; x \mapsto c(x) + \text{penalty}(x) \tag{3.17}$$

be the modified objective function, where

$$\text{penalty}(x) = \begin{cases} 0, & \text{if } x \in F \\ < 0 \text{ and sufficiently large}, & \text{otherwise} \end{cases} \tag{3.18}$$

The definition of sufficiently large depends on the context, but usually this means a function with values large enough for suppressing $x \notin F$ in the optimized bitstring distributions and at least enforcing $x^* \in F$ for the $x^* \in Z(N)$ that maximizes $\widetilde{c}(x)$.

This approach results in an optimization problem of the same type as discussed in Section 3.1, so the approach presented there may be directly applied. The success of this approach depends on the concrete choice of penalty function: A too weak penalty function might not seperate feasible and non-feasible states correctly, a too strong penalty function might hinder the optimization of the parameters β and γ. As QAOA is only an approximate optimization algorithm, the results will also include bitstrings $x \in Z(N) \setminus F$ (even without noise).

Many examples of this approach may be found. Two examples of applying this approach to the knapsack problem are [GH19] and [Roc+20], which we will discuss in Chapter 4.

3.2.2 Enforcing Hard Constraints

In contrast, this section presents an approach strictly enforcing $x \in F$. This is achieved by altering the mixing operator U_B. This approach has been described briefly in [FGG14, Section VII] and expanded upon in [MW19].

In this section, we will first describe the approach from an operator oriented perspective. We start by discussing the mixing operator U_B that was used in Section 3.1 from a new perspective. We use this as a basis to define a new mixing operator that enforces $x \in F$. Secondly, we describe the connection between QAA and this approach. Finally, we briefly present the slightly different approach of using heuristic constraint enforcing mixers presented in [Had+19].

An Operator Centered Perspective

The mixing operator $U_B(\beta) = \mathrm{e}^{-i\beta B}$ used in Section 3.1 is generated by the mixing Hamiltonian

$$B = \sum_{j=0}^{N-1} \sigma_x^j. \tag{3.19}$$

An equivalent definition of B is

$$B \left|x\right\rangle = \sum_{j=0}^{N-1} \left|n_j(x)\right\rangle \ \forall x \in Z(N), \tag{3.20}$$

where $n_j(x)$ is the j-th neighbor of x (x with j-th bit flipped). In this definition, it is easy to see that

$$\left\langle x\right| B \left|x'\right\rangle = \begin{cases} 1, & \text{if } \mathrm{Ham}(x,x') = 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall x, x' \in Z(N) \tag{3.21}$$

where $\mathrm{Ham}(x,x')$ is the Hamming distance of x, x', i.e. the number of bit flips required to transform x to x'. Let $G = (V, E)$ be a hypercube graph, that is $V = Z(N)$ and $E = \{\{x, y\} \mid x, y \in V \text{ and } \mathrm{Ham}(x,y) = 1\}$. Then B is the adjacency matrix of G. Therefore, $U_B = \mathrm{e}^{-i\beta B}$ describes a quantum walk for time β on G. G is connected, so the quantum walk $U_B(\beta)$ creates a superposition of all states $\left|x\right\rangle, x \in V = Z(N)$, depending on the value of β and the previous state.

For the constrained optimization problem, the feasible solutions are restricted to a subset $F \subset Z(N)$. Strictly enforcing this constraint means ensuring that only bitstrings $x \in F$ appear in the measured bitstring distribution. This requires all states $\left|\beta, \gamma\right\rangle \in \mathcal{H}$ generated by the QAOA circuit to obey

$$\left|\beta, \gamma\right\rangle \in \mathcal{H}_F = \mathrm{span}_{\mathbb{C}} \left\{\left|x\right\rangle, \ x \in F\right\} \tag{3.22}$$

For this, we alter the mixing Hamiltonian. We define a subgraph $G_F = (F, E_F)$ of G which is restricted to feasible states, with $E_F = \{\{x, y\} \mid x, y \in F \text{ and } \mathrm{Ham}(x,y)\}$. We define B as the adjacency matrix of G_F by

$$\left\langle x\right| B \left|x'\right\rangle = \begin{cases} 1, & \text{if } x, x' \in F \text{ and } \mathrm{Ham}(x,x') = 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall x, x' \in Z(N). \tag{3.23}$$

B generates the quantum walk $U_B(\beta) = \mathrm{e}^{-i\beta B}$ on G_F, which we use as a constraint enforcing mixing operator. Note that the eigenvalues of B are not necessarily integer (nor rational), s.t. we may no longer restrict $\beta \in [0, \pi)$, but instead $\beta \in \mathbb{R}$.

We will now discuss the behavior of U_B in more detail. Let $x, x' \in F$. If there is a path between x, x' on G_F, we find that for $\beta \neq 0 :\ \langle x| U_B(\beta) |x'\rangle \neq 0$. If there is no path, then $\langle x| U_B(\beta) |x'\rangle = 0\ \forall \beta \in \mathbb{R}$. In particular, U_B acts as the identity on non-feasible states: Let $x \in Z(N) \setminus F$. Then $B|x\rangle = 0 \ \Rightarrow\ U_B(\beta)|x\rangle = \mathrm{e}^{-i\beta B}|x\rangle = \mathbb{I}|x\rangle$. From these properties, it follows that

$$\begin{aligned} &\forall |\psi\rangle \in \mathcal{H}_F :\ U_B|\psi\rangle \in \mathcal{H}_F \\ \text{and}\quad &\forall |\psi\rangle \in \mathcal{H}_{Z(N)\setminus F} :\ U_B|\psi\rangle \in \mathcal{H}_{Z(N)\setminus F}, \end{aligned} \tag{3.24}$$

where $\mathcal{H}_{Z(N)\setminus F} = \mathrm{span}_{\mathbb{C}}\{|x\rangle,\ x \in Z(N) \setminus F\}$. That is, feasible states are only mapped to feasible states and non-feasible states are only mapped to non-feasible states. Further, let G_F be connected and $|\psi\rangle \in \mathcal{H}_F$. Then, $U_B(\beta)|\psi\rangle$ is a superposition of all states $|x\rangle, x \in F$, i.e. all feasible states are mixed. This is not the case, if G_F is not connected, so that this is a necessary requirement for the approach to be applicable.

We know that C and thus U_C is diagonal in computational basis, so that

$$\begin{aligned} &\forall |\psi\rangle \in \mathcal{H}_F :\ U_C|\psi\rangle \in \mathcal{H}_F \\ \text{and}\quad &\forall |\psi\rangle \in \mathcal{H}_{Z(N)\setminus F} :\ U_C|\psi\rangle \in \mathcal{H}_{Z(N)\setminus F}. \end{aligned} \tag{3.25}$$

From Equation (3.14), we get that

$$\forall |s\rangle \in \mathcal{H}_F :\ |\beta, \gamma\rangle \in \mathcal{H}_F, \tag{3.26}$$

i.e. the constraints are strictly enforced by our QAOA circuit, when starting in a feasible initial state.

A Connection to QAA

We will now describe the connection of this approach to QAA. Again, this is not done with full mathematical rigor, for which we refer to [Bin22, Chapter 3].

We consider a transition from $-C$ to B over a sufficiently large time span T_1, in this case as a linear interpolation. The corresponding Hamiltonian is

$$H_1(t) = \left(1 - \frac{t}{T_1}\right)(-C) + \frac{t}{T_1} B \tag{3.27}$$

The time evolution generated by $H_1(t)$ is the unitary operator

$$U_1(t) = \lim_{n\to\infty} \prod_{j=0}^{n} \mathrm{e}^{-iH_1(j\frac{t}{n})\frac{t}{n}} \tag{3.28}$$

In a brief time interval $t \in [t_0, t_0 + \mathrm{d}t]$, we may assume $H_1(t) = H_1(t_0) = \text{const.}$, so that we can write

$$U_1(t_0 \to t_0 + \mathrm{d}t) = \mathrm{e}^{-iH_1(t_0)\mathrm{d}t} \tag{3.29}$$

$$= \mathrm{e}^{-i\left(\left(1-\frac{t_0}{T_1}\right)(-C)+\frac{t_0}{T_1}B\right)\mathrm{d}t} \tag{3.30}$$

Let $x \in Z(N) \setminus F$. Then $B|x\rangle = 0$ and thus

$$U_1(t_0 \to t_0 + \mathrm{d}t)\,|x\rangle = \mathrm{e}^{-i\left(\left(1-\frac{t_0}{T_1}\right)(-C)+\frac{t_0}{T_1}B\right)\mathrm{d}t}\,|x\rangle \tag{3.31}$$

$$= \mathrm{e}^{i\left(1-\frac{t_0}{T_1}\right)c(x)\mathrm{d}t}\,|x\rangle \tag{3.32}$$

from which we can conclude that

$$U_1\left(T_1\right)|x\rangle = \mathrm{e}^{i\phi(x)}\,|x\rangle \tag{3.33}$$

for an appropriate phase factor $\phi\left(x\right)$. As all basis vectors of $\mathcal{H}_{Z(N)\setminus F}$ are conserved up to a phase and U is unitary, we know that the orthonormal basis (ONB) $\{|x\rangle \mid x \in F\}$ of $\mathcal{H}_F$ will be mapped to an ONB of $\mathcal{H}_F$ again. That is

$$\forall\,|\psi\rangle \in \mathcal{H}_F :\ U_1(T_1)\,|\psi\rangle \in \mathcal{H}_F. \tag{3.34}$$

Now let $x \in F$. $|x\rangle$ is an eigenstate of $-C$ and specifically of $-C|_{\mathcal{H}_F}$. Assuming that the conditions of the adiabatic theorem are satisfied (which is not trivially true and depends on the structure of G_F), $U_1(T_1)$ maps all such $|x\rangle$ to eigenvectors of $B|_{\mathcal{H}_F}$. This map conserves the ordering of eigenvalues, i.e. the highest-eigenvalue eigenstate $|x_0\rangle$ of $-C|_{\mathcal{H}_F}$ (i.e. the ground state of $C|_{\mathcal{H}_F}$) is mapped to the highest-eigenvalue eigenstate $|\psi_B\rangle$ of $B|_{\mathcal{H}_F}$.

A similar argument can be made for a transition from B to C over a sufficiently large time span T_2. A suiting Hamiltonian is the linear interpolation

$$H_2\left(t\right) = \left(1 - \frac{t}{T_2}\right)B + \frac{t}{T_2}C \tag{3.35}$$

For the time development U_2 generated by H_2 we can conclude, by making the same argument as above, that the highest-eigenvalue eigenstate $|\psi_B\rangle$ of $B|_{\mathcal{H}_F}$ is mapped to the highest-eigenvalue eigenstate $|x^*\rangle$ of $C|_{\mathcal{H}_F}$. Thus, (under certain conditions) the adiabatic transition

$$-C \xrightarrow{T_1} B \xrightarrow{T_2} C \tag{3.36}$$

maps the ground state $|x_0\rangle$ of $C|_{\mathcal{H}_F}$ to the solution eigenstate $|x^*\rangle \in F$ of the constrained optimization problem. The unitary describing this transition is

$$U(T) = U_2(T_2)U_1(T_1), \tag{3.37}$$

with $T = T_1 + T_2$. It is generated by the piecewise Hamiltonian

$$H(t) = \begin{cases} H_1(t) & \text{for } 0 \le t \le T_1 \\ H_2(t - T_1) & \text{for } T_1 \le t \le T \end{cases}. \tag{3.38}$$

From here on the argument is nearly identical to that in Section 3.1. By applying the trotter product formula to the definitions of U_1, U_2, we can write

$$U_1(T_1) \approx \prod_{j=1}^{p_1} U_B(\beta_j^{(1)})\, U_C(\gamma_j^{(1)}) \quad \text{and} \tag{3.39}$$

$$U_2(T_2) \approx \prod_{j=1}^{p_2} U_B(\beta_j^{(2)})\, U_C(\gamma_j^{(2)}) \tag{3.40}$$

where $U_B(\beta) = \mathrm{e}^{-i\beta B}$ and $U_C(\gamma) = \mathrm{e}^{-i\gamma C}$, and thus

$$U(T) \approx \prod_{j=1}^{p} U_B(\beta_j)\, U_C(\gamma_j) \tag{3.41}$$

for $p = p_1 + p_2$ and suiting β_j, γ_j. We define

$$|\beta, \gamma\rangle = \prod_{j=1}^{p} U_B(\beta_j)\, U_C(\gamma_j)\, |x_0\rangle\,, \tag{3.42}$$

determined by the $2p$ angles $\beta = (\beta_1, \ldots, \beta_p)$ and $\gamma = (\gamma_1, \ldots, \gamma_p)$. This state plays the same role as Section 3.1. The key difference is that only binary strings $x \in F$ will be measured due to the structure of the mixing Hamiltonian.

Having drawn this connections between QAA and QAOA with the altered mixing operator, we know that for $p \to \infty$ the QAOA circuit will return an optimal solution, given that the conditions for optimality of QAA are satisfied. For a sufficiently high rate of convergence, we may hope to find feasibly small values of p for which the QAOA circuit produces good approximate solutions.

Heuristic Mixers

The focus of this thesis are the approaches that we discussed so far. Nonetheless we will briefly discuss the topic of heuristic mixers as presented in [Had+19]. This approach is not concerned with mixing Hamiltonians and the relation to QAA, but focuses on the mixing operators and their properties, especially for enforcing hard constraints. This is especially interesting for small p, where the connection to QAA is questionable. In [Had+19] different heuristic mixers and their application to a range of optimization problems are presented, and a guideline to designing good heuristic mixers is given. The mixers exploit the structure of the specific problem for which they are designed, allowing for a shallow depth combined with preserving feasibility (enforcing hard constraints). This is especially interesting for NISQ devices. So far, no application of this approach to the knapsack problem is known to the author.

Chapter 4

Approaches for Applying QAOA to the Knapsack Problem

The knapsack problem (as defined in Section 2.1) is a constrained optimization problem, as discussed in Section 3.2. Each possible choice of any of the N items is represented by a bitstring $x \in Z(N)$. The objective function is the value function

$$v : Z(N) \to \mathbb{N},\ x \mapsto v(x) = \sum_{j=0}^{N-1} v_j x_j. \tag{4.1}$$

The subset of feasible solutions is

$$F = \left\{ x \in Z(N) :\ w(x) = \sum_{j=0}^{N-1} w_j x_j \leq W \right\}. \tag{4.2}$$

In this chapter we will present three different approaches to applying the conceptual framework of QAOA to the knapsack problem, based on the general discussion of QAOA for constrained optimization problems (Section 3.2). First we will discuss using soft constraints with a penalty function scaling quadratically (Section 4.1). We will then introduce the concept of a feasibility oracle and describe the implementation of a feasibility oracle for the knapsack problem (Section 4.2). Based on this, we present a second approach based on soft constraints with a penalty function scaling linearly (Section 4.3). Finally, we will discuss an approach enforcing hard constraints (Section 4.4), using the quantum walk mixer presented in Section 3.2.2. This is also based on the feasibility oracle presented in Section 4.2. For all these approaches, we will present the idea behind the circuit, derive an implementation of it, and discuss its requirements in terms of qubits and gates.

4.1 Soft Constraints with Quadratic Penalty

In this subsection we describe an approach based on converting the hard constraint $w(x) \leq W$ of the knapsack problem to a soft constraint by introducing a penalty function that scales quadratically with the amount by which the maximum weight W is exceeded. This approach may be found nearly identically in [Luc14], [GH19] and [Roc+20]. The description here is heavily based on [Roc+20], while being more detailed. Slight alterations have been

made to the nomenclature and the problem is converted to a maximization problem for consistency within this thesis.

This approach quite strictly follows the outline in Section 3.2.1. The initial state $|s\rangle$ as well as the mixing Hamiltonian B and thus the mixing operator U_B are conceptually the same as those used in the unconstrained approach (Section 3.1). The difference lies in the phase separation operator U_C, or more precisely in the objective Hamiltonian C which generates U_C.

Let X, Y be two registers of N and W qubits respectively. Let $X[j], Y[n]$ denote the j-th qubit of the X register and the n-th qubit of the Y register respectively, $j = 0, \ldots, N-1$; $n = 0, \ldots W-1$. The computational basis states $|x\rangle, |y\rangle$ of X, Y are identified with bitstrings $x \in Z(N), y \in Z(W)$. X will be used for storing the item choices, i.e. a state $|x\rangle$ of X refers to an item choice $x \in Z(N)$. Y will be used for storing the current weight of the item choice in X as a one-hot encoding, i.e. a state $|y\rangle$ should be defined by the bitstring $y \in Z(W)$ with

$$y_n = \begin{cases} & \text{if } w(x) - 1 = n \\ 1, & \text{or } n = W-1 \text{ and } w(x) > W \\ & \text{or } n = 0 \text{ and } w(x) = 0 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in \{0, \ldots, W-1\}. \tag{4.3}$$

We will discuss the structure of the objective Hamiltonian by first discussing corresponding functions $h : Z(N) \times Z(W) \to \mathbb{Q}$ and $h_1, h_2 : Z(N) \times Z(W) \to \mathbb{Z}$. In the following, let sums over j and/or k be sums over $j, k = 0, \ldots, N-1$ and sums over n and/or l be sums over $n, l = 0, \ldots, W-1$, unless otherwise noted. Let

$$h_1(x, y) = v(x) = \sum_j v_j x_j \tag{4.4}$$

$$h_2(x, y) = -\left(1 - \sum_n y_n\right)^2 - \left(\sum_n (n+1) y_n - \sum_j w_j x_j\right)^2 \tag{4.5}$$

$\forall x \in Z(N), y \in Z(W)$. h_1 represents the value of an item choice x. h_2 acts as the penalty term while also enforcing the one-hot encoding of $w(x)$ in y: The first term enforces that $y_n = 1$ for exactly one value of n by penalizing deviations quadratically. The second term enforces that y represents $w(x)$ correctly, penalizing deviations quadratically. If $w(x) > W$, this is not possible, leading to a quadratically scaling penalty for non-feasible states. Let

$$h = a\, h_1 + b\, h_2, \tag{4.6}$$

where $a, b \in \mathbb{Q}_{>0} = \text{const}$ are used for adjusting the scaling of penalty and value. It is necessary that

$$0 < a \max_j v_j < b, \tag{4.7}$$

such that the penalty for adding an item with too large weight is higher than the value of the item, even if it is the most valuable item. Otherwise the optimal solution of h might be a non-feasible state. h is the function corresponding to the objective Hamiltonian C. We rephrase h in terms of Pauli Z matrices. Let $s_j = \sigma^z_{X[j]}$ the application of a Pauli Z matrix to qubit j of the X register $\forall j$. Similarly, let $u_j = \sigma^z_{Y[n]}$ $\forall n$. Replace

$$x_j \mapsto \frac{1}{2}(s_j + \mathbb{I}), \quad y_n \mapsto \frac{1}{2}(u_n + \mathbb{I}) \quad \forall j, n. \tag{4.8}$$

This results in

$$h_1 \mapsto H_1 = \sum_j \frac{v_j}{2} s_j \tag{4.9}$$

and

$$\begin{aligned} h_2 \mapsto H_2 = -\Bigg\{ & \sum_n \left[\frac{W}{2} - 1 + n \left(\frac{W^2+W}{4} - \frac{1}{2} \sum_j w_j \right) \right] u_n \\ & + \frac{1}{2} \sum_{n<l} (1+nl)\, u_n u_l \\ & + \frac{1}{2} \sum_j \left(-\frac{W^2+W}{2} + \sum_k w_k \right) w_j s_j \\ & + \frac{1}{2} \sum_{j<k} w_j w_k s_j s_k \\ & - \frac{1}{2} \sum_{n,j} n w_j u_n s_j \Bigg\}, \end{aligned} \tag{4.10}$$

defining the objective Hamiltonian

$$h \mapsto C = a\, H_1 + b\, H_2. \tag{4.11}$$

The constant terms are dismissed as they only result in a global phase of the phase separation operator

$$U_C(\gamma) = \mathrm{e}^{-i\gamma C}, \tag{4.12}$$

which is irrelevant. Rewrite $a, b \in \mathbb{Q}_{>0}$ as fully reduced fractions $a = \frac{a_1}{a_2}$, $b = \frac{b_1}{b_2}$, $a_1, b_1, a_2, b_2 \in \mathbb{N}$. $U_C(\gamma)$ is periodic in γ with periodicity $\frac{\mathrm{lcm}(a_2,b_2)}{\gcd(a_1,b_1)}\, 2\pi$, so that we can restrict

$$\gamma \in \left[0, \frac{\mathrm{lcm}(a_2, b_2)}{\gcd(a_1, b_1)}\, 2\pi \right), \tag{4.13}$$

where lcm and gcd refer to the lowest common multiple and the greatest common divisor, respectively. For the case of $a, b \in \mathbb{R}_{>0}$ this is not possible in general, as U_C is aperiodic for some irrational a and b.

We will now discuss how to implement U_C on a quantum computer. We re-write

$$H_2 = H_{y,1} + H_{y,2} + H_{x,1} + H_{x,2} + H_{x,y} \tag{4.14}$$

referring to the different terms of H_2 in the ordering of equation (4.10). The index reflects on which register and on how many qubits the term acts. All these terms commute with each other and with H_1. Thus, U_C may be written as

$$\begin{aligned} U_C(\gamma) &= \mathrm{e}^{-i\gamma(H_1 + H_{y,1} + H_{y,2} + H_{x,1} + H_{x,2} + H_{x,y})} \\ &= \mathrm{e}^{-i\gamma H_1} \mathrm{e}^{-i\gamma H_{y,1}} \mathrm{e}^{-i\gamma H_{y,2}} \mathrm{e}^{-i\gamma H_{x,1}} \mathrm{e}^{-i\gamma H_{x,2}} \mathrm{e}^{-i\gamma H_{x,y}}. \end{aligned} \tag{4.15}$$

Let $q_1, q_2 \in X \cup Y$ be qubits. We define the one- and two-qubit z-rotations

$$R^z_{q_1}(\theta) = \mathrm{e}^{-i\theta\, \sigma^z_{q_1}/2}, \tag{4.16}$$

$$R^{zz}_{q_1,q_2}(\theta) = \mathrm{e}^{-i\theta\, \sigma^z_{q_1} \otimes \sigma^z_{q_2}/2}. \tag{4.17}$$

We re-write

$$\mathrm{e}^{-i\gamma H_1} = \prod_j R^z_{X[j]}(\gamma a v_j) \tag{4.18}$$

$$\mathrm{e}^{-i\gamma H_{y,1}} = \prod_n R^z_{Y[n]}\left(-\gamma b\left[W - 2 + n\left(\frac{W^2+W}{2} - \sum_j w_j\right)\right]\right) \tag{4.19}$$

$$\mathrm{e}^{-i\gamma H_{y,2}} = \prod_{n<l} R^{zz}_{Y[n,l]}\left(-\gamma b\left(1 + nl\right)\right) \tag{4.20}$$

$$\mathrm{e}^{-i\gamma H_{x,1}} = \prod_j R^z_{X[j]}\left(-\gamma b\left(\sum_k w_k - \frac{W^2+W}{2}\right) w_j\right) \tag{4.21}$$

$$\mathrm{e}^{-i\gamma H_{x,2}} = \prod_{j<k} R^{zz}_{X[j,k]}\left(-\gamma b w_j w_k\right) \tag{4.22}$$

$$\mathrm{e}^{-i\gamma H_{x,y}} = \prod_{n,j} R^{zz}_{X[j],Y[n]}\left(\gamma b n w_j\right), \tag{4.23}$$

i.e. U_C may be expressed as a series of one- and two-qubit z-rotations with arbitrary ordering.

For completeness, the initial state for this problem is

$$|s\rangle = \frac{1}{\sqrt{2^{N+W}}} \sum_{x=0}^{2^N-1} |x\rangle \otimes \sum_{y=0}^{2^W-1} |y\rangle\,. \tag{4.24}$$

It is generated by applying a Hadamard gate to every qubit $q \in X \cup Y$, given an initial state $|0\rangle$. The mixing Hamiltonian is

$$B = \sum_j \sigma^x_{X[j]} + \sum_n \sigma^x_{Y[n]}. \tag{4.25}$$

This generates the mixing operator

$$U_B(\beta) = \mathrm{e}^{-i\beta B} = \prod_j R^x_{X[j]}(2\beta) \prod_n R^x_{Y[n]}(2\beta), \tag{4.26}$$

where

$$R^x_q(\theta) = \mathrm{e}^{-i\theta\,\sigma^x_q/2} \tag{4.27}$$

is the x-rotation for qubit $q \in X \cup Y$. As before, we may restrict $\beta \in [0, \pi)$.

An example of a $p = 1$ QAOA circuit for this approach, applied to a knapsack problem instance with $N = 3$ and $W = 2$ is shown in Figure 4.1[1]. To the left of 1, the initial state $|s\rangle$ is created. Between 1 and 4, the implementation of U_C is shown. Between 1 and 2, the single-qubit rotations are applied. Between 2 and 4 the two-qubit rotations are applied, first only acting on the X or Y register respectively, followed by the two-qubit rotations simultaneously acting on X and Y register. To the right of 4, the implementation of U_B is shown.

For implementing such an approach, $N + W$ qubits are required. Creating $|s\rangle$ and implementing U_B require $N + W$ single-qubit gates each. Implementing U_C requires $NW +$

[1]This and any further quantum circuit diagrams have been drawn using the Quantikz package [Kay19].

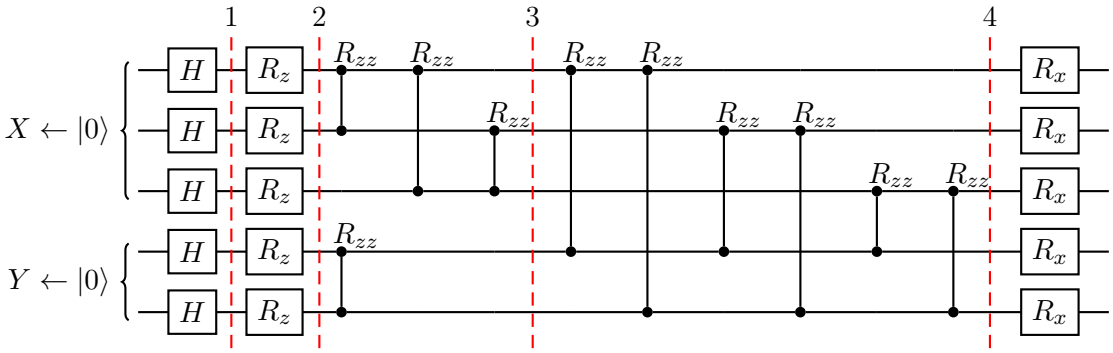


Figure 4.1: QAOA circuit with $p = 1$ for the quadratic penalty approach, applied to a knapsack problem instance with $N = 3$ and $W = 2$. The R_x, R_z, R_{zz} in the circuit diagram are identical to the R^x, R^z, R^{zz} in the explanation of the approach. The parameters were omitted for better readability.

$\frac{1}{2}N(N-1) + \frac{1}{2}W(W-1)$ two-qubit gates and $N + W$ single-qubit gates. This results in an overall requirement of $\mathcal{O}(p(N^2 + W^2))$ gates for a depth p QAOA circuit.

It is worth mentioning that in [Roc+20] a variant of this approach is briefly mentioned, encoding the weight in binary in the Y register and thus reducing the required number of qubits to $N + \lfloor \log_2 W + 1 \rfloor$.

4.2 Feasibility Oracle

We will now introduce the concept of a feasibility oracle, on which the approaches presented in Sections 4.3 and 4.4 are based, and present an implementation for the knapsack problem. The concept is based on [MW19]. The implementation is inspired by [GH19], but improved upon in this thesis.

Let a function

$$f : Z(N) \to \{0, 1\},\ x \mapsto f(x) = \begin{cases} 1 & \text{if } x \in F \\ 0 & \text{otherwise} \end{cases} \tag{4.28}$$

indicate the feasibility of a bitstring $x \in Z(N)$. We define a feasibility oracle to be a unitary U_f acting as

$$U_f |x, y\rangle = |x, y \oplus f(x)\rangle \quad \forall x \in Z(N), y \in Z(1), \tag{4.29}$$

i.e. a feasibility oracle toggles a flag qubit $|y\rangle$ based on whether a state $|x\rangle$, $x \in Z(N)$ is feasible.

For the knapsack problem, a state $|x\rangle, x \in Z(N)$ represents a choice of items. A choice is feasible ($x \in F$), if and only if $w(x) \leq W$. A feasibility oracle can be implemented by calculating $w(x)$ in an ancillary register and toggling a flag qubit based on whether $w(x) \leq W$.

Let X, A_W, and A_F be registers of lengths N, n, and 1 respectively. X is used for storing the choices, A_W is used for storing the weight of the item choice, and A_F is the flag qubit representing the feasibility of the state in X. To be able to store all possible weights $w(x)$

in A_W, it is necessary that

$$n \geq \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor + 1. \tag{4.30}$$

Initially $X \leftarrow |x\rangle\,,\ x \in Z(N)$, $A_W \leftarrow |0\rangle$, and $A_F \leftarrow |y\rangle\,,\ y \in Z(1)$. To calculate the weight, w_j is added to A_W, controlled by $X[j]$, for all $j \in \{0, \ldots, N-1\}$. The details of the implementation are discussed later. To test the satisfaction of the inequality, we use a multiple-controlled not gate. For suitable $W_0 \in \mathbb{N}$, $k \in \mathbb{N}$,

$$w(x) \leq W \quad \Leftrightarrow \quad w(x) + W_0 < W + W_0 + 1 = 2^k \tag{4.31}$$

$$\Leftrightarrow \quad (w(x) + W_0)_l = 0 \quad \forall l \in \mathbb{N}, l \geq k, \tag{4.32}$$

where $(w(x) + W_0)_l$ is the l-th digit of the binary representation of $w(x) + W_0$. That is, the inequality may be checked by adding a suiting W_0 to A_W after calculating $w(x)$, which we describe as the application of a unitary

$$U_1 : |x, 0, y\rangle \mapsto |x, (w(x) + W_0)\,, y\rangle\,, \tag{4.33}$$

and then applying a unitary

$$U_2 : |x, (w(x) + W_0)\,, y\rangle \mapsto \left|x, (w(x) + W_0)\,, y \oplus \left(w(x) + W_0 \leq 2^k\right)\right\rangle \tag{4.34}$$

$$= |x, w(x) + W_0, y \oplus f(x)\rangle \tag{4.35}$$

Combined, we find that after uncomputing U_1

$$U_f\,|x, 0, y\rangle = |x, 0, y \oplus f(x)\rangle = U_1^\dagger U_2 U_1\,|x, 0, y\rangle\,, \tag{4.36}$$

that is U_f may be implemented in terms of U_1 and U_2 with the help of an ancilla register set to $|0\rangle$.

U_1 is implemented by the circuit in Figure 4.2. For the addition we use an algorithm based on the quantum Fourier transform [Dra00], which does not require any ancillary qubits and requires $\mathcal{O}(n^2)$ gates for initialization and $\mathcal{O}(n)$ gates for each (controlled) addition (See Appendix A for a more detailed discussion of the addition algorithm). This is an improvement upon the classically inspired addition algorithm in [GH19] which requires an additional carry register of $n-1$ qubits while also requiring $\mathcal{O}(n^2)$ operations. This implementation of U_1 requires $\mathcal{O}(Nn^2)$ operations.

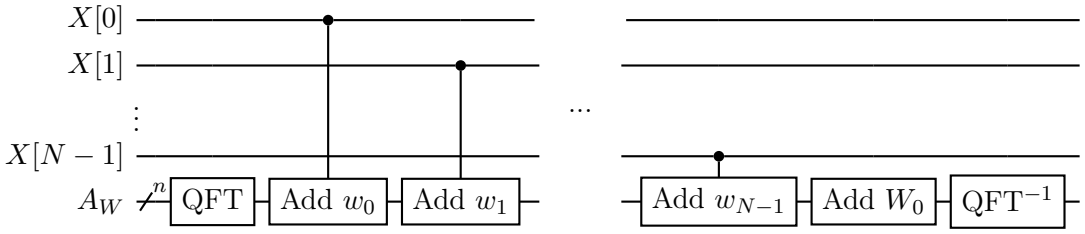


Figure 4.2: A circuit implementing U_1.

U_2 is implemented by the circuit in Figure 4.3, closely resembling the expression in (4.31). In this figure we choose $X = \sigma^x$ to denote the Pauli X or "not" gate. The multiple-controlled-not gate can be implemented without the use of any ancillary qubits and using

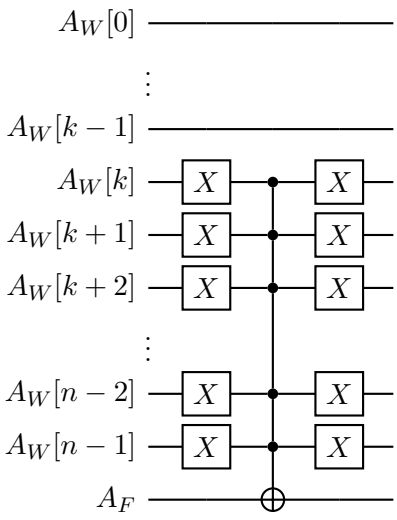


Figure 4.3: A circuit implementing U_2.

$\mathcal{O}(n^2)$ operations [SP22]. Thus the implementation of U_2 requires $\mathcal{O}(n^2)$ operations, so that the implementation of U_f requires $\mathcal{O}(Nn^2)$ operations.

We will now discuss the exact values of the parameters n, k, and W_0. We find that

$$k = \lfloor \log_2 W \rfloor + 1 \tag{4.37}$$

and

$$W_0 = 2^k - W - 1. \tag{4.38}$$

For the implementation of U_2 to be correct, it is necessary that $n > k$. We assumed that

$$\sum_{j=0}^{N-1} w_j > W. \tag{4.39}$$

From this we know, using the monotonicity of the $\log_2$ function, that

$$\left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor \geq \lfloor \log_2 W \rfloor . \tag{4.40}$$

For $\left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor > \lfloor \log_2 W \rfloor$, we find that the smallest n to satisfy both $n > k$ and (4.30) is

$$n = \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor + 1. \tag{4.41}$$

For $\left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor = \lfloor \log_2 W \rfloor$, the smallest possible value is

$$n = \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor + 2. \tag{4.42}$$

We choose n to be as small as possible and summarize this as

$$n = \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor + \begin{cases} 1 & \text{if } \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor > \lfloor \log_2 W \rfloor \\ 2 & \text{if } \left\lfloor \log_2 \sum_{j=0}^{N-1} w_j \right\rfloor = \lfloor \log_2 W \rfloor \end{cases}. \tag{4.43}$$

Therefore the number of required qubits is in $\mathcal{O}(N + \log \sum_{j=0}^{N-1} w_j)$ and the number of gates is in $\mathcal{O}(N(\log \sum_{j=0}^{N-1} w_j)^2)$.

4.3 Soft Constraints with Linear Penalty

In this subsection we describe an approach based on converting the hard constraint $w(x) \leq W$ of the knapsack problem to a soft constraint by introducing a penalty scaling linearly with the amount by which the maximum weight W is exceeded. The approach is inspired by [GH19], in which a similar approach is applied to a closely related problem, but the implementation has been improved upon in this thesis.

This approach also quite strictly follows the outline in Section 3.2.1. The initial state $|s\rangle$ as well as the mixing Hamiltonian B and thus the mixing operator U_B are conceptually the same as those used in the unconstrained approach (Section 3.1). The difference lies in the phase separation operator U_C, or more precisely in the objective Hamiltonian C which generates U_C.

The central idea is introducing a linearly scaling soft constraint to enforce $w(x) \leq W$. Let the objective function $c : Z(N) \to \mathbb{Q}$ be defined by

$$c(x) = v(x) + \text{penalty}(x) \tag{4.44}$$

with

$$\text{penalty}(x) = \begin{cases} 0, & \text{if } w(x) \leq W \\ -a\,(w(x) - W)\,, & \text{if } w(x) > W \end{cases} \tag{4.45}$$

and $a \in \mathbb{Q}_{>0}$ a constant parameter controlling the scaling of the penalty. Adding an item which makes the item choice non-feasible or adding an item to an already non-feasible choice must always be penalized. This must even be the case if the maximum weight W is only exceeded by 1. This requires

$$a > \max_j v_j. \tag{4.46}$$

By choosing $a \in \mathbb{Q}$, U_C is periodic in γ. Rewrite a as a fully reduced fraction $\frac{a_1}{a_2}$, $a_1, a_2 \in \mathbb{N}$. Due to this periodicity, we can restrict

$$\gamma \in [0, a_2\, 2\pi)\,. \tag{4.47}$$

The implementation of the penalty is based on a slight variation of the feasibility oracle. Multiple registers are necessary for this approach. Let X, A_W, and A_F be registers of size N, n, and 1, respectively, where n is chosen as in (4.43). Let $R[j]$ denote the j-th qubit of R for all registers R. The computational basis states $|x\rangle\,, x \in Z(N)$ of X are used for representing the item choices $x \in Z(N)$. A_F is used as a flag qubit indicating whether $x \in F$ for a state $|x\rangle$ of X. A_W is an ancillary register used by the feasibility oracle in computing the state of A_F as well as the penalty function.

The U_C corresponding to c must act on register X as

$$U_C(\gamma)\left|x\right\rangle = \mathrm{e}^{-i\gamma c(x)}\left|x\right\rangle = \mathrm{e}^{-i\gamma v(x)}\mathrm{e}^{-i\gamma \mathrm{penalty}(x)}\left|x\right\rangle \quad \forall x \in Z(N). \tag{4.48}$$

The value based phase can be directly applied using phase gates

$$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{i\theta} \end{pmatrix}. \tag{4.49}$$

Let $P_{R[j]}$ denote a phase gate P applied to qubit $R[j]$ for all registers R. We find that

$$\prod_{j=0}^{N-1} P_{X[j]}(-\gamma v_j)\left|x\right\rangle = \mathrm{e}^{-i\gamma v(x)}\left|x\right\rangle \quad \forall x \in Z(N). \tag{4.50}$$

The corresponding circuit is shown in Figure 4.4. The implementation requires N phase gates.

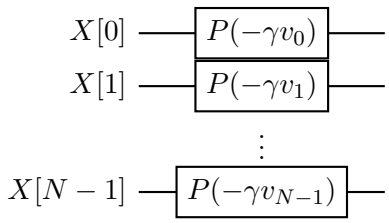


Figure 4.4: A circuit for applying the phase based on the value of the item choice.

The penalty based phase is implemented using a altered version of the feasibility oracle. Let

$$U_f' = U_2 U_1. \tag{4.51}$$

This acts as

$$U_f' : \left|x, 0, y\right\rangle \mapsto \left|x, w(x) + W_0, y \oplus f(x)\right\rangle \tag{4.52}$$

on X, A_W, and A_F and may be implemented as shown in Figures 4.2 and 4.3. In case the constraints are violated (which we encode as $A_F \leftarrow \left|1\right\rangle$), it is necessary to apply a phase $\mathrm{e}^{-i\gamma \mathrm{penalty}(x)}$ corresponding to the penalty value. Otherwise $\mathrm{penalty}(x) = 0$ and no phase is applied. This is implemented using phase gates controlled by A_F. For $w(x) > W$,

$$\mathrm{penalty}(x) = -a\left(w(x) - W\right) = -a\left(\left(w(x) + W_0\right) - \left(W + W_0\right)\right). \tag{4.53}$$

The phase $\mathrm{e}^{i\gamma a(w(x)+W_0)}$ is applied using the binary encoding of $w(x) + W_0$ stored in A_W. The phase $\mathrm{e}^{-i\gamma a(W+W_0)} = \mathrm{e}^{-i\gamma a 2^k}$ is applied using a phase gate on A_F. The resulting circuit is shown in Figure 4.5. The implementation requires $\mathcal{O}(n)$ gates.

After having applied the penalty based phase, it is necessary to restore all ancilla qubits to their original state. This uncomputing may be done by applying $(U_f')^\dagger$. An overview of the complete phase separation circuit is shown in Figure 4.6. Here we choose P_v and P_p to denote the value and penalty based applications of phase from Figures 4.4 and 4.5, respectively. The implementation requires $\mathcal{O}(Nn^2)$ gates and $N + n + 1$ qubits.

As in Section 3.1, the initial state is

$$\left|s\right\rangle = \frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} \left|x\right\rangle \otimes \left|0\right\rangle \otimes \left|0\right\rangle. \tag{4.54}$$

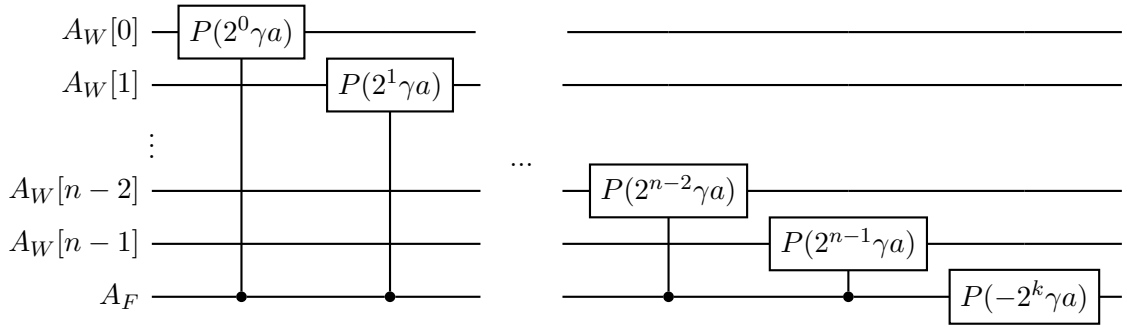


Figure 4.5: A circuit for applying a phase corresponding to the penalty value.

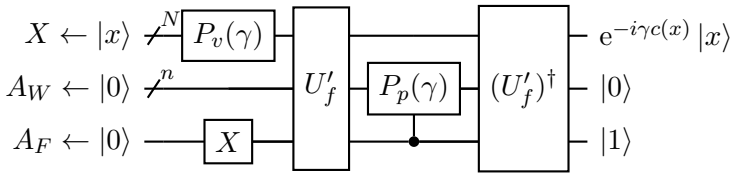


Figure 4.6: An overview of the circuit implementing the phase seperation operator U_C.

It is generated by applying a Hadamard gate to every qubit $q \in X$, given an initial state 0. Also the same mixing Hamiltonian

$$B = \sum_{j=0}^{N-1} \sigma^x_{X[j]} \tag{4.55}$$

and mixing operator

$$U_B(\beta) = \mathrm{e}^{-i\beta B} = \prod_{j=0}^{N-1} R^x_{X[j]}(2\beta) \tag{4.56}$$

are used. Both only act on the X register. As before, we may restrict $\beta \in [0, \pi)$.

This results in the complete QAOA circuit requiring $N+n+1$ qubits and $\mathcal{O}(pNn^2)$ gates. This is an improvement upon the approach presented in [GH19], which required $N + 2n$ qubits.

4.4 Hard Constraints Using Quantum Walk Mixer

So far in this chapter, we discussed two different approaches to implementing soft constraints, as described in Section 3.2.1. In this section we will present an implementation of the approach for enforcing hard constraints which we described in Section 3.2.2. We will present a general implementation of the quantum walk mixer. This implementation is based on [MW19], but some improvements have been made. The basis of this implementation is a feasibility oracle. We use the results from Section 4.2 to give a concrete implementation for the knapsack problem. We present implementations of initial state, mixer, and phase separation operator, and discuss the requirements in terms of qubits and gates. The description of this approach is heavily based on [MW19].

The basis of this approach is the mixing operator

$$U_B(\beta) = \mathrm{e}^{-i\beta B}. \tag{4.57}$$

It is generated by the mixing Hamiltonian B, defined in (3.23) by

$$\langle x| B |x'\rangle = \begin{cases} 1 & \text{if } x, x' \in F \text{ and } \mathrm{Ham}(x, x') = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \forall x, x' \in Z(N). \tag{4.58}$$

We can easily see that the corresponding graph of feasible states G_F, for which this is the adjacency matrix, is connected. We use the feasibility indicating function f defined in (4.28) and the j-th neighbor $n_j(x)$ of an item choice $x \in Z(N)$ defined in Section 3.2.2 as x with a flipped j-th bit. With this we define a function

$$f_j(x) = f(x)f(n_j(x)) \quad \forall x \in Z(N), \tag{4.59}$$

indicating whether x and its j-th neighbor are both feasible choices. We rewrite the definition of B as

$$B |x\rangle = \sum_{j=0}^{N-1} f_j(x) |n_j(x)\rangle \qquad \forall x \in Z(N). \tag{4.60}$$

It may be easily checked that both definitions (4.58) and (4.60) are equivalent.

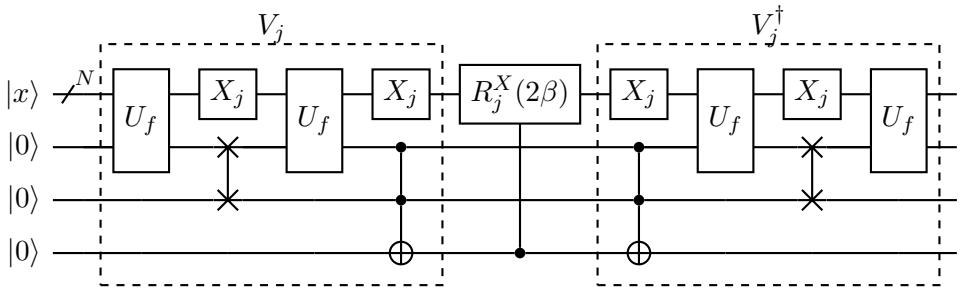


Figure 4.7: A general circuit for a single qubit quantum walk mixer.

We will now use this definition of B to derive our implementation of U_B. Let V_j be an operator defined by the the marked section of the circuit in Figure 4.7, where U_f is a feasibility oracle as defined in (4.29). Let $x \in Z(N)$ be an arbitrary item choice. V_j is unitary, that is

$$V_j^\dagger V_j |x, 0, 0, 0\rangle = V_j V_j^\dagger |x, 0, 0, 0\rangle = |x, 0, 0, 0\rangle . \tag{4.61}$$

It is easily checked that V_j behaves as

$$V_j |x, 0, 0, 0\rangle = |x, f(n_j(x)), f(x), f_j(x)\rangle . \tag{4.62}$$

From the unitarity of V_j we directly get

$$V_j^\dagger |x, f(n_j(x)), f(x), f_j(x)\rangle = |x, 0, 0, 0\rangle . \tag{4.63}$$

Furthermore we find that

$$\begin{aligned} &V_j^\dagger |n_j(x), f(n_j(x)), f(x), f_j(x)\rangle \\ &\quad = |n_j(x), f(x) \oplus f(n_j(x)), f(x) \oplus f(n_j(x)), f_j(x) \oplus f_j(x)\rangle \\ &\quad = \begin{cases} |n_j(x), 0, 0, 0\rangle & \text{if } f(x) = f(n_j(x)) \\ |n_j(x), 1, 1, 0\rangle & \text{otherwise} \end{cases} \end{aligned} \tag{4.64}$$

It is easily checked that a bit flip of the j-th qubit acts as

$$X_j |x\rangle = |n_j(x)\rangle . \tag{4.65}$$

Therefore we find together with (4.62) that a conditioned bit flip applied after V_j acts as

$$\left(X_j \otimes \mathbb{I}^{\otimes 2} \otimes |1\rangle\langle 1|\right) V_j \,|x,0,0,0\rangle = f_j(x)\,|n_j(x), f(n_j(x)), f(x), f_j(x)\rangle\,. \tag{4.66}$$

From this and (4.63), we get

$$V_j^\dagger \left(X_j \otimes \mathbb{I}^{\otimes 2} \otimes |1\rangle\langle 1|\right) V_j \,|x,0,0,0\rangle = f_j(x)\,|n_j(x),0,0,0\rangle\,. \tag{4.67}$$

Let

$$\tilde{B} = \sum_{j=0}^{N-1} V_j^\dagger \left(X_j \otimes \mathbb{I}^{\otimes 2} \otimes |1\rangle\langle 1|\right) V_j. \tag{4.68}$$

Using (4.67), we find that

$$\tilde{B}\,|x,0,0,0\rangle = \sum_{j=0}^{N-1} f_j(x)\,|n_j(x),0,0,0\rangle\,. \tag{4.69}$$

This closely resembles (4.60), from which we can conclude that $\tilde{B}$ implements B given three ancillary qubits, or more precisely

$$\tilde{B}\,|x,0,0,0\rangle = B \otimes \mathbb{I}^{\otimes 3}\,|x,0,0,0\rangle \quad \forall x \in Z(N). \tag{4.70}$$

From this it follows directly that

$$\begin{aligned} \mathrm{e}^{-i\beta\tilde{B}}\,|x,0,0,0\rangle &= \mathrm{e}^{-i\beta B\otimes\mathbb{I}^{\otimes 3}}\,|x,0,0,0\rangle \\ &= U_B(\beta) \otimes \mathbb{I}^{\otimes 3}\,|x,0,0,0\rangle \end{aligned} \tag{4.71}$$

and U_B is generated by $\tilde{B}$, using three ancilla qubits (plus those necessary for the implementation of U_f).

In general the terms of $\tilde{B}$ do not commute. Therefore, we implement $\mathrm{e}^{-i\beta\tilde{B}}$ only approximately, using the trotter product formula. Let us first discuss the terms of the trotter product individually. We find that $\forall j \in \{0,\ldots,N-1\}$

$$\begin{aligned} &\mathrm{e}^{-i\beta V_j^\dagger\left(X_j\otimes\mathbb{I}^{\otimes 2}\otimes|1\rangle\langle 1|\right)V_j} \\ &\quad= V_j^\dagger \mathrm{e}^{-i\beta X_j\otimes\mathbb{I}^{\otimes 2}\otimes|1\rangle\langle 1|} V_j \\ &\quad= V_j^\dagger \left(R_j^X(2\beta)\otimes\mathbb{I}^{\otimes 2}\otimes|1\rangle\langle 1| + \mathbb{I}^{\otimes(N+2)}\otimes|0\rangle\langle 0|\right) V_j \end{aligned} \tag{4.72}$$

The corresponding circuit is shown in Figure 4.7. Using the identity

$$R_j^X(2\beta) = \cos(\beta)\mathbb{I}^{\otimes N} - i\sin(\beta)X_j, \tag{4.73}$$

we find that

$$\begin{aligned} &V_j^\dagger \left(R_j^X(2\beta)\otimes\mathbb{I}^{\otimes 2}\otimes|1\rangle\langle 1| + \mathbb{I}^{\otimes(N+2)}\otimes|0\rangle\langle 0|\right) V_j \\ &\quad= \begin{cases} \cos(\beta)\,|x,0,0,0\rangle - i\sin(\beta)\,|n_j(x),0,0,0\rangle\,, & \text{if } f_j(x)=1 \\ |x,0,0,0\rangle\,, & \text{otherwise} \end{cases} \end{aligned} \tag{4.74}$$

Therefore, these terms may be interpreted as mixing x with its j-th neighbor, provided both are feasible. We refer to these terms as single qubit mixers. Exactly as for the unconstrained

approaches, we may restrict $x \in [0, \pi)$ for these terms. With these simplifications the complete trotter product formula can be written as

$$\mathrm{e}^{-i\beta\tilde{B}} = \left(\prod_{j=0}^{N-1} V_j^\dagger \left(R_j^X(2\beta/m) \otimes \mathbb{I}^{\otimes 2} \otimes |1\rangle\langle 1| + \mathbb{I}^{\otimes (N+2)} \otimes |0\rangle\langle 0| \right) V_j \right)^m + \mathcal{O}\left(\frac{N\beta^2}{m} \right) \tag{4.75}$$

giving us our implementation of $\mathrm{e}^{-i\beta\tilde{B}} = U_B(\beta) \otimes \mathbb{I}^{\otimes 3}$. As the β in the description of the single qubit mixers is replaced by a $\frac{\beta}{m}$ in the trotter product formula, we must adjust the restriction of β correspondingly, so that we get $\beta \in [0, m\pi)$. Given a feasibility oracle using n_q ancilla qubits and n_g gates, this implementation requires $N + n_q + 3$ qubits and $\mathcal{O}(Nmn_g)$ gates. This is an improvement upon the approach in [MW19], which requires $2N + n_q + 3$ qubits and twice as many uses of the feasibility oracle.

Specifically for the knapsack problem, we can use the feasibility oracle from Section 4.2. This requires $\mathcal{O}(\log \sum_{j=0}^{N-1} w_j)$ ancilla qubits and $\mathcal{O}(N(\log \sum_{j=0}^{N-1} w_j)^2)$ gates. Therefore the implementation of the mixer requires

$$\mathcal{O}\left(N + \log \sum_{j=0}^{N-1} w_j \right) \tag{4.76}$$

qubits and

$$\mathcal{O}\left(N^2 m \left(\log \sum_{j=0}^{N-1} w_j \right)^2 \right) \tag{4.77}$$

gates.

As an objective function we use the value function v. The corresponding phase separation operator is

$$U_C(\gamma) |x\rangle = \mathrm{e}^{-i\gamma v(x)} |x\rangle . \tag{4.78}$$

The implementation of this operator is already discussed in Section 4.3, specifically Equation (4.50) and Figure 4.4. It requires N single qubit phase gates.

The item choice $x \in Z(N)$ with the lowest value $v(x)$ is for all instances of the knapsack problem the choice of no items, i.e. $x = 0$. This choice is also feasible ($x \in F$) for all instances of the knapsack problem. Thus the state $|0\rangle$ is used as initial state, in accordance with Section 3.2.2.

The complete quantum circuit requires

$$\mathcal{O}\left(N + \log \sum_{j=0}^{N-1} w_j \right) \tag{4.79}$$

qubits and

$$\mathcal{O}\left(pN^2 m \left(\log \sum_{j=0}^{N-1} w_j \right)^2 \right) \tag{4.80}$$

gates.

Chapter 5

Simulation

How useful the approaches presented in Chapter 4 actually are, depends on how they perform compared to classical optimization algorithms. Due to the high cost of quantum computing hardware, the key factor is how they perform in the regime where classical algorithms fail to perform well - typically the regime of large problem sizes. A key problem is that currently existing quantum computers are limited to a small number of non-error corrected qubits [Gen21], limiting the size of problems and circuit depth of algorithms that may be currently explored. Simulations of quantum algorithms on classical computers are also limited to a small number of qubits due to the high computational overhead [Tur21]. This means that for the time being, statements about the performance of any quantum combinatorial optimization algorithm in these regimes must be analytically derived.

Nonetheless it can be interesting to investigate the behavior of quantum algorithms for small problem instances. Firstly, this is a possibility to learn about the specifics of implementation and how these may affect the results. Secondly, the observations made for small circuit sizes may be a good starting point for an analytical discussion of the algorithm at hand, as certain problems with the algorithm might already occur at small circuit sizes. Finally, with more efficient simulation algorithms and better quantum hardware, the implementations used for small instance simulation might be helpful as a starting point for exploring larger problem instances.

In this chapter we present implementations of the different approaches from Chapter 4 in Qiskit [Ani+21]. We discuss the behavior of these implementations for small problem instances by simulating them and explore the dependence of their performance on various parameters. After discussing the implementation as well as the methods used in this chapter (Section 5.1), we present optimized probability distributions generated using the different approaches as a first result (Section 5.2). In Section 5.3, we explore the quality of the probability distributions generated by the different approaches for different problem instances. In the following sections 5.4 - 5.6 the dependence of various other parameters is discussed. Finally, in Section 5.7, the implications of the results are discussed.

5.1 Implementation and Methods

For simulation, the different approaches from Chapter 4 have been implemented as quantum circuits in Qiskit [Ani+21]. The Python code for this chapter may be found as a git repository on GitHub [Hol22]. As this might be interesting for further research, the repos-

itory has been licensed under a permissive license and the implementation has been kept general, such that the code should work with arbitrary instances of the knapsack problem.

For simulation, the state vector backend of qiskit has been used. This simulates a noiseless quantum circuit and, in our case, returns the state vector

$$|\beta, \gamma\rangle = \prod_{j=1}^{p} U_B(\beta_j) U_C(\gamma_j) |s\rangle \tag{5.1}$$

in computational basis. From this the probability distribution

$$p(x) = |\langle x|\beta, \gamma\rangle|^2, \quad x \in Z(N) \tag{5.2}$$

of sampling from the QAOA circuit may be calculated. As described at the end of Section 3.1, this probability distribution needs to be optimized by finding good parameters β, γ. As a metric for the quality of the distribution we use the expectancy

$$\mathbb{E}_p[c] = \sum_{x \in Z(N)} p(x) c(x) \tag{5.3}$$

of the objective function c under the bitstring distribution p, with high values indicating a good distribution. We use the SHGO (Simplical Homology Global Optimization) algorithm [ESF18] implemented in SciPy [Vir+20] to find β, γ for which $\mathbb{E}_p[c]$ is maximized. We choose a general purpose global optimization algorithm as we do not know a priori how the optimization landscape is shaped and there are possibly many local maxima. The parameter space is given by the periodicity conditions for β, γ discussed in Chapter 4. The full parameter space is used for optimization.

A series of very simple knapsack problems were chosen for simulation. They are listed in Table 5.1. As the simulations were carried out with limited computational resources, only circuit sizes up to around 10 qubits were feasible for simulation and especially optimization in a reasonably short time. To respect this limit, only problem instances with at most four items were chosen. Also W and the w_j were chosen to be sufficiently small.

Name	Values	Weights	Maximum Weight
A	(1, 2)	(1, 1)	1
B	(2, 1)	(1, 1)	1
C	(1, 2)	(1, 1)	2
D	(1, 1, 2)	(1, 1, 1)	2
E	(1, 2, 4)	(1, 2, 3)	3
F	(2, 3, 5)	(2, 2, 2)	3
G	(1, 2, 1, 3)	(1, 2, 2, 1)	4

Table 5.1: The knapsack problem instances used for simulation.

In the following we will present item choices $x \in Z(N)$ not as bitstrings but as tuples, for consistency with the presentation in Table 5.1. Consider e.g. Problem A. An item choice $(1, 0)$ refers to the choice of the item with value 1 and weight 1, an item choice $(0, 1)$ refers to the choice of the item with value 2 and weight 1. This ordering is the reverse of the ordering used in qiskit.

All three approaches use different objective functions, in the case of the soft constraint approaches also depending on additional parameters. For comparing the performances of the different approaches, a common metric is necessary. We use the objective function

$$c(x) = f(x)v(x), \tag{5.4}$$

where f is a feasibility oracle as and v is the value function of the knapsack problem. This is 0 for all non-feasible choices and otherwise returns the choice's value. It does not depend on any parameters such that the solutions of all approaches are comparable. By only counting the values of the feasible choices, the average objective function value of a choice probability distribution accounts for the percentage of feasible choices, favoring choice distributions with high feasibility ratio. While this is by far not perfect, it may be seen as a useful heuristic for comparison.

For different instances of the knapsack problem, this might still not be comparable. Let an optimal solution of a knapsack problem be an $x^* \in F$ with $v(x^*) = \max_{x \in F} v(x)$. Optimal solutions of different knapsack problem instances have different values in general. A metric compensating for this is the approximation ratio. The term is used differently across the literature. Here we define it as

$$\rho = \frac{\mathbb{E}_p[c]}{v(x^*)}, \tag{5.5}$$

where x^* is an optimal solution. We use it as a measure of how close a probability distribution p is to an optimal solution. A value of $\rho = 1$ corresponds to a distribution only containing optimal solutions and a value of $\rho = 0$ corresponds to a distribution only containing trivial (no items chosen) or non-feasible solutions.

5.2 Optimized Probability Distributions

We will now present optimized probability distributions generated by the different approaches for problem A from Table 5.1 as an indication that the QAOA circuits and optimization methods behave as expected. Firstly, we present an optimized probability distribution of a quadratic penalty based approach, as described in Section 4.1. The parameters $p = 4$, $a = 1$ and $b = 10$ were chosen. This choice of a and b fulfills the condition (4.7). The optimized angles are

$$\begin{aligned} \gamma &= (47.18430682, 15.80724481, 47.11960037, 15.8987693) \\ \beta &= (1.66091219, 2.34327252, 0.92273586, 2.36687105) \end{aligned} \tag{5.6}$$

The corresponding probability distribution is shown in Figure 5.1a[1]. Secondly, we present a similar probability distribution generated by a linear penalty based approach as described in Section 4.3. The parameters $p = 2$ and $a = 10$ were chosen. This choice of a satisfies the condition (4.46). The optimized angles are

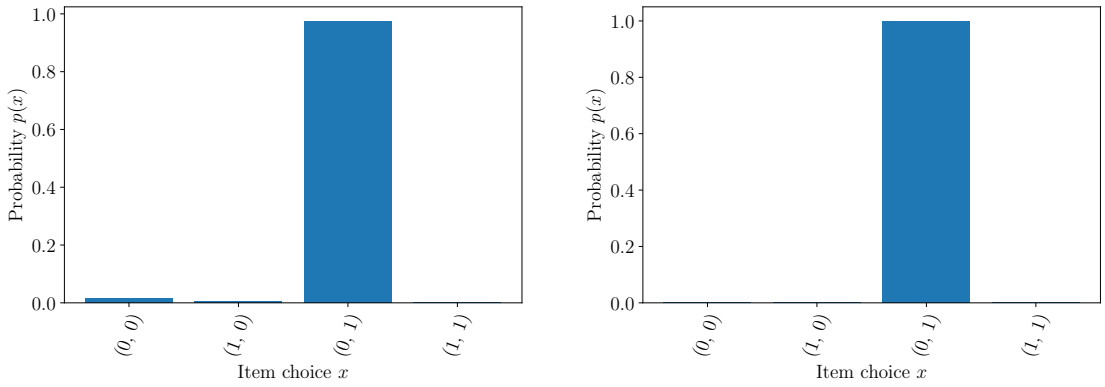
$$\begin{aligned} \gamma &= (4.18879028, 1.1087974) \\ \beta &= (1.38596931, 0.62628165) \end{aligned} \tag{5.7}$$

The resulting probability distribution is shown in Figure 5.1b. Finally, we present the probability distribution from a quantum walk mixer based approach as described in Section 4.4. The chosen parameters are $p = 2$ and $m = 3$. The optimized angles are

$$\begin{aligned} \gamma &= (1.57079633, 3.14159265) \\ \beta &= (0.93367446, 5.68411203) \end{aligned} \tag{5.8}$$

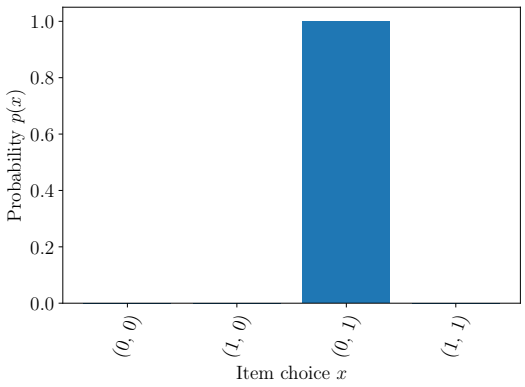
[1]These and any further plots in this chapter have been created using Matplotlib [Hun07].

The resulting distribution is shown in Figure 5.1c.



(a) Quadratic penalty based approach

(b) Linear penalty based approach



(c) Quantum Walk Mixer based approach

Figure 5.1: Probability distributions generated by the QAOA circuits of the different approaches for problem A.

All three distributions show a high probability (close to 1) of the optimal item choice $x = (0, 1)$. Also, all three distributions show an effective suppression of the non-feasible item choice $x = (1, 1)$. This indicates that the circuits and the optimization routine seem to work as expected. A slight difference can be observed when comparing the quadratic penalty approach (Figure 5.1a) with the other two approaches (Figures 5.1b and 5.1c). The quadratic penalty approach shows higher probabilities for the non-optimal choices, in particular the ground state $(0, 0)$, compared to the other approaches, even though the circuit depth of $p = 4$ is higher than the depth of $p = 2$ chosen for the other two approaches. It might be that the quadratic penalty approach requires higher values of p than the other two approaches for a comparable quality of distribution.

5.3 Problem Dependence of Distribution Quality

All three approaches yield an optimized distribution for problem A. Now we will compare how well the approaches optimize for the different problems listed in Table 5.1. As a metric for comparison the approximation ratio ρ is used. For each problem QAOA circuits for each of the three different approaches are simulated and β, γ are optimized. From the resulting probability distribution, the approximation ratio is calculated. For the quadratic penalty

based approach, $p = 4$ and $a = 1$ were chosen. b was chosen to be twice the lower bound from condition (4.7), so that its value altered from problem to problem. For the linear penalty based approach, $p = 2$ was chosen and similarly a was chosen to be twice the lower bound from condition (4.46). For the quantum walk mixer based approach, $p = 2$ and $m = 3$ were chosen. The results are shown in figure 5.2. The labels "quad", "lin", and "qw" refer to the different approaches, in the order as described above (i.e. quadratic penalty based, linear penalty based, and quantum walk mixer based approaches respectively).

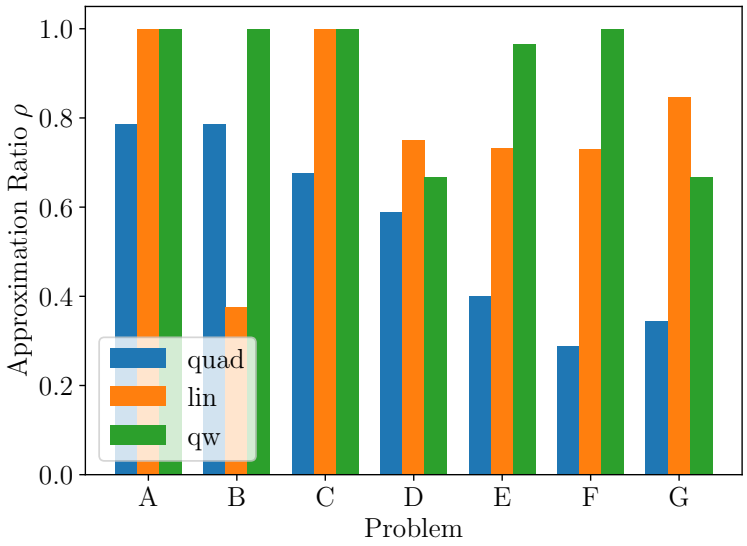


Figure 5.2: Approximation ratios for the different approaches applied to the problems from Table 5.1.

The approximation ratios of the quadratic penalty based approach lie in the range of 0.3 to 0.8. For problem A and B an approximation ratio of nearly 0.8 is reached. This is not consistent with the result in Figure 5.1a, but we were not able to identify the reason for this. For the other problems the approximation ratio decreases roughly in the order of the problems. The lowest approximation ratio is reached for problem F. The approximation ratios of the linear penalty based approach all lie in the range of 0.7 to 1, except for problem B for which the approximation ratio is close to 0.4. For problem A and C the approximation ratios are close to 1, for problems D, E, F and G the approximation ratios are close to 0.8. The quantum walk mixer based approach results in approximation ratios of either close to 1 for most of the problems or of slightly under 0.7 for problems D and G.

For all approaches the approximation ratio shows a problem dependence. It seems to be loosely linked to the item number N. Problems A, B, and C have two items, problems D, E, and F have three items, and problem G has four items. For larger item numbers the approximation ratio seems to slightly decrease, although there are some exceptions, in particular the linear penalty approach applied to problem B with $\rho = 0.4$. Furthermore this does not explain why the approximation ratio for the quantum walk approach applied to problem D is smaller than that for problem E or F.

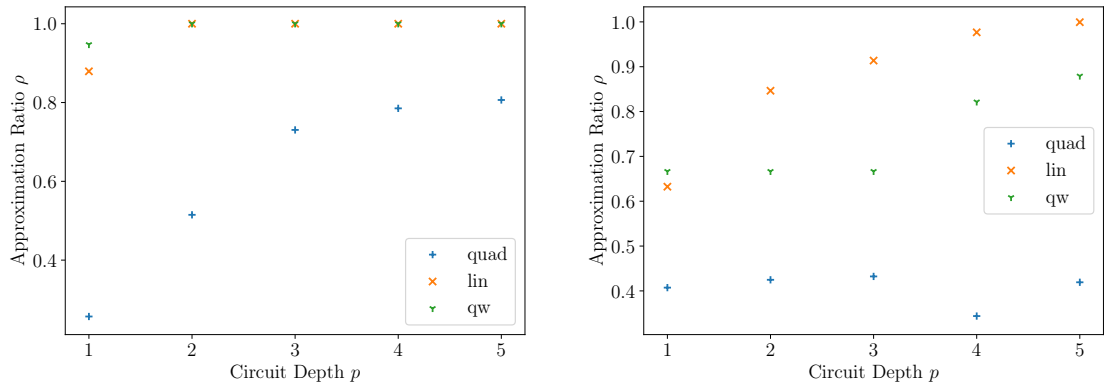
One possible explanation of the results might be that for some problems the parameters p, a, b, m were adequately chosen but for others they were not, in the sense that the optimal choice of parameters β, γ leads to a low approximation ratio. Another possible explanation might be that the parameters β, γ were not optimally chosen for some problems, i.e. the

classical optimization failed to find an optimal solution, because of the problem dependent structure of the function to be optimized. It might also be that both effects act in combination, e.g. that a bad choice of the penalty scaling factors a and b leads to the classical optimization failing.

To further investigate this behavior, we will now check the dependence of the approximation ratio on the circuit depth p (Section 5.4), the penalty scaling factors a and b (Section 5.5), and the approximation parameters m (Section 5.6).

5.4 p-Dependence of Distribution Quality

Now we investigate the dependence of the distribution quality on the circuit depth p. We use problems A and G from Table 5.1 as an example. For $p = 1, \ldots, 5$ we calculate the approximation ratios for the three different approaches. We do this by optimizing β and γ for each approach and value of p, calculating the probability distribution, and from this calculating the approximation ratios. For the quadratic penalty based approach, the chosen parameters are $a = 1$ and $b = 4$ for problem A and $a = 1$ and $b = 6$ for problem G. For the linear penalty based approach, $a = 4$ and $a = 6$ were chosen for problems A and G, respectively. For the quantum walk mixer based approach, $m = 3$ was chosen for both problems. The results are shown in Figure 5.3. Again, the labels "quad", "lin", and "qw" refer to the different approaches, in the order as described above (i.e. quadratic penalty based, linear penalty based, and quantum walk mixer based approaches respectively).



(a) Problem A

(b) Problem G

Figure 5.3: Dependence of the approximation ratio on the parameter p for the three different approaches.

For problem A, both the linear penalty based approach and the quantum walk mixer based approach give approximation ratios close to 1 for $p = 2, \ldots, 5$ and around 0.9 for $p = 1$. The quadratic penalty based approach gives approximation ratios from close to 0.3 for $p = 1$ and monotonously increasing up to 0.8 for $p = 5$. For problem G, the quadratic penalty based approach yields values of ρ close to 0.4 for $p = 1, \ldots, 5$, slightly increasing for $p \leq 3$ and reaching its maximum at $p = 3$. For the linear penalty based approach, the approximation ratio increases monotonously for $p = 1, \ldots, 5$ from slightly above 0.6 to 1. The quantum walk mixer based approach results in an approximation ratio constant at close to 0.7 for $p = 1, 2, 3$ and increasing for $p = 4, 5$ with values of slightly above 0.8 and close to 0.9, respectively. The results are consistent with those from Figure 5.2.

In general, an increase in the approximation ratio for increasing p can be observed. This is to be expected as for higher values of p the approximation of the underlying adiabatic evolution becomes better, so that the final state of the circuit becomes increasingly similar to the optimal solution. Further, the different approaches show different approximation ratios, supporting the thesis that the quadratic penalty based approach requires higher values of p than the other two approaches for the same performance. An exception of the described behavior is the quadratic penalty based approach applied to problem G. This is related to the performance of the classical optimizer.

For all three presented approaches, $U_B(0) = \mathbb{I}$ and $U_C(0) = \mathbb{I}$. Therefore a QAOA circuit with depth $p+1$ is able to produce any state of a QAOA circuit with depth p. In particular, this means that the best possible approximation ratio ρ_p^* of a depth p circuit may be also reached with a depth $p+1$ circuit. In other words, the optimal approximation ratios ρ^* of these circuits relate via

$$\rho_{p+1}^* \geq \rho_p^*. \tag{5.9}$$

Whenever the approximation ratio of a depth $p+1$ circuit is smaller than that of a depth p circuit, this allows to conclude that the approximation ratio of the depth $p+1$ circuit is not optimal.

Relating this to Figure 5.3, we find that for the quadratic penalty based approach applied to problem G the classical optimization method fails to find the optimal parameters β, γ for $p = 4$ and $p = 5$. One reason for this might be the chosen approach to optimization, combined with the relatively high dimensionality of the optimization problem (8 and 10 parameters, respectively). Possibly other, more suited optimization methods might lead to better approximation ratios for the same circuits. Another important aspect is the exact relation of the expectation value of the objective function to β, γ. A relation with many local maxima, for example, might hinder the optimization algorithm to succeed in finding a global maximum. Finally, the choice of penalty scaling factors might influence the ability of the optimization algorithm to find a global optimum.

5.5 Dependence on Penalty Scaling Factors

In addition to the parameters β, γ, the soft constraint based approaches from Sections 4.1 and 4.3 also have parameters controlling the scaling of the penalty functions. We will now discuss how the values of these parameters influence the quality of the resulting distributions. Again, we take the approximation ratio as a metric for distribution quality and we choose problems from Table 5.1 as examples.

First, we discuss the quadratic penalty based approach applied to problem A. The scaling of the value and the penalty in the objective function are controlled by the parameters a and b, respectively. For simplification we set $a = 1$ and only vary the parameter b in the range from 0 to 10 in steps of size 0.2. For each pair a, b the parameters β, γ are optimized and the approximation ratio is calculated from the resulting probability distribution. The results for $p = 4$ are shown in Figure 5.4.

The values are broadly scattered in the range 0 to 1, with most values close to 1 for $b > 1$ and close to 0 for $b \leq 1$. There are some deviations from this, the largest occurring for $b = 3, 6, 7$ with $\rho \leq 0.6$. There is no clearly discernible pattern of these deviations. One possible explanation is that a particular choice of b might change the behavior of the circuit in a way that the optimal distribution becomes worse. Another explanation might be that

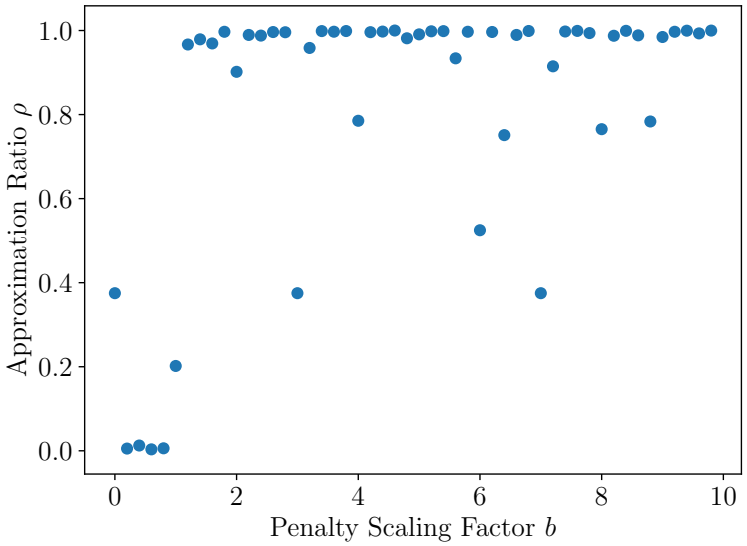
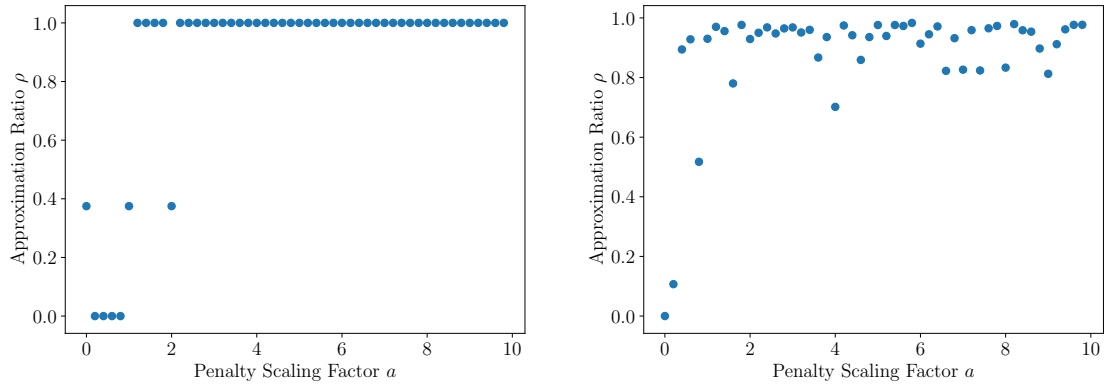


Figure 5.4: Dependence of the approximation ratio on the value of b for a quadratic penalty based approach applied to problem A.

the optimization landscape is changed in a way that the classical optimizer fails to find the global optimum. It is not clear which deviations may be attributed to which cause.

Now we discuss the linear penalty based approach applied to problems A and G. The scaling of the penalty in this approach is controlled by a parameter a. To explore the dependence on this parameter we choose $p = 3$ and optimize β, γ for the resulting circuit and a in the range of 0 to 10 in steps of size 0.2. We calculate the corresponding probability distributions and their approximation ratios. The results are shown in Figure 5.5.



(a) Problem A

(b) Problem G

Figure 5.5: Dependence of the approximation ratio on the value of a for a linear penalty approach.

For problem A, $\rho \approx 1$ for $a > 1$ and $\rho \approx 0$ for $a < 1$. The three exceptions are $a = 0, 1, 2$, for which $\rho \approx 0.4$. For problem G, ρ fluctuates between roughly 0.8 and 1 for $a \geq 1$ with deviations of $\rho \leq 0.6$ for $a = 0, 0.2, 0.8$. No discernible pattern is recognizable in the fluctuations.

For problem A, the global optimum seems to be achieved reliably. For problem G the fluctuations of ρ for small changes in a presumably indicate that this is not the case. This suggests that the choice of problem significantly changes the optimization landscape and thereby the ability of the optimization algorithm to find an optimal solution. Similar to the quadratic penalty based approach, it remains unclear for both problems A and G what the exact cause of the deviations is.

5.6 m-Dependence of Distribution Quality

Finally, we explore the dependence of the quantum walk mixer based approach on the parameter m. As before, we choose $p = 2$. For all problems from Table 5.1 and values $m = 1, 2, 3, 10, 20$, the approximation ratio of the optimized probability distributions is calculated. The results are shown in Figure 5.6.

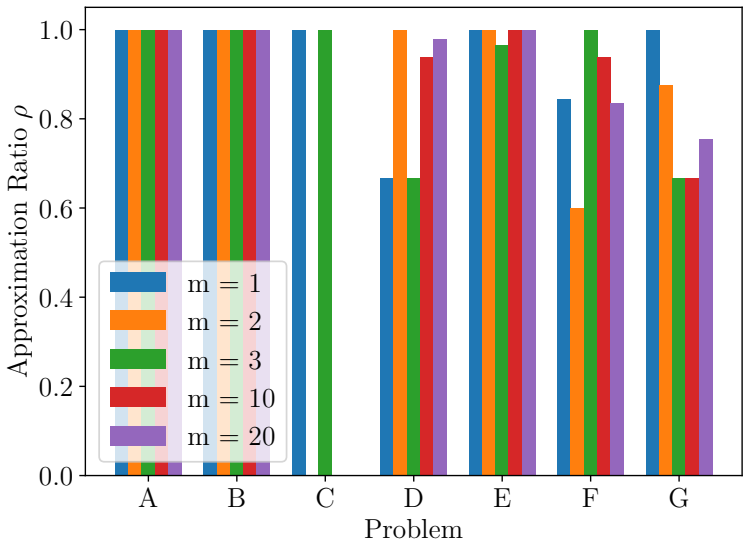


Figure 5.6: Dependence of the approximation ratio on the value of m for a quantum walk mixer based approach applied to the problems from Table 5.1.

For most problems and most values of m, approximation ratios of nearly 1 are reached. This is particularly true for problems A, B, and E, for which all values of m result in an approximation ratio close to 1. Deviations occur for problems C, D, F, and G. For problem C, $m = 1$ and $m = 3$ result in $\rho \approx 1$ whereas the other choices of m lead to $\rho \approx 0$. For problem D, $m = 1$ and $m = 3$ lead to $\rho \approx 0.7$ and the other choices lead to $\rho \approx 1$. For problem F, the largest deviation is $\rho \approx 0.6$ for $m = 2$. The other values of m lead to $\rho \geq 0.8$, with the maximum of $\rho = 1$ reached for $m = 3$. For problem G, $\rho = 1$ for $m = 1$, $\rho \approx 0.9$ for $m = 2$, and $\rho < 0.8$ for the other values of m. These results are consistent with Figure 5.2.

Overall, no clear pattern for a dependence on m is discernible. In the case of problem C, it might be that only odd values of n allow any optimization due to the structure of the problem specific circuit. Similarly, optimization for problem D might require even values of m. This remains speculation as the results are inconclusive. It cannot be distinguished whether the deviations are to be attributed to the problem dependence of the optimization

landscape and combined with possible shortcomings of the classical optimizer or whether they are a result of the value of m.

5.7 Implications

Considering the results presented in Sections 5.3 - 5.6, a few things are notable. The quadratic penalty based approach shows a significantly worse performance compared to the other two approaches. This makes sense as there are more qubits whose state needs to be optimized and there are fewer constraints on the possible qubit states. In particular, we observe the classical optimization method failing for problem G and higher values of p. This makes the feasibility of the presented approach questionable for larger/more complicated problems, as we would expect the classical optimization problem to become more complicated. Nonetheless, these results have to be considered carefully as the parameters a, b were chosen rather arbitrarily and although no obvious relation between the choice of these parameters and the performance were found, we observed that the choice of parameters can have a large impact on the resulting approximation ratios. A similar case can be made for the choice of m, so that the results of these simulations remain inconclusive to large parts.

Chapter 6

Conclusion and Outlook

In this thesis we explored how QAOA might be applied to the knapsack problem. After giving an overview of the knapsack problem, we presented the conceptual framework of QAOA and motivated it by its connection to quantum computing by adiabatic evolution. We discussed how QAOA may be used for constrained optimization problems in general. Here we focused on two approaches, namely converting the hard constraints to a penalty on the one hand and strictly enforcing the constraints using the mixing operator on the other hand. We presented three ways of applying these general approaches to the knapsack problem. Firstly, we presented a soft constraint based approach with a quadratically scaling penalty as it may be found in the literature. Secondly, we presented an improved version of a soft constraint based approach with linearly scaling penalty. For this we introduced the notion of a feasibility oracle and presented an implementation for the knapsack problem. Finally, we applied a mixer based approach to enforcing hard constraints which we improved and applied to the knapsack problem, also based on the feasibility oracle.

We explored the behavior of these approaches numerically for small instances of the knapsack problem. For this we implemented and simulated them in Qiskit. We presented results showing the performance of the different approaches in dependence of different parameters and problem instances. While giving a first insight into the behavior of the different approaches, the results remain inconclusive in many ways.

Due to the limited scope of this thesis we were not able to discuss other approaches like the heuristic mixers used in the quantum alternating operator ansatz. Furthermore we were not able to explore the dependence of these approaches on noise, both theoretically and numerically. Together with a discussion of classical optimization methods and an analysis of the optimization landscape, these might be interesting topics for further research.

Appendix A

Addition Using the Quantum Fourier Transform

In this appendix, we will describe the addition algorithm used in the implementation of the feasibility oracle for the knapsack problem (Section 4.2). It has first been described in [Dra00], to which we refer for a more detailed discussion. Here we will focus only on the aspects necessary for this thesis, with our description heavily borrowing from the original paper.

Consider an n-qubit state $|a\rangle$, $a \in \mathbb{Z} \mod 2^n$, and a whole number $b \in \mathbb{Z} \mod 2^n$. The objective is to find an algorithm mapping

$$|a\rangle \mapsto |a+b\rangle\,, \tag{A.1}$$

as well as a controlled variant acting as

$$\begin{aligned} |a\rangle \otimes |0\rangle &\mapsto |a\rangle \\ |a\rangle \otimes |1\rangle &\mapsto |a+b\rangle\,, \end{aligned} \tag{A.2}$$

as these are necessary for the implementation of the feasibility oracle.

The central idea of this approach is applying a quantum Fourier transform to the initial state and carrying out the addition by applying (controlled) phase gates to the transformed state. We define the quantum Fourier transform to be

$$\mathcal{F}\,|a\rangle = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e(ak/2^n)\,|k\rangle\,, \tag{A.3}$$

using the notation $e(t) = \exp(2\pi i t)$. Further, for $l \in \mathbb{N}$, let

$$|\phi_l(a)\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e\Big(a/2^l\Big)\,|1\rangle\right). \tag{A.4}$$

Using the binary representation $k_{n-1} \dots k_0$ of $k \in \mathbb{Z} \mod 2^n$, we may rewrite

$$\begin{aligned} |\phi_1(a)\rangle \otimes \cdots \otimes |\phi_n(a)\rangle &= \frac{1}{2^{n/2}} \left(\left(|0\rangle + e(a/2^1) |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e(a/2^n) |a/2^n\rangle \right) \right) \\ &= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} \left(\prod_{l=0}^{n-1} k_l e(a/2^{n-l}) \right) |k\rangle \\ &= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e(ak/2^n) |k\rangle \\ &= \mathcal{F} |a\rangle . \end{aligned} \tag{A.5}$$

As may easily be checked, the quantum circuit in Figure A.1 implements $\mathcal{F}$ up to the ordering of qubits. Here we choose H to denote the Hadamard gate and

$$\Phi(k) = |0\rangle\langle 0| + e(1/2^k) |1\rangle\langle 1| \tag{A.6}$$

as an abbreviating notation for phase gates with particular phase values.

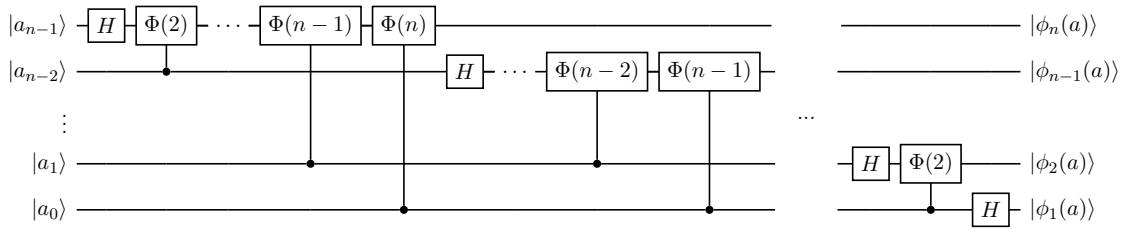


Figure A.1: Circuit implementing the quantum Fourier transform up to ordering of qubits.

Addition may be implemented using phase gates

$$P(x) = |0\rangle\langle 0| + e(x) |1\rangle\langle 1| . \tag{A.7}$$

We find that

$$P(b/2^l) |\phi_l(a)\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e((a+b)/2^l) |1\rangle \right) = |\phi_l(a+b)\rangle . \tag{A.8}$$

That is, by application of a phase gate we may map $|\phi_l(a)\rangle \mapsto |\phi_l(a+b)\rangle$. By doing this for every qubit, we get the transformation A_b, acting as

$$\begin{aligned} A_b \mathcal{F} |a\rangle &= A_b |\phi_1(a)\rangle \otimes \cdots \otimes |\phi_n(a)\rangle \\ &= |\phi_1(a+b)\rangle \otimes \cdots \otimes |\phi_n(a+b)\rangle \\ &= \mathcal{F} |a+b\rangle . \end{aligned} \tag{A.9}$$

By applying an inverse quantum Fourier transform to this state, we get

$$\mathcal{F}^{-1} A_b \mathcal{F} |a\rangle = \mathcal{F}^{-1} \mathcal{F} |a+b\rangle = |a+b\rangle . \tag{A.10}$$

This implements the intended addition using $\mathcal{O}(n^2)$ gates for the quantum Fourier transforms and $\mathcal{O}(n)$ gates for each addition, and without the use of ancillary qubits. The controlled addition may be defined by using controlled phase gates instead of phase gates in the definition of A_b. In Figure 4.2 we chose to denote $\mathcal{F}$ as "QFT" and A_b as "Add b" for better readibility.

Bibliography

[Ani+21] M. D. Sajid Anis et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[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. DOI: 10.1007/BF01011339.

[Bin22] Lennart Binkowski. "Constraint Graph Model Analysis of the Quantum Alternating Operator Ansatz". Master's Thesis. Leibniz University Hanover, 2022.

[Coo00] Stephen Cook. *The P vs. NP Problem*. Apr. 2000. URL: https://www.claymath.org/sites/default/files/pvsnp.pdf (visited on 11/03/2022).

[Deu85] David Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (July 8, 1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070.

[Dra00] Thomas G. Draper. *Addition on a Quantum Computer*. Aug. 7, 2000. arXiv: quant-ph/0008033.

[ESF18] Stefan C. Endres, Carl Sandrock, and Walter W. Focke. "A simplicial homology algorithm for Lipschitz optimisation". In: *Journal of Global Optimization* 72.2 (Oct. 2018), pp. 181–217. DOI: 10.1007/s10898-018-0645-y.

[Far+00] Edward Farhi et al. *Quantum Computation by Adiabatic Evolution*. Jan. 28, 2000. arXiv: quant-ph/0001106.

[Fey81] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (May 7, 1981), pp. 467–488. DOI: 10.1007/BF02650179.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. Nov. 14, 2014. arXiv: 1411.4028.

[Gen21] Edd Gent. "IBM's 127-Qubit Eagle Is the Biggest Quantum Computer Yet". In: (Nov. 22, 2021). URL: https://singularityhub.com/2021/11/22/ibms-127-qubit-eagle-is-the-biggest-quantum-computer-yet/ (visited on 10/13/2022).

[GH19] Pierre Dupuy de la Grand'rive and Jean-Francois Hullo. *Knapsack Problem variants of QAOA for battery revenue optimisation*. Aug. 15, 2019. arXiv: 1908.02210.

[Gro96] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. ACM Press, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[Had+19] Stuart Hadfield et al. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz". In: *Algorithms* 12.2 (Feb. 12, 2019), p. 34. DOI: 10.3390/a12020034.

[Hol22] Jan Rasmus Holst. *Code for the Bachelor's Thesis "QAOA for the Knapsack Problem"*. Oct. 2022. URL: https://github.com/rhlst/qaoa-for-the-knapsack-problem.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[IBM22] IBM. *Our new 2022 Development Roadmap*. 2022. URL: https://www.ibm.com/quantum/roadmap (visited on 10/13/2022).

[JLD22] Jorik Jooken, Pieter Leyman, and Patrick De Causmaecker. "A new class of hard problem instances for the 0–1 knapsack problem". In: *European Journal of Operational Research* 301.3 (Sept. 2022), pp. 841–854. DOI: 10.1016/j.ejor.2021.12.009.

[Kay19] Alastair Kay. *Tutorial on the Quantikz Package*. 2019. arXiv: 1809.03842.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 2004. DOI: 10.1007/978-3-540-24777-7.

[Luc14] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2.5 (2014). DOI: 10.3389/fphy.2014.00005.

[MPT00] Silvano Martello, David Pisinger, and Paolo Toth. "New trends in exact algorithms for the 0–1 knapsack problem". In: *European Journal of Operational Research* 123.2 (June 2000), pp. 325–332. DOI: 10.1016/S0377-2217(99)00260-X.

[MPT99] Silvano Martello, David Pisinger, and Paolo Toth. "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem". In: *Management Science* 45.3 (Mar. 1999), pp. 414–424. DOI: 10.1287/mnsc.45.3.414.

[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), p. 61. DOI: 10.1007/s11128-019-2171-3.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th Anniversary edition. Cambridge University Press, 2010. ISBN: 978-1-107-00217-3.

[Pis05] David Pisinger. "Where are the hard knapsack problems?" In: *Computers & Operations Research* 32.9 (Sept. 2005), pp. 2271–2284. DOI: 10.1016/j.cor.2004.03.002.

[Pre18] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 6, 2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[Roc+20] Christoph Roch et al. *Cross Entropy Hyperparameter Optimization for Constrained Problem Hamiltonians Applied to QAOA*. Aug. 20, 2020. arXiv: 2003.05292v2.

[Sho94] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[SP22] Adenilton J. da Silva and Daniel K. Park. *Linear-depth quantum circuits for multi-qubit controlled gates*. Mar. 22, 2022. arXiv: 2203.11882.

[Tur21] Jordi Tura. "Boosting simulation of quantum computers". In: *Nature Computational Science* 1.10 (Oct. 2021), pp. 638–639. DOI: 10.1038/s43588-021-00145-5.

[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.

Acknowledgements

Writing this thesis would not have been possible without the kind support of the following people. I would like to thank Prof. Osborne for the opportunity to write this thesis in his research group. I would like to thank Marvin Schwiering, Tim Heine, Martin Steinbach, Lennart Binkowski, and Andreea Lefterovici for diligent proofreading and many helpful discussions. And finally, I would like to thank my parents Corinna Albrecht and Ulrich Holst, as well as my partner Marte Henningsen, for constant encouragement and emotional support during the process of writing this thesis.