Leibniz Universität Hannover
Institut für Theoretische Physik

Quantenalgorithmen für das Travelling Salesman Problem

vorgelegt von

Lara Naomi Lelakowski

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

Abgabedatum:	29. Juli 2022
Studiengang:	Fächerübergreifender Bachelor Physik Mathe
Matrikelnummer:	10029578
Erstgutachter:	Prof. Dr. Tobias J. Osborne
Zweitgutachter:	Sören Wilkening

Zusammenfassung

Quantencomputer sind ein zentraler Bestandteil der Forschung an zukünftigen Technologien. Sie führen durch die Gesetze der Quantenmechanik zu schnelleren Laufzeiten und sind leistungsstärker als klassische Computer [1]. Dies führt dazu, dass auf Quantencomputern **NP**-schwere Probleme effizienter gelöst werden können, als es auf klassischen Computern möglich ist. Bei einem der **NP**-schweren Probleme handelt es sich um das Travelling Salesman Problem. Ambainis et al. haben im Jahr 2018 einen Quantenalgorithmus auf Basis der dynamischen Programmierung entwickelt, der im Vergleich zur klassischen dynamischen Programmierung zu einem Speedup von $\mathcal{O}^*(2^n)$ zu $\mathcal{O}^*(1.7274^n)$ in der Laufzeit zur Lösung des Travelling Salesman Problems führen soll [2]. Ziel der vorliegenden Bachelorarbeit ist es, die Grundlagen für den Quantenalgorithmus von Ambainis et al. zu erläutern, den Quantenalgorithmus zu beschreiben und für den Quantenalgorithmus benötigte Quantenschaltkreise aufzuzeigen. Bei der Beschreibung ergibt sich, dass der Quantenalgorithmus in der von Ambainis et al. beschriebenen Form einige Schritte vernachlässigt, da sie in der Laufzeitordnung vernachlässigbar sind. Aufgrund dessen wird der Algorithmus in dieser Arbeit angepasst, sodass ein vollständiger Quantenalgorithmus vorgestellt werden kann.

Inhaltsverzeichnis

Abkürzungsverzeichnis

TSP ..... Travelling Salesman Problem
Qubit ..... Quantenbit
ASCII ..... American Standard Code for Information Interchange
QAOA ..... Quantum Approximate Optimization Algorithmus
XOR ..... Exklusive-OR
Q-NOT ..... Quanten-NOT
CNOT ..... Controlled-NOT
CZ ..... Controlled-Z
RAM ..... random access memory
qRAM ..... quantum random access memory
QFT-Addierer ..... Quantum Fourier Transform-Addierer

1 Kapitel 1. Einleitung

In der heutigen Zeit sind Computer sehr weit verbreitet und werden sowohl beruflich wie auch privat in vielen Bereichen angewandt. Dabei wird stetig daran geforscht, leistungsstärkere und schnellere Computer auf den Markt zu bringen. Hierbei gibt das Moore´sche Gesetz einen Überblick über die Leistung von Computern. Dieses Gesetz besagt, dass sich die Leistung von Computern in der Forschung etwa alle zwei Jahre verdoppelt, wobei die Kosten etwa konstant bleiben [1]. Es wird allerdings erwartet, dass dieses Verhalten über lange Zeit nicht aufrecht erhalten werden kann [1]. Integrierte Chips können nämlich nicht immer kleiner gebaut werden, da die Strukturen durch die Größe von Atomen beschränkt sind [3].

Um weiterhin immer leistungsstärkere Computer zu entwickeln, kann beispielsweise eine neue Art von Computern betrachtet werden. Eine mögliche Art bilden die Quantencomputer. Diese basieren auf den Grundsätzen der Quantenmechanik und arbeiten mit Qubits statt Bits [1, 3]. In Kapitel 2 wird beschrieben, dass ein Hinzufügen von nur einem Qubit zu einem Quantencomputer dessen Leistung verdoppelt. Außerdem können Quantencomputer durch die Verwendung der Möglichkeit zur Superposition von quantenmechanischen Zuständen einen Speedup hinsichtlich der Lösung von **NP**-schweren Problemen mit sich bringen. Dies wird in Kapitel 2 und 3 beschrieben.

Bei einem der **NP**-schweren Probleme handelt es sich um das Travelling Salesman Problem. Dieses Problem befasst sich damit, die kürzeste Rundreise über eine bestimmte Anzahl an Städten durchzuführen. Zur Lösung dieses Problems existieren verschiedene Algorithmen. Dabei ist der schnellste klassische Algorithmus von Held und Karp und führt zu einer Laufzeit von $\mathcal{O}^*(2^n)$ [4]. Die Bedeutung von $\mathcal{O}^*$ wird in Abschnitt 3.1 beschrieben. Quantenalgorithmen können diese Laufzeit senken. Es gibt verschiedene Ansätze für solche Algorithmen. Einen ersten Ansatz bildet der QAOA, also der „**Q**uantum **A**pproximate **O**ptimization **A**lgorithmus“ [5]. Ob ein Speedup mit diesem erreichbar ist, ist jedoch unklar. Einen zweiten Lösungsansatz bildet der „Branch and Bound“-Algorithmus [6]. Auf diesem Algorithmus baut der Concorde-Solver auf [7]. Dabei handelt es sich um einen Onlinerechner mit einer der zur Zeit höchsten existierenden Rechenleistung, der das Travelling Salesman Problem löst [7]. Mit dem Ansatz über den „Branch and Bound“-Algorithmus wird sich in der folgenden Arbeit nicht beschäftigt. Stattdessen wird der Quantenalgorithmus von Ambainis et al. vorgestellt, der auf dem Algorithmus von Held und Karp aufbaut und eine Laufzeit von $\mathcal{O}^*(1.727391^n)$ erreicht [2].

Um den Quantenalgorithmus von Ambainis et al. ausführlich betrachten zu können, wird in Kapitel 2 auf die Grundlagen eingegangen, die zur Betrachtung von Quantencomputern und Quantenalgorithmen benötigt werden. Dabei wird vorausgesetzt, dass die Grundlagen der Quantenmechanik und somit auch Effekte wie die Superposition und Verschränkung bekannt sind. In Kapitel 3 werden das Travelling Salesman Problem und

die für dessen Lösung benötigten Quantenalgorithmen vorgestellt. Im Anschluss werden in Kapitel 4 der klassische Algorithmus und der Quantenalgorithmus für das Problem beschrieben. Außerdem wird in diesem Kapitel dargelegt, wie der Quantenalgorithmus angepasst werden kann und wieso er in der beschriebenen Form die beste Laufzeit erreicht. Zum Schluss werden in Kapitel 5 einige Quantenschaltkreise vorgestellt, die für die Implementierung des Quantenalgorithmus benötigt werden.

Kapitel 2.
2 Theoretische Grundlagen

Ein Quantencomputer arbeitet mit Qubits als kleinstmögliche Speichereinheit [1]. Damit dies funktioniert, werden Schaltkreise benötigt. In diese werden Quantengatter eingebaut. Das Prinzip der Qubits und der Quantengatter soll in diesem Kapitel erläutert werden. Dazu wird in Abschnitt 2.1 beschrieben, was Qubits sind, wie sie sich gegenüber klassischen Bits unterscheiden und welche Eigenschaften sie besitzen. In Abschnitt 2.2 wird dann darauf eingegangen, wie mit den Qubits in Schaltkreisen gearbeitet wird. Dazu werden zunächst einige klassische Gatter vorgestellt, um darauf aufbauend die Quantengatter zu beschreiben und ihre Funktion zu erläutern.

2.1. Qubits

In einem klassischen Computer werden Informationen in Bits dargestellt [8]. Die Bits können hierbei zwei verschiedene Zustände annehmen, die als „0“ und „1“ definiert werden [8]. Durch den Binärcode wird festgelegt, wie man eine Ansammlung der Zustände „0“ und „1“ in Zahlen des Dezimalsystems oder in Buchstaben übersetzen kann [9]. Jedes klassische Bit nimmt somit einen bestimmten Wert an, entweder eine Null oder eine Eins. Quantencomputer unterscheiden sich von klassischen Computern dadurch, dass sich die Bits quantenmechanisch verhalten. Aufgrund dessen werden die Bits Quantenbits, kurz Qubits, genannt [1]. Das Besondere dieser Qubits ist, dass eine Superposition der Zustände möglich ist [8]. Wird ein Zustand, der in Superposition vorliegt, allerdings gemessen, so nimmt das Qubit bei der Messung einen bestimmten Zustand an. Dies entspricht dem Messpostulat der Quantenmechanik [1].

Die Superposition der quantenmechanischen Zustände bewirkt, dass eine bestimmte Anzahl an Daten auf weniger Qubits als auf klassischen Bits gespeichert werden kann. Die Qubits können nämlich mehrere Zustände zeitgleich annehmen und somit auch mehrere Daten zeitgleich speichern [1]. Dies führt zu der Hoffnung der aktuellen Forschung, dass eine geringere Anzahl an Qubits zu einer höheren Leistung im Vergleich zu den bisher genutzten Bits führen kann [10].

2.1.1. Speicherung in Bits

Wie bereits erwähnt, arbeiten klassische Computer mit Bits, welche jeweils die Zustände „0“ und „1“ annehmen können [8]. Aus diesen Zuständen setzen sich durch den Binärcode definierte Zahlen und Buchstaben zusammen. So entspricht die Zahl „20“ im Binärcode der „00010100“, der Buchstabe A wird als „01000001“ definiert und a schreibt man als „01100001“ [11]. In diesem Beispiel wurden acht Bits verwendet. Da jedes Bit zwei

verschiedene Zustände annehmen kann, entsteht ein Code, der $2^8 = 256$ verschiedene Zeichen darstellen kann. Dieser Code ist auch als der ASCII, also der **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, bekannt [11]. In einige Codierungen fällt die erste Ziffer des Codes weg, da der ASCII zunächst durch sieben Bits definiert wurde und das achte Bit erst später hinzugefügt wurde [12]. Im Folgenden wird nur die Codierung der Zahlen betrachtet. Dies geschieht durch das Binärsystem. Das Binärsystem unterscheidet sich gegenüber dem ASCII dadurch, dass es nur die Darstellung von Zahlen und nicht die von Buchstaben oder weiteren Zeichen beschreibt [9].

Man betrachte nun die Zahlen 0-3. Um eine dieser Zahlen darzustellen, braucht man $log_2(4) = 2$ Bits. Es entsteht die folgende Codierung der Zahlen:

$$0\widehat{=}00; \quad 1\widehat{=}01; \quad 2\widehat{=}10; \quad 3\widehat{=}11$$

Um allerdings alle vier Zahlen simultan darzustellen, benötigt man $4 \cdot log_2(4) = 8$ Bits. Allgemeiner gefasst kann man mit n Bits $2^n = N$ verschiedene Zahlen beschreiben. Um all diese Zahlen zeitgleich abzurufen oder zu speichern, benötigt man $n \cdot N = n \cdot 2^n$ Bits. Möchte man doppelt so viele Informationen auf einem klassischen Computer speichern, wird auch die doppelte Anzahl an Bits benötigt [1].

Mit Zahlen, die in der Binärdarstellung dargestellt werden, lassen sich des Weiteren mathematische Operationen ausführen. So kann man durch bestimmte Schaltkreise eine Addition, Multiplikation und viele weitere mathematische Operationen durchführen. Aufwendigere Algorithmen können auch Probleme wie das Travelling Salesman Problem lösen. Hierbei wird die kürzeste Rundreise durch n Städte gesucht, wobei jede Stadt genau einmal besucht wird [4]. Um dieses Problem zu lösen, benötigt der schnellste klassische Algorithmus, die dynamische Programmierung, eine Laufzeit von $\mathcal{O}(n^2 2^n)$ [2]. An dieser Stelle setzen die Quantencomputer an. Diese können durch geeignete Algorithmen zu einer exponentiell schnelleren Suche nach dem kürzesten Weg führen. Ein Ansatz mit der dynamischen Programmierung führt zu einer Laufzeit von $\mathcal{O}^*(1.728^n)$ [2]. Verantwortlich für diese Laufzeit sind die Qubits. Ihre Eigenschaften werden im Folgenden beschrieben.

2.1.2. Eigenschaften der Qubits

Qubits können ebenso wie die Bits Zustände annehmen. Die Zustände, die bei den Bits als „0“ und „1“ benannt wurden, sind bei den Qubits „$|0\rangle$“ und „$|1\rangle$“ [9]. Qubits bringen im Vergleich zu den Bits aber den Vorteil der Superposition mit sich. Ein Qubit kann nicht nur einen bestimmten Zustand, sondern auch eine Überlagerung mehrerer Zustände annehmen. So ist der Zustand eines Qubits gegeben durch [4]:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

Man kann die Zustände $|0\rangle$ und $|1\rangle$ auch in Vektorschreibweise darstellen [9]. Dabei sehen die Zustände wie folgt aus:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

$$\Rightarrow |\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.3}$$

Die Variablen α und β sind komplexe Zahlen und geben die Amplituden der Zustände an [9]. Dabei entsprechen ihre Quadrate der Wahrscheinlichkeit, dass der jeweilige Zustand bei einer Messung gemessen wird [9]. Das liegt daran, dass bei einer Zustandsmessung der Qubits die Superposition der Zustände nicht gemessen werden kann. Stattdessen wird der Zustand $|0\rangle$ mit einer Wahrscheinlichkeit von $|\alpha|^2$ und der Zustand $|1\rangle$ mit einer Wahrscheinlichkeit von $|\beta|^2$ gemessen [1]. Durch die Normierungsbedingung ist die Summe der Wahrscheinlichkeiten immer 1, also gilt $|\alpha|^2 + |\beta|^2 = 1$ [1]. Ein Qubit kann folglich mehrere Zustände zeitgleich annehmen. Möchte man nun die Zahlen 0 und 1 speichern, so bräuchte man auf einem klassischen Computer zwei Bits, welche die zugehörigen Zustände separat speichern. Auf einem Quantencomputer reicht lediglich ein Qubit, das sich beispielsweise im Zustand $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ befindet [1]. Ebenso funktioniert dies mit zwei Qubits, die eine Überlagerung der Zustände $|00\rangle$, $|01\rangle$, $|10\rangle$ und $|11\rangle$ bilden können. Hierfür werden also im quantenmechanischen Fall 2 Qubits und im klassischen Fall, wie in Abschnitt 2.1.1 beschrieben, 8 Bits benötigt. Die Qubits schaffen es somit, die vorher benötigte Anzahl an $n \cdot 2^n$ Bits auf n Qubits zu reduzieren. Das führt dazu, dass das Hinzufügen von einem weiteren Qubit zu einem System mit n Qubits die Leistung verdoppelt. Dies lässt sich daraus herleiten, da durch $n+1$ Qubits $N = 2^{n+1} = 2 \cdot 2^n$ Informationen dargestellt werden können. Um den riesigen Vorteil der Qubits aufzuzeigen, kann man ein System mit 500 Qubits betrachten. Mit diesen 500 Qubits kann man ein System der Form $|x_{499} \dots x_1 x_0\rangle$ mit $2^{500} \approx 10^{150}$ Amplituden definieren, was mehr Amplituden sind als Atome im Universum [1].

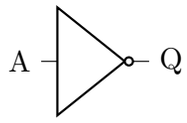
2.2. Gatter

Damit ein Computer funktioniert und die Bits bestimmte Prozesse durchführen, müssen Schaltkreise gebaut werden. Diese Schaltkreise basieren auf einer Anordnung an Gattern. Tritt ein Eingangssignal in ein Gatter ein, wird dieses moduliert [1]. Somit lässt sich abhängig vom einkommenden Signal das Ausgangssignal verändern. In einem klassischen Schaltkreis gibt es beispielsweise das NOT-Gatter, das den Zustand eines Bits umdreht [1]. Es findet also der Wechsel von dem Bitzustand 0 zu 1 oder von der 1 zur 0 statt. Auch in Quantencomputern existieren sogenannte Quantenschaltkreise. Die bekanntesten Gatter, die hierbei angewandt werden, sind das Pauli-X-Gatter, das Pauli-Z-Gatter und das Hadamard-Gatter [1]. Die Gatter werden im Folgenden genauer beschrieben.

2.2.1. Klassische Gatter

In einen klassischen Schaltkreis können n Eingangsbits injiziert werden. Abhängig von den Zuständen dieser und den Gattern im Schaltkreis werden die Zustände der Eingangsbits verändert. Als Output erhält man m Ausgangsbits. Der Wechsel von n Eingangszuständen zu m Ausgangszuständen wird durch die Gatter verursacht [1]. Da in klassischen Schaltkreisen die Bits eindeutig den Zuständen 0 und 1 zugeordnet werden können, können im Folgenden einige Wahrheitstabellen der bekanntesten klassischen Gatter dargestellt werden. Die Funktion dieser klassischen Gatter wird dabei ebenfalls beschrieben. Die Funktion und das Aussehen der Gatter wurden aus dem Buch von Nielsen und Chuang übernommen [1]. Die Gatter werden von links nach rechts gelesen. Das bedeutet, dass links der Input und rechts der Output dargestellt ist.

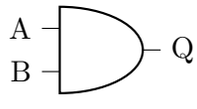
Das NOT-Gatter (Abb. 2.1) wurde in Abschnitt 2.2 bereits beschrieben. Es ändert den Wahrheitswert eines Bits von 0 zu 1 oder von 1 zu 0, abhängig vom Wert des Eingangsbits. Diese Operation findet auf nur einem Bit statt und hat somit ein Eingangs- und ein Ausgangsbit. Es handelt sich hierbei um das einzige nicht-triviale Gatter, das auf einem einzelnen Bit operiert [1].



A	Q
0	1
1	0

Abbildung 2.1.: Links abgebildet ist das NOT-Gatter. A stellt den Zustand des Eingangsbits und Q den des Ausgangsbits dar. Wie das NOT-Gatter den Zustand ändert, zeigt die Wahrheitstabelle rechts.

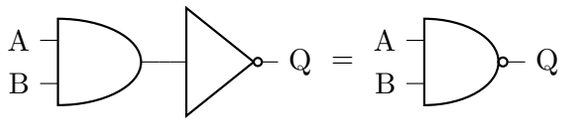
Neben den Einbit-Gattern gibt es einige Gatter, die zwei Eingangsbits und ein Ausgangsbit besitzen. Eines davon ist das AND-Gatter (vgl. Abb. 2.2). Werden in dieses Gatter zwei Eingangsbits im Zustand 1 injiziert, so erhält man als Output ebenfalls den Zustand 1. Entspricht mindestens einer der Zustände der Eingangsbits allerdings einer 0, so erhält man auch als Output den Zustand 0.



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Abbildung 2.2.: Links abgebildet ist das AND-Gatter mit den Zuständen A und B der Eingangsbits. Das Gatter gibt Q als Zustand des Ausgangsbits aus. Welchen Zustand Q annimmt, ist in der Wahrheitstabelle rechts dargestellt.

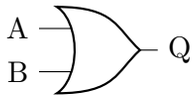
Außerdem ist eine Kombination der beiden eben beschriebenen Gatter möglich. Schaltet man das AND- und das NOT-Gatter hintereinander, so erhält man das NAND-Gatter. Dieses gibt den Zustand 0 aus, falls beide Eingangsbits im Zustand 1 vorliegen, sonst gibt es den Zustand 1 aus. Das Schaltsymbol und die Wahrheitstabelle sind in Abb. 2.3 dargestellt.



A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

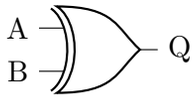
Abbildung 2.3.: Wie links dargestellt ergibt sich aus dem Hintereinanderschalten von AND- und NOT-Gatter das NAND-Gatter. Die Eingangsbits befinden sich in den Zuständen A und B. Welchen Zustand das Ausgangsbit Q annimmt, ist in der Wahrheitstabelle rechts abgebildet.

Neben dem AND- und NAND-Gatter existieren noch weitere Gatter mit zwei Eingangsbits und einem Ausgangsbit. Bei einem davon handelt es sich um das OR-Gatter. Wenn mindestens eines der Eingangsbits im Zustand 1 ist, wird das Ausgangsbit im Zustand 1 ausgegeben. Wenn alle Eingangsbits dem Zustand 0 entsprechen, wird das Ausgangsbit ebenfalls im Zustand 0 ausgegeben. Das XOR-Gatter, also das Exklusive-OR-Gatter, gibt hingegen genau dann den Zustand 1 aus, wenn genau ein Eingangsbit im Zustand 1 ist. Das Schaltzeichen und die Wahrheitstabelle des OR-Gatters sind in Abb. 2.4 dargestellt, die des XOR-Gatters in Abb. 2.5.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

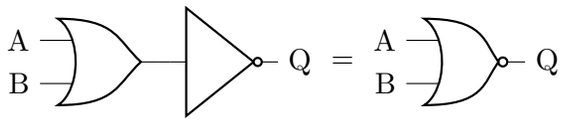
Abbildung 2.4.: Links wird das OR-Gatter dargestellt, wobei die Eingangsbits die Zustände A und B annehmen. Der Zustand Q des Ausgangsbits wird durch die Wahrheitstabelle rechts beschrieben.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

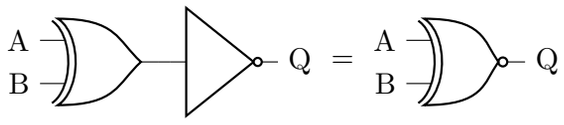
Abbildung 2.5.: Das Gatter der XOR-Schaltung ist links dargestellt. Die Eingangsbits liegen in den Zuständen A und B vor. Welchen Zustand Q das Ausgangsbit annimmt, beschreibt die Wahrheitstabelle rechts.

Auch hinter diese beiden Gatter kann man ein NOT-Gatter schalten. Die Kombination mit dem OR-Gatter wird NOR-Gatter genannt. Dieses gibt genau dann den Zustand 1 aus, wenn alle Eingangsbits dem Zustand 0 entsprechen, sonst wird der Zustand 0 ausgegeben (vgl. Abb. 2.6). Wenn man hingegen hinter das XOR-Gatter ein NOT-Gatter schaltet, erhält man das XNOR-Gatter, welches in Abb. 2.7 abgebildet ist [13]. Dieses gibt den Zustand 1 aus, falls beide Eingangsbits den gleichen Zustand annehmen, sonst wird der Zustand 0 ausgegeben [13].



A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Abbildung 2.6.: Das NOR-Gatter ist rechts abgebildet. Abhängig von den Zuständen A und B der Eingangsbits ergibt sich der Zustand Q des Ausgangsbits. Dies gibt die Wahrheitstabelle rechts an.

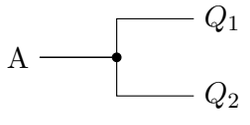


A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Abbildung 2.7.: Das XNOR-Gatter ist rechts dargestellt, wobei A und B die Zustände der Eingangsbits beschreiben. Welchen Zustand Q das Ausgangsbit annimmt, gibt die Wahrheitstabelle rechts an.

Diese sieben Gatter sind die bekanntesten klassischen Gatter [1, 13]. Alle sieben lassen sich nur aus NAND-Gattern oder nur aus XOR-Gattern zusammensetzen [1]. Neben den sieben Gattern können auch alle anderen klassischen Gatter aus diesen beiden Gattern dargestellt werden. Somit sind das NAND-Gatter und das XOR-Gatter universell.

In einem klassischen Schaltkreis kann man außerdem die Bits mit Hilfe einer Verbindungsstelle vervielfachen [13]. Eine solche Verbindungsstelle findet sich in Abb. 2.8 vor.



A	Q_1	Q_2
0	0	0
1	1	1

Abbildung 2.8.: Durch die links abgebildete Verbindungsstelle wird das Eingangsbit, das im Zustand A vorliegt, verdoppelt. Dabei ergeben sich zwei Ausgangsbits, die in den Zuatänden Q_1 und Q_2 vorliegen. Die Zustände der Ausgangsbits ergeben sich wie in der Wahrheitstabelle rechts abgebildet.

Sie verdoppelt somit das Bit, ohne dass dieses seinen Zustand ändert [13]. Dabei wird die Verbindungsstelle nicht mit zu den Gattern gezählt, die sich durch universelle Gatter bilden lassen [13].

2.2.2. Quantengatter

Im Gegensatz zu den klassischen Gattern ist es für die Quantenschaltkreise nicht möglich, eine Verbindungsstelle, ein NAND-Gatter oder ein XOR-Gatter einzubauen [1]. Dies beruht auf einer Grundeigenschaft der Quantengatter. Jedes Quantengatter entspricht im mathematischen Sinn einem unitären Operator U, der auf einem Hilbertraum $\mathcal{H}$

operiert. Das bedeutet, dass der Operator mehrere Bedingungen erfüllt [9]. Diese sind die Folgenden:

$$U : \mathcal{H}^{\otimes n} \to \mathcal{H}^{\otimes n} \tag{2.4}$$

$$U^{\dagger}U = \mathbb{1} \tag{2.5}$$

Aus der Gleichung (2.4) folgt, dass ein Gatter von n Eingangsqubits auf n Ausgangsqubits wirkt. Somit folgt auch, dass alle Gatter bis auf das NOT-Gatter, die im Abschnitt 2.2.1 genannt wurden, nicht direkt in Quantengatter umgewandelt werden können [1]. Dies liegt daran, dass fast alle der in Abschnitt 2.2.1 genannten Gatter zwei Eingangsbits und ein Ausgangsbit bzw. die Verbindungsstelle ein Eingangsbit und zwei Ausgangsbits aufweisen. Nur das NOT-Gatter erfüllt die Bedingung mit der gleich bleibenden Anzahl an Eingangs- und Ausgangsbits. Allerdings kann man die klassischen Gatter so variieren, dass analoge Quantengatter zu finden sind, die ähnlich wie beispielsweise das XOR-Gatter funktionieren. Einige dieser Gatter werden in diesem Abschnitt vorgestellt. Analoge Quantengatter lassen sich aber nicht für alle klassischen Gatter finden. So auch nicht für das universelle NAND-Gatter, da dieses nicht unitär ist [1]. Die Bedingung eines unitären Gatters wird durch Gleichung (2.5) beschrieben.

Da Qubits nicht nur die Zustände $|0\rangle$ und $|1\rangle$ annehmen können, sondern auch eine Superposition dieser möglich ist, kann keine vollständige Wahrheitstabelle aufgestellt werden. Stattdessen werden die Quantengatter der Konvention nach in Matrixform dargestellt [1]. Die Größe der Matrix des Gatters ist dabei $2^n \times 2^n$, wobei n die Anzahl an Qubits angibt, auf die das Gatter wirkt [1]. Für ein Ein-Qubit-Gatter ergibt sich eine 2×2-Matrix, die wie folgt aussieht:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Man betrachte nun die Gleichung (2.5), die aussagt, dass ein Gatter unitär sein muss. Für die Matrix folgt [9]:

$$\begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a^*a + c^*c & a^*b + c^*d \\ b^*a + d^*c & b^*b + d^*d \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.6}$$

Nachfolgend werden einige Quantengatter vorgestellt, welche die Bedingungen aus Gleichung (2.4) und (2.5) erfüllen. Die Quantengatter und ihre Wirkung wurden dabei aus [1] und [9] übernommen.

Ausgewählte Ein-Qubit-Gatter

Eine Matrix, die Gleichung (2.6) erfüllt, ist die des Pauli-X-Gatters. Dieses Gatter funktioniert wie das klassische NOT-Gatter und wird aufgrund dessen Quanten-NOT-Gatter genannt. Abgekürzt findet man es in verschiedenen Lehrbüchern auch unter der Bezeichnung Q-NOT-Gatter. Die Matrix von diesem Gatter sieht wie folgt aus:

$$\mathrm{X} \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Dass dieses Gatter unitär ist, kann man in einer kurzen Rechnung zeigen:

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

Die Wirkung des Gatters auf den Zustand $|\psi\rangle$ aus Gleichung (2.1) ist durch

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \left|\tilde{\psi}\right\rangle = \beta\,|0\rangle + \alpha\,|1\rangle$$

gegeben. Schreibt man diese Zustandsänderung mit Gleichung (2.3) um, so sehen der Zustand vor und nach Anwendung des Pauli-X-Gatters wie folgt aus:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \rightarrow \left|\tilde{\psi}\right\rangle = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Somit bewirkt das Pauli-X-Gatter oder auch Q-NOT-Gatter, dass die Amplituden der Zustände $|0\rangle$ und $|1\rangle$ getauscht werden. In einem Schaltkreis kann dieses Gatter durch zwei verschiedene Symbole gekennzeichnet werden. In dieser Arbeit wird das in Abb. 2.9 rechts abgebildete Gatter benutzt.

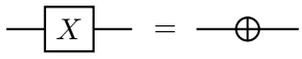
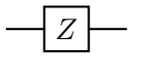


Abbildung 2.9.: Pauli-X- oder Q-NOT-Gatter

Betrachtet man das Pauli-X-Gatter als eine Operation in einem Koordinatensystem, das die Zustände $|0\rangle$ und $|1\rangle$ auf den Achsen hat, so entspricht es einer Spiegelung des Zustandes an der Geraden des 45°-Winkels (vgl. Abb. 2.12) [8].

Neben dem Pauli-X-Gatter existieren noch zwei weitere bekannte Quantengatter, die als Ein-Qubit-Gatter operieren. Dabei handelt es sich um das Pauli-Z-Gatter und das Hadamard-Gatter. Das Pauli-Y-Gatter wird nicht beschrieben, da es in dieser Arbeit keine Anwendung findet. Darüber hinaus wurde es auch in den gelesenen Textbüchern [1, 8, 9] nicht so intensiv behandelt wie die anderen Gatter.

Das Pauli-Z-Gatter tauscht das Vorzeichen der Amplitude des $|1\rangle$-Zustandes. In Abb. 2.10 ist dargestellt, wie es in einem Schaltkreis dargestellt werden kann und wie es in der Matrixschreibweise beschrieben wird.



$$\mathrm{Z} \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

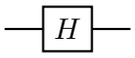
Abbildung 2.10.: Links ist abgebildet, wie das Pauli-Z-Gatter in einem Quantenschaltkreis dargestellt wird. Rechts ist die Funktion dieses Gatters in Matrixform zu sehen.

Dieses Gatter bewirkt die folgende Zustandsänderung:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \left|\tilde{\psi}\right\rangle = \alpha\,|0\rangle - \beta\,|1\rangle$$
$$|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \rightarrow \left|\tilde{\psi}\right\rangle = \begin{bmatrix}\alpha\\ -\beta\end{bmatrix}$$

Da es sich um eine Änderung des Vorzeichens der Amplitude des $|1\rangle$-Zustandes handelt, kann man auch dieses Gatter in dem Koordinatensystem mit $|0\rangle$ und $|1\rangle$ auf den Achsen beschreiben. Hierbei handelt es sich, wie in Abb. 2.12 dargestellt, um eine Spiegelung des Zustandes an der $|0\rangle$-Achse [8]. Bei dieser Spiegelung wird nämlich das Vorzeichen des $|1\rangle$-Zustandes gewechselt und die Amplitude des $|0\rangle$-Zustandes bleibt unverändert, was dem Pauli-Z-Gatter entspricht.

Das letzte Gatter, das auf ein einzelnes Qubit wirkt und hier beschrieben wird, ist das Hadamard-Gatter. Die Matrix dieses Gatters und die Darstellung im Schaltkreis sind in Abb. 2.11 gezeigt.



$$\mathrm{H} \equiv \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \tag{2.7}$$

Abbildung 2.11.: Links ist das Hadamard-Gatter dargestellt. Wie es sich auf einen Zustand auswirkt beschribt die Matrix, die links abgebildet ist.

In Polarkoordinaten handelt es sich um eine Drehung des Zustandes um 90° um die y-Achse und darauf folgend eine Drehung um 180° um die x-Achse. Wendet man das Gatter auf einen Zustand an, so hat dies die folgende Zustandsänderung zur Folge:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \left|\tilde{\psi}\right\rangle = \frac{\alpha+\beta}{\sqrt{2}}\,|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}\,|1\rangle$$
$$|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \rightarrow \left|\tilde{\psi}\right\rangle = \begin{bmatrix}\frac{\alpha+\beta}{\sqrt{2}}\\ \frac{\alpha-\beta}{\sqrt{2}}\end{bmatrix}$$

Dieses Gatter bewirkt eine Spiegelung des Zustandes an einer Geraden im 22.5°-Winkel im Koordinatensystem mit $|0\rangle$ und $|1\rangle$ an den Achsen, was in Abb. 2.12 dargestellt ist [8].

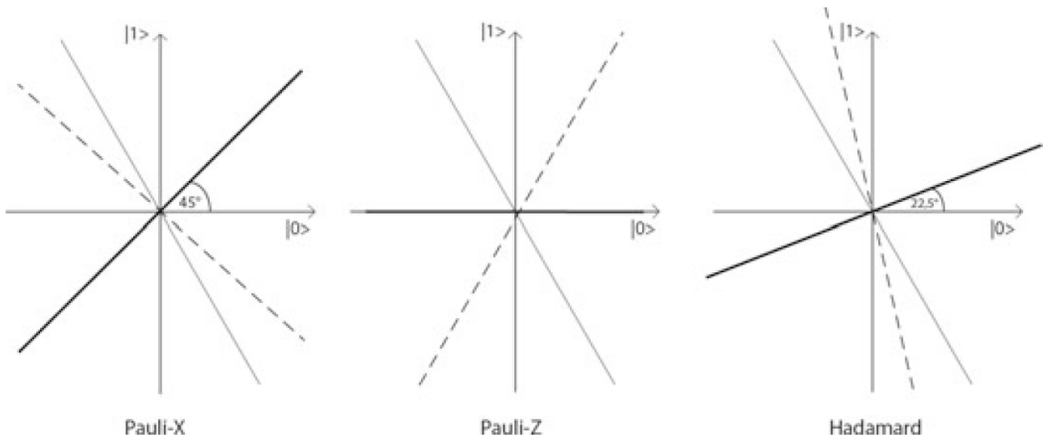


Abbildung 2.12.: Abgebildet ist die Wirkung dreier Gatter auf einen Zustand nach [8]. Die Spiegelachsen der Pauli-X-, Pauli-Z- und Hadamard-Quantengatter sind dick und ihre Wirkung auf einen im 120°-Winkel polaisierten Zustand durch eine gestrichelte Linie markiert.

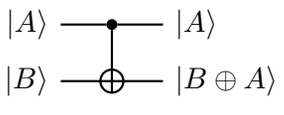
Dies sind die drei bekanntesten Ein-Qubit-Gatter. Wie sich erkennen lässt, ist das Pauli-X-Gatter das quantenmechanische Analogon zum klassischen NOT-Gatter.

Ausgewählte Mehr-Qubit-Gatter

Im quantenmechanischen Fall kann man ebenfalls wie im klassischen Fall Gatter auf mehrere Qubits wirken lassen. So auch beim Controlled-NOT-Gatter (CNOT-Gatter). In dieses werden die Zustände $|A\rangle$ und $|B\rangle$ injiziert. Das Quantengatter gibt als Ausgangszustände die Zustände $|A\rangle$ und $|B \oplus A\rangle$ aus. $\oplus$ beschreibt die Addition von A und B modulo 2. Somit ändert sich der Zustand $|B\rangle$, falls sich der Zustand $|A\rangle$ aus dem Zustand $|1\rangle$ zusammensetzt. Die Zustandsänderung wird durch

$$|\psi\rangle = |A, B\rangle \rightarrow \left|\tilde{\psi}\right\rangle = |A, B \oplus A\rangle$$

beschrieben. Diese kann durch die Matrix in Abb. 2.13 bewirkt werden. Somit entspricht die Matrix aus Abb. 2.13 der Matrix für das CNOT-Gatter. Das Schaltsymbol ist ebenfalls in dieser Abbildung dargestellt.



$$\text{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Abbildung 2.13.: Das CNOT-Gatter wirkt auf zwei Qubits. Links ist gezeigt, wie es in einem Quantenschaltkreis abgebildet wird. Rechts ist die Funktion des Gatters in Matrixform dargestellt.

Zur besseren Anschaulichkeit, was das CNOT-Gatter für eine Zustandsänderung bewirkt, kann eine Wahrheitstabelle angelegt werden. Diese findet sich in Tabelle 2.1 vor. Es muss allerdings beachtet werden, dass diese nicht vollständig ist. Wegen der bei Qubits möglichen Superpsoition von Zuständen, kann das Gatter auch auf andere Zustände wirken als auf diejenigen, welche in der Tabelle aufgelistet sind.

Input		Output	
$\|A\rangle$	$\|B\rangle$	$\|A\rangle$	$\|B \oplus A\rangle$
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Tabelle 2.1.: Die Wahrheitstabelle des CNOT-Gatters, die hier abgebildet ist, beschränkt sich auf die Zustände $|0\rangle$ und $|1\rangle$ der einkommenden Qubits.

In dieser Wahrheitstabelle wird der Zustand $|B\rangle \rightarrow |B \oplus A\rangle$ genau dann gewechselt, wenn der Input dem Zustand $|A\rangle = |1\rangle$ entspricht. Es lässt sich nun vermuten, dass man mit dem XOR-Gatter Zustände kopieren kann. Wenn man den Zustand $|B\rangle$ als $|0\rangle$ präpariert, entsteht folgende Zustandsänderung:

$$\begin{aligned} |\psi\rangle\,|0\rangle &= [\alpha\,|0\rangle + \beta\,|1\rangle]\,|0\rangle = \alpha\,|00\rangle + \beta\,|10\rangle \\ &\rightarrow \left|\tilde{\psi}\right\rangle = \alpha\,|00\rangle + \beta\,|11\rangle \end{aligned}$$

Würde man den Zustand wirklich kopieren, so müsste man den Zustand

$$\left|\tilde{\psi}\right\rangle = |\psi\rangle\,|\psi\rangle = \alpha^2\,|00\rangle + \alpha\beta\,|01\rangle + \alpha\beta\,|10\rangle + \beta^2\,|11\rangle$$

erhalten [1]. Die Zustände der Qubits unterliegen nämlich der Verschränkung. Somit bleibt das „No-Cloning-Theorem“ unverletzt [1]. Dieses besagt, dass kein Qubit perfekt kopiert werden kann, weswegen auch kein analoges Quantengatter zur Verbindungsstelle existiert.

Das CNOT-Gatter lässt sich mit dem klassischen XOR-Gatter vergleichen. Der Unterschied besteht darin, dass zum einen mit Qubits statt Bits gearbeitet wird, wodurch auch Superposition möglich ist. Zum anderen sind zwei Ausgangszustände bei dem CNOT-Gatter vorhanden, beim XOR-Gatter gibt es nur ein Ausgangsbit. Durch diese Unterschiede erfüllt das CNOT-Gatter die Bedingungen für ein Quantengatter.

Ähnlich zu dem CNOT-Gatter funktioniert ein Toffoli-Gatter. Dieses benötigt ebenfalls mehrere Qubits für den Input. Eine Zustandsänderung des Zustandes $|C\rangle$ geschieht dabei genau dann, wenn alle Zustände bis auf $|C\rangle$ den $|1\rangle$-Zustand enthalten. Die Zustandsänderung für drei Qubits sieht wie folgt aus:

$$|\psi\rangle = |A, B, C\rangle \rightarrow \left|\tilde{\psi}\right\rangle = |A, B, C \oplus AB\rangle$$

In diesem Beispiel wird der Zustand $|C\rangle$ verändert, wenn beide Zustände $|A\rangle$ und $|B\rangle$ jeweils den Zustand $|1\rangle$ enthalten. Das Gatter kann in der Matrixschreibweise folgendermaßen dargestellt werden:

$$\text{Toffoli} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

In einem Schaltkreis wird es wie in Abb. 2.14 gezeigt eingebaut.

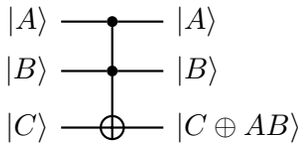


Abbildung 2.14.: Dargestellt ist ein Toffoli-Gatter, das den Zustand $|C\rangle$ zu $|C \oplus AB\rangle$ ändert.

Auch für das Toffoli-Gatter kann man wie für das CNOT-Gatter eine Wahrheitstabelle betrachten (vgl. Tabelle 2.2), die allerdings wie auch Tabelle 2.1 nicht ganz vollständig ist.

Input			Output		
$\|A\rangle$	$\|B\rangle$	$\|C\rangle$	$\|A\rangle$	$\|B\rangle$	$\|C \oplus AB\rangle$
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Tabelle 2.2.: Abgebildet ist hier die Wahrheitstabelle des Toffoli-Gatters. Dabei wurden nur die Zustände $|0\rangle$ und $|1\rangle$ der einkommenden Qubits betrachtet.

Man kann hierbei beobachten, dass bei Nutzung des Zustandes $|0\rangle$ als $|C\rangle$ eine analoge Wahrheitstabelle zum klassischen AND-Gatter entsteht.

Eine Variation des CNOT-Gatters und des Toffoli-Gatters wie in Abb. 2.15 erhält man, indem man statt einer Nutzung des Zustandes $|1\rangle$ für den Zustandswechsel den Zustand $|0\rangle$ fordert. Dies ist durch die Anwendung des Pauli-X-Gatters vor und hinter dem Quantengatter an dem Qubit, an dem der Zustand $|0\rangle$ gefordert wird, möglich.

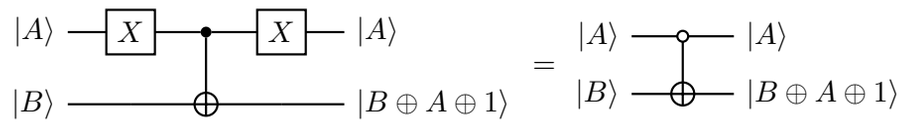


Abbildung 2.15.: Aus der Anwendung zweier Pauli-X-Gatter in Kombination mit einem CNOT-Gatter (links) ergibt sich das invertierte CNOT-Gatter (rechts).

Somit wird der Zustand $|B\rangle$ genau dann geändert, wenn sich $|A\rangle$ aus dem Zustand $|0\rangle$ zusammensetzt. Dieses Gatter wird das invertierte CNOT-Gatter genannt.

Ein weiteres Gatter bildet das Controlled-Z-Gatter. Dieses wird als CZ-Gatter abgekürzt. Es führt zu einem Vorzeichenwechsel des Zustandes $|11\rangle$. In Matrixschreibweise wird es als

$$\mathrm{CZ} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

beschrieben. Die Zustandsänderung, die nur auf den Zustand $|11\rangle$ wirkt, wird beschrieben durch:

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle \rightarrow \left|\tilde{\psi}\right\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle - \delta\,|11\rangle$$

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \left|\tilde{\psi}\right\rangle = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ -\delta \end{bmatrix}.$$

Es gibt zwei Möglichkeiten, das CZ-Gatter in einem Schaltkreis darzustellen. Diese sind in Abb. 2.16 dargestellt. In der folgenden Arbeit wird die zweite, also die rechte Darstellung, verwendet.

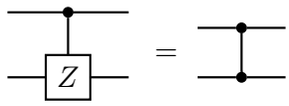


Abbildung 2.16.: Abgebildet sind zwei mögliche Darstellungen des CZ-Gatters, wobei im Folgenden die rechte Darstellung verwendet wird.

Ebenfalls wie das CNOT-Gatter kann man auch das CZ-Gatter modifizieren, indem man es um mehrere Qubits erweitert oder invertierte CZ-Gatter daraus erstellt. Dies funktioniert analog zu dem CNOT-Gatter, das zu einem Toffoli-Gatter oder invertiertem CNOT-Gatter verändert wird.

Mit den CNOT-Gatter, dem Toffoli-Gatter und den Ein-Qubit-Gattern kann man jedes beliebige Quantengatter aufstellen. Der Beweis dazu findet sich im Kapitel 4.5 des Buches „Quantum Computation and Quantum Information“ vor [1].

3 Kapitel 3. Grundlagen für das Travellings Salesman Problem

3.1. NP-Schwere Probleme

Ein wichtiges Thema, in dem Quantencomputer eine wesentliche Verbesserung im Gegensatz zu klassischen Computern erwarten lassen, ist die Komplexitätstheorie. Diese betrachtet mehrere Klassen an Problemen. Die zwei wichtigsten Klassen sind die **P**- und **NP**-Probleme [1]. Neben diesen gibt es noch viele weitere Problemklassen, im Folgenden wird aber nur auf die wichtigsten beiden Klassen eingegangen. Die Klasse der **P**-Probleme lässt sich auf einem klassischen Computer schnell lösen. Dabei handelt es sich um Probleme mit einer polynomiellen Laufzeit [1]. Das bedeutet, dass die Dauer der Problemlösung durch die Funktion $f(n) \propto n^c$ beschrieben werden kann, wobei c eine endliche Zahl und n die Größe des Problems beschreibt [1]. Die Laufzeit wird durch $\mathcal{O}(n^c)$ angegeben [1]. Es gibt jedoch auch Probleme, die sich nicht in einer polynomiellen Laufzeit lösen lassen. Eines davon ist die Faktorisierung einer ganzen Zahl m [1]. Allerdings lässt sich eine mögliche Lösung dieses Problems schnell überprüfen [1]. Dadurch definiert sich die Klasse der **NP**-schweren Probleme: Die Probleme haben eine nicht polynomielle Laufzeit, die Lösung der Probleme lässt sich aber auf einem klassischen Computer in einer polynomiellen Zeit überprüfen [14]. Ein Beispiel einer nicht polynomiellen Laufzeit ist die exponentielle Laufzeit. Diese wird durch $\mathcal{O}^*(k^n)$ beschrieben, wobei n die Größe des Problems beschreibt und k einer endlichen Zahl entspricht [1]. Das $\mathcal{O}^*$ bedeutet im Vergleich zu $\mathcal{O}$, dass polynomielle Komplexitäten der Laufzeit ignoriert werden. Das heißt, dass $f(n) = n^c \cdot k^n = poly(n) \cdot k^n$ in der Komplexitätstheorie durch $f(n) = \mathcal{O}(n^c k^n)$ oder aber durch $f(n) = \mathcal{O}^*(k^n)$ beschrieben werden kann [2].

Im Vergleich zu einem **P**-Problem wächst die Laufzeit eines **NP**-Problems mit steigender Problemgröße sehr stark an. Aufgrund dessen kann ein **NP**-Problem auf einem klassischen Computer für eine große Problemgröße nur schwer gelöst werden [1]. Es lässt sich beispielsweise eine sehr lange Laufzeit erwarten.

Ein weiteres Beispiel eines **NP**-Problems ist das Travelling Salesman Problem, das im Folgenden beschrieben wird [1].

3.2. Travelling Salesman Problem - Das Problem des Handlungsreisenden

In dieser Arbeit wird sich mit einem klassichen Algorithmus und Quantenalgorithmen zur Lösung des Travelling Salesman Problems beschäftigt. Das Travelling Salesman Problem (auch TSP oder das Problem des Handlungsreisenden genannt) beschreibt ein Optimierungsproblem aus der Kombinatorik [4]. Das Ziel dieses Problems ist es, n Ecken miteinander zu verbinden, sodass die Weglänge minimiert wird [15]. Dabei muss jede Ecke genau ein Mal besucht werden. Nur der Anfangspunkt darf zweimal besucht werden, da an diesem der Weg wieder enden soll [15]. Das Problem ähnelt dem Hamiltonkreisproblem. Dieses befasst sich mit der Frage, ob überhaupt ein Weg existiert, der jede Ecke von S, mit Ausnahme des Startpunktes, genau ein Mal besucht [1]. Das TSP sucht also nach dem kürzesten Hamiltonkreis. Die Weglänge wird durch die Kantengewichte a beschrieben. a_{ij} ist dabei das Kantengewicht von Ecke i nach Ecke j [4]. Wird nun im Gesamtweg der direkte Weg von i nach j genutzt, so fließt die dazugehörige Kantenlänge in die Weglänge des Gesamtweges mit ein.

Ein Beispiel für das TSP ist eine Anwendung auf eine Rundreise, auf der eine bestimmte Anzahl an Städten besucht werden soll. Die Ecken beschreiben hierbei verschiedene Städte. Die Kantengewichte hingegen entsprechen der Entfernung je zweier Städte. Jede Stadt muss genau ein Mal besucht werden. Nun besteht die Aufgabe daraus eine Rundreise zu finden, die jede Stadt genau ein Mal besucht und bei der die Länge des Gesamtweges minimiert wird. Dabei soll die Rundreise an der Stadt enden, bei welcher der Weg begonnen hat. Neben der Minimierung der Strecke zwischen den Städten kann man beispielsweise auch die Reisedauer, die Reisekosten oder andere Parameter minimieren. Der Parameter, der minimiert werden soll, entspricht dem Kantengewicht.

Um das TSP zu lösen, gibt es verschiedene Methoden. Dabei stellt sich die Frage, welcher dieser Lösungsansätze die Laufzeit optimiert. Im Folgenden wird dies für eine Auswahl an Lösungsmöglichkeiten geklärt.

Eine erste Möglichkeit ist die zufällige Suche nach einem optimalen Weg. Bei n Ecken gibt es $n!$ Kombinationsmöglichkeiten für die Reihenfolge der Städte, was zu einer Laufzeit von $\mathcal{O}^*(n!)$ führt [2]. Dabei schafft der Grover-Algorithmus eine Beschleunigung zu einer Laufzeit von $\mathcal{O}^*(\sqrt{n!})$ [2]. Wie der Grover-Algorithmus funktioniert, wird in Abschnitt 3.4 beschrieben. Der klassische Algorithmus, der die beste Laufzeit hat, basiert auf der dynamischen Programmierung [2]. Dieser führt zu einer Laufzeit von $\mathcal{O}(n^2 2^n \log L)$ [2]. Auf Basis dieses Algorithmus haben Ambainis et al. einen Quantenalgorithmus entworfen, der eine Laufzeit von $\mathcal{O}^*(1.728^n \log L)$ erreicht, wobei L das maximale Kantengewicht beschreibt [2]. Von den vier beschriebenen Algorithmen hat der von Ambainis et al. die beste Laufzeit. Er wird in Abschnitt 4.2 beschrieben und auf diesen Algorithmus wird sich in der folgenden Arbeit fokussiert.

3.3. Quantenalgorithmen für eine schnellere Lösung von Problemen

Es lässt sich also beobachten, dass Quantenalgorithmen schnellere Lösungen erwarten lassen als Algorithmen an klassischen Computern. Dabei stellt sich die wichtige Frage, ob **P**- und **NP**-Probleme sich überhaupt unterscheiden. Es besteht die Möglichkeit, dass alle **NP**-Probleme sich mit Algorithmen lösen lassen, die polynomielle Laufzeiten haben [1]. Die richtigen Algorithmen könnten bisher einfach nicht gefunden worden sein. Dies ist eine wichtige Frage der Komplexitätstheorie, die bisher noch nicht gelöst wurde und mit der sich auch eines der sieben Millennium-Probleme auseinandersetzt [14]. Somit muss gelöst werden, ob $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$, also $\mathbf{P} \subset \mathbf{NP}$, gilt [1]. Falls $\mathbf{P} \neq \mathbf{NP}$ gilt, lassen sich **NP**-schwere Probleme auf einem klassischen Computer nicht effizient lösen [1]. In diesem Fall ist die Hoffnung, dass Quantencomputer zu einer schnelleren Lösung führen [1]. Dabei helfen Quantenalgorithmen, die kürzere Laufzeiten haben als klassische Algorithmen. Ein Beispiel dafür ist der Grover-Algorithmus.

3.4. Grover-Algorithmus

Der Grover-Algorithmus ist ein quantenmechanischer Algorithmus, der von Grover entwickelt wurde und eine schnellere Suche von Elementen aus einer Liste liefern soll [16]. Dieses Kapitel basiert auf diesem Paper von Grover sowie auf dem Buch von Nielsen und Chuang [1] und dem Paper von Boyer et al. [17]. Die Suche nach bestimmten Elementen S_ν läuft wie folgt ab: Zuerst wird ein System definiert, das $N = 2^n$ Zustände hat. Dabei ist n die Anzahl an Qubits, die zur Definition der Zustände genutzt werden. Es ergeben sich die Zustände $S_1, S_2, \ldots, S_N$. Der gesuchte Zustand S_ν soll nun unter diesen Zuständen gefunden werden. Dafür wird eine Bedingung C definiert. Wird diese Bedingung auf einen Zustand S_i aus der Menge an Zuständen angewandt, so ergibt sich $C(S_i) = 1$, falls S_i dem gesuchten Zustand S_ν entspricht. Es folgt also $C(S_\nu) = 1$. Entspricht S_i nicht dem gesuchten Zustand, so folgt $C(S_i) = 0$. Diese Bedingung kann aber auch für das Finden mehrerer Elemente in der Liste angewandt werden. $C(S_i)$ ergibt dabei 1 für alle Zustände S_i, die eine bestimmte Bedingung erfüllen. Um nun herauszufinden, welche Zustände die Bedingung C erfüllen, wird beispielsweise der Grover-Algorithmus benutzt.

Man nehme an, man sucht in einer Liste mit $N = 2^n$ Einträgen einen bestimmten Eintrag. Durch willkürliches Herumprobieren bräuchte man $\frac{N}{2}$ Suchen, um den gewünschten Eintrag mit einer Wahrscheinlichkeit von $\frac{1}{2}$ zu finden. Wendet man allerdings den Grover-Algorithmus an, so kann man die Laufzeit von $\mathcal{O}(N)$ auf eine Laufzeit von $\mathcal{O}(\sqrt{N})$ senken. Der Grover-Algorithmus lässt somit eine quadratische Beschleunigung der Suche zu.

Der Grover-Algorithmus besteht aus drei Schritten.

1. Der erste Schritt schafft ein System, in dem alle Zustände $S_1, \ldots, S_N$ mit der gleichen Wahrscheinlichkeit auftauchen. Da N Zustände vorhanden sind, ist die Amplitude jedes Zustandes $\frac{1}{\sqrt{N}}$. Im Folgenden werden die Zustände $|00\ldots0\rangle, |00\ldots1\rangle, \ldots, |11\ldots1\rangle$ benannt. Sie bilden zusammen eine Orthonomalbasis. Insgesamt ergibt sich dadurch das

System im folgenden Zustand:

$$|S\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle \tag{3.1}$$

$|i\rangle$ stellt hierbei die Binärdarstellung der Zustände dar. Der gesuchte Zustand wird $|\nu\rangle$ genannt.

Um den Zustand $|S\rangle$ zu erzeugen, können zuerst alle n Qubits in dem Zustand $|0\rangle$ präpariert werden. Danach wird auf allen Qubits das Hadamard-Gatter angewandt. Dadurch wird eine Superposition aller Zustände mit gleichen Amplituden erzeugt, was dem Zustand $|S\rangle$ entspricht. Dieser Schritt benötigt eine Laufzeit von $\mathcal{O}(n) = \mathcal{O}(\log_2 N)$.

2. Der nächste Schritt des Algorithmus ist der Hauptteil des Algorithmus. Dabei lässt sich der zweite Schritt in zwei Teile unterteilen.

2.a. Im ersten Teil wird ein Orakel angewandt. Dieses dreht die Phase eines Zustandes i um π, falls $C(i) = 1$ also $i = \nu$ gilt. Bei $C(i) = 0$, also $i \neq \nu$, bleibt der Zustand unverändert. Dieses Orakel wird U_ν genannt. Es führt somit die folgende Operation durch:

$$U_\nu |i\rangle = \begin{cases} |i\rangle & \text{falls } \; i \neq \nu \\ -|i\rangle & \text{falls } \; i = \nu \end{cases} \tag{3.2}$$

Das Orakel kann ebenso wie die Gatter in Abschnitt 2.2.2 in der Matrixdarstellung dargestellt werden. Dabei gilt für das Orakel:

$$U_\nu = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \leftarrow \nu \\ \\ \\ \\ \end{matrix} \tag{3.3}$$

Die Matrix kann auch allgemeiner aufgeschrieben werden. Hierbei kann die Funktion C verwendet werden, die aussagt, ob es sich um das gesuchte Element handelt oder nicht. Somit kann man Gleichung (3.2) und Gleichung (3.3) umschreiben zu:

$$U_\nu |i\rangle = (-1)^{C(i)} |i\rangle \quad \text{mit} \quad U_\nu = \begin{bmatrix} (-1)^{C(0)} & 0 & \cdots & 0 \\ 0 & (-1)^{C(1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{C(N-1)} \end{bmatrix} \tag{3.4}$$

2.b. Nach dem Anwenden des Orakels wird die Diffusionstransformation, beschrieben durch die Matrix D, angewandt. Die Matrix D kann implementiert werden als $D = H_n R H_n$. H_n ist hierbei die Walsh-Hadamard-Transformationsmatrix und wird beschrieben durch

$$H_{n_{ij}} = \frac{1}{\sqrt{2^n}} \cdot (-1)^{\overline{i} \cdot \overline{j}}.$$

Hierbei stellen $\overline{i}$ und $\overline{j}$ die binäre Darstellung von i und j und $\overline{i} \cdot \overline{j}$ das bitweise Skalarprodukt dieser dar. Die Matrix entspricht der Hadamard-Matrix für das Hadamard-Gatter aus Gleichung (2.7) für eine Größe von 2×2. Für den Algorithmus wird aber eine $2^n \times 2^n$-große Matrix benötigt. Diese wird durch $H_n = H_1 \otimes H_{n-1}$ konstruiert, wobei H_i eine $2^i \times 2^i$-große Matrix darstellt [18]. Somit ist H_1 die aus Gleichung (2.7) bekannte Matrix H.

R hingegen beschreibt die Rotationsmatrix. Diese ist wie folgt definiert:

$$\begin{aligned} R_{ij} &= 0 && \text{falls } \; i \neq j \\ R_{ii} &= 1 && \text{falls } \; i = 0 \\ R_{ii} &= -1 && \text{falls } \; i \neq 0 \end{aligned}$$

Somit folgt:

$$R = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{bmatrix}$$

R kann man auch als $R = 2\,|0\rangle\,\langle 0| - \mathbb{1}$ schreiben.

Die beiden Matrizen H_n und R werden durch $H_n R H_n$ zu D zusammengesetzt. Durch Umschreiben von R ergibt sich:

$$\begin{aligned} D &= H_n R H_n \\ &= H_n (2\,|0\rangle\,\langle 0| - \mathbb{1}) H_n \\ &= 2\,|S\rangle\,\langle S| - \mathbb{1} \end{aligned}$$

Dabei ist $|S\rangle$ der Zustand aus Gleichung (3.1). Durch die Gleichung lässt sich erkennen, dass die Anwendung der Matrix D als eine Spiegelung des Zustandes an dem Mittelwert aller Amplituden interpretiert werden kann.

Schritt 2.a. und 2.b. werden beide $\mathcal{O}(\sqrt{N})$ Mal durchgeführt.

3. Im letzten Schritt wird der Endzustand gemessen. Wenn $C(\nu) = 1$ eine einzelne Lösung besitzt, dann folgt, dass der Zustand $|\nu\rangle$ mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ gemessen wird.

Der Grover-Algorithmus lässt sich auch in einem Schaltkreis darstellen. Dieser wird in Abb. 3.1 schematisch gezeigt.

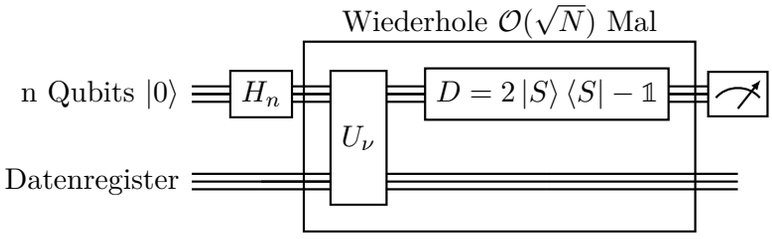


Abbildung 3.1.: Dargestellt ist ein schematischer Aufbau des Quantenschaltkreises für einen Grover-Algorithmus nach [1]. Der Algorithmus arbeitet dabei mit den ersten n Qubits. Das Datenregister ist zum Prüfen der Bedingung C durch das Orakel eingebaut.

3.4.1. Grafische Darstellung des Grover-Algorithmus

Der zweite Schritt des Grover-Algorithmus lässt sich auf zwei Weisen grafisch darstellen. Für die erste Darstellungsweise ist ein weiterer Zustand nötig. Dieser ist ähnlich zu $|S\rangle$, allerdings ist er orthogonal zu $|\nu\rangle$. Der Zustand sieht wie folgt aus:

$$|S'\rangle = \frac{1}{\sqrt{N-1}} \left(\sum_{i=0}^{N-1} |i\rangle - |\nu\rangle \right)$$

Die erste Darstellungsweise (links) in Abb. 3.2, 3.3 und 3.4 stellt ein Koordinatensystem mit $|S'\rangle$ und $|\nu\rangle$ auf den Achsen dar. Wenn der Zustand sich dem gesuchten Zustand $|\nu\rangle$ annähert, nähert sich der Zustand in dem Koordinatensystem der Ordinate an. Die zweite Darstellungsweise (rechts) betrachtet die Amplituden der einzelnen Zustände. Der Grover-Algorithmus hat das Ziel, die Amplitude von $|\nu\rangle$ an ± 1 und die der anderen Zustände an 0 anzunähern. Der erste Schritt des Grover-Algorithmus stellt den Zustand $|S\rangle$ her. Dies wird grafisch in Abb. 3.2 dargestellt.

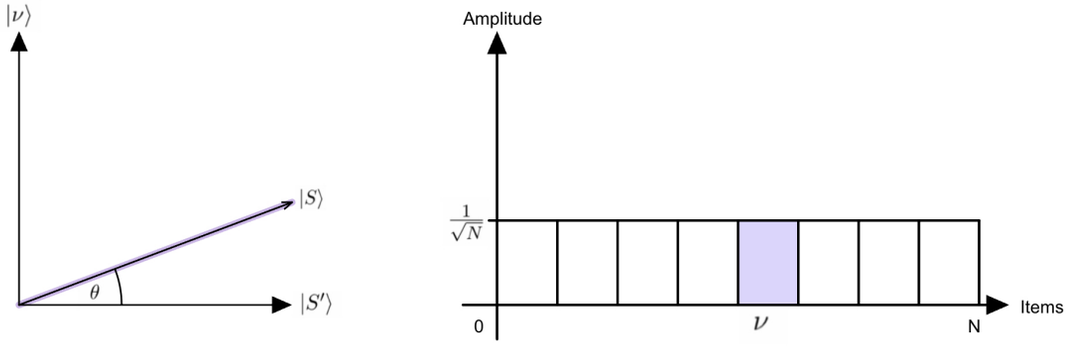


Abbildung 3.2.: Zu sehen ist der Zustand $|S\rangle$ nach dem ersten Schritt des Grover-Algorithmus nach [19]. Links ist in violett der Zustand $|S\rangle$ in einem Koordinatensystem mit $|S'\rangle$ und $|\nu\rangle$ an den Achsen dargestellt. Rechts hingegen ist eine Abbildung der Amplituden zu sehen, wobei der gesuchte Zustand violett dargestellt ist. Alle Amplituden nehmen im ersten Schritt den gleichen Wert $\frac{1}{\sqrt{N}}$ an.

Der Winkel, der in der linken Abbildung zwischen $|S\rangle$ und $|S'\rangle$ entsteht, ist

$$\theta = \arcsin \langle S|\nu\rangle = \arcsin \frac{1}{\sqrt{N}}. \tag{3.5}$$

Nun wird das Orakel U_ν auf den Zustand $|S\rangle$ angewandt, sodass die Phase von $|\nu\rangle$ umgedreht wird. In Abb. 3.3 wird dies dargestellt.

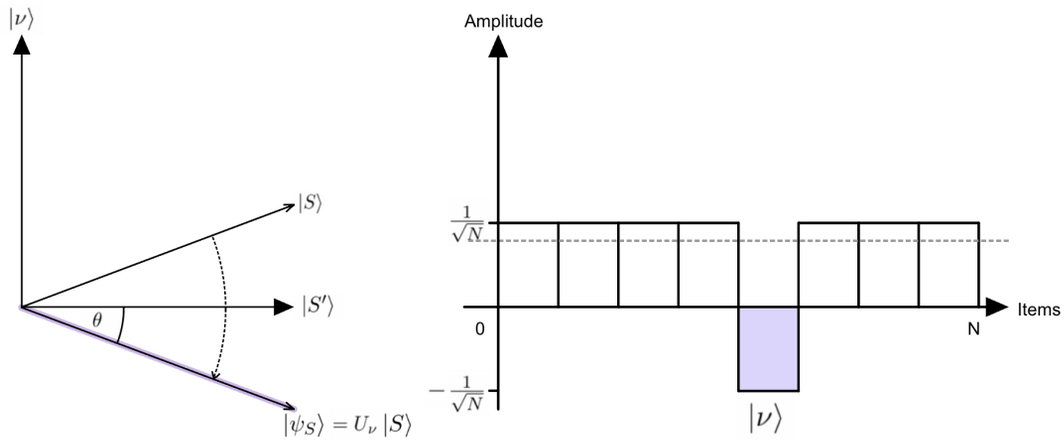


Abbildung 3.3.: Das Orakel U_ν aus dem Grover-Algorithmus wird auf den Zustand $|S\rangle$ angewandt. In der linken Darstellung wird $|S\rangle$ dadurch an der Abszisse gespiegelt, wodurch der neue Zustand $|\psi_S\rangle$ entsteht. Dieser ist violett gekennzeichnet. In der rechten Abbildung wird nur die Amplitude des Zustandes $|\nu\rangle$, die ebenfalls violett dargestellt ist, an der Abszisse gespiegelt. Der Mittelwert aller Amplituden ist grau gestrichelt abgebildet.

Das Orakel führt zu einer Phasenänderung des Zustandes $|\nu\rangle$. In der linken Grafik wird aufgrund dessen der ganze Zustand an der Abszisse reflektiert. Der neue Zustand wird $|\psi_S\rangle$ benannt. In der rechten Grafik wird die Amplitude des Zustandes $|\nu\rangle$ an der Abszisse reflektiert. Die grau gestrichelte Linie stellt dabei den Mittelwert der Amplituden dar.

Die Anwendung des Diffusionstransformationsoperators D auf $|\psi_S\rangle$ ist in Abb. 3.4 zu sehen. Dieser führt zu einer Reflexion von dem Zustand $|\psi_S\rangle$ an dem Zustand $|S\rangle$ in der linken Darstellung bzw. einer Reflexion aller Amplituden am Mittelwert dieser in der Abbildung rechts. Es entsteht der Zustand $|\psi'_S\rangle$.

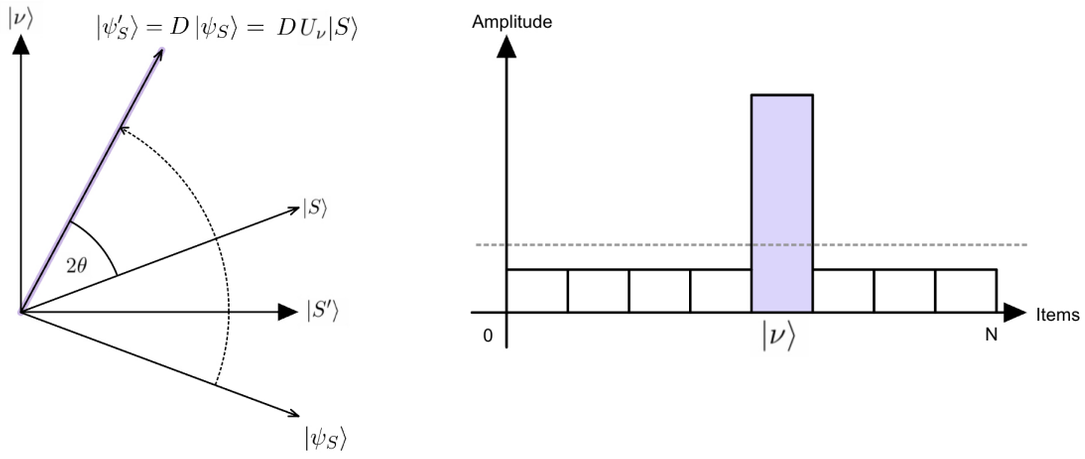


Abbildung 3.4.: Abgebildet ist die Anwendung des Diffusionstransformationsoperators D auf den Zustand $|\psi_S\rangle$. Links ist dargestellt, wie dieser an $|S\rangle$ reflektiert wird. Der neue Zustand (violiett) wird $|\psi'_S\rangle$ genannt. Er hat einen Winkel von 2θ zu $|S\rangle$. Rechts ist zu sehen, wie alle Amplituden am Mittelwert der vorherigen Amplituden (grau gestrichelt) reflektiert wurden. Der Zustand $|\nu\rangle$ ist hierbei violett dargestellt.

Somit wird nach der Anwendung des Orakels und des Diffusionstransformationsoperators in der linken Grafik der Zustand um $2\theta = 2\arcsin\frac{1}{\sqrt{N}}$ zu $|\nu\rangle$ gedreht. Durch Wiederholung dieser beiden Schritte wird der Zustand immer näher an $|\nu\rangle$ gedreht. Nach x Schritten ist der Winkel zwischen dem aktuellen Zustand und $|S'\rangle$ $(1+2x)\arcsin\frac{1}{\sqrt{N}}$. Das Ziel ist es, einen Winkel nahe $\frac{\pi}{2}$ zu erreichen, weil der Zustand sich dann nahe $|\nu\rangle$ befindet. Für große N gilt $\arcsin\frac{1}{\sqrt{N}} \approx \frac{1}{\sqrt{N}}$. Es werden also

$$\frac{\pi}{2} = (1+2x)\arcsin\frac{1}{\sqrt{N}} \quad \Leftrightarrow \quad x = \frac{\pi}{4}\frac{1}{\arcsin\frac{1}{\sqrt{N}}} - \frac{1}{2} \tag{3.6}$$

$$\Rightarrow \quad x \approx \frac{\pi}{4}\sqrt{N} - \frac{1}{2}$$

Schritte für das Erreichen der richtigen Lösung benötigt. Nach $\mathcal{O}(\sqrt{N})$ Schritten befindet sich der Zustand am nächsten zur Ordinate, weswegen die Messung des gesuchten Zustandes dann am wahrscheinlichsten ist. Wird der Grover-Algorithmus häufiger angewandt, so wird der Zustand über $|\nu\rangle$ hinweg gedreht.

In der rechten Darstellung in Abb. 3.4 wird nach Anwendung des Orakels und des Diffusionstransformationsoperators die Amplitude von $|\nu\rangle$ erhöht und die der anderen Zustände kleiner. Auch hier wird der Effekt wieder umgedreht, wenn der Grover-Algorithmus häufiger als $\mathcal{O}(\sqrt{N})$ Mal angewandt wird.

3.4.2. Mathematische Darstellung des Grover-Algorithmus

Neben der grafischen Darstellung des Grover-Algorithmus kann man auch den Zustand betrachten, der sich mit jeder Grover-Iteration verändert. Dabei ist dieser Zustand nach dem ersten Schritt wie folgt:

$$|S\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

Nun wird der Zustand $|\nu\rangle$ gesucht. Daher ergibt es Sinn den Zustand $|S\rangle$ umzuschreiben zu:

$$|S(k,l)\rangle = k\,|\nu\rangle + \sum_{x\neq\nu} l\,|x\rangle$$

Dabei kann x Werte zwischen 0 und $N-1$ annehmen. Damit $|S(k,l)\rangle$ normiert ist, muss $k^2 + (N-1)l^2 = 1$ gelten. Nach j Wiederholungen des zweiten Schrittes des Grover-Algorithmus nehmen k und l die Werte aus Gleichung (3.7) und Gleichung (3.8) an. θ lässt sich aus Gleichung (3.5) entnehmen.

$$k_j = \sin\left((2j+1)\theta\right) \tag{3.7}$$

$$l_j = \frac{1}{\sqrt{N-1}} \cos\left((2j+1)\theta\right) \tag{3.8}$$

Möchte man den gesuchten Zustand $|\nu\rangle$ mit einer Wahrscheinlichkeit von $\frac{1}{2}$ finden, so reicht eine Betrachtung von $k = \frac{1}{2}$. Soll hingegen $|\nu\rangle$ mit einer Wahrscheinlichkeit von 1 gefunden werden, muss $k = 1$ betrachtet werden. Für die Betrachtung von $k = 1$ ergibt sich die gleiche Laufzeit wie in Gleichung (3.6).

Gibt es nun t Lösungen, so verändert sich die Laufzeit des Algorithmus zu $\mathcal{O}\left(\sqrt{\frac{N}{t}}\right)$. Der Zustand $|S\rangle$ sieht dabei wie folgt aus:

$$|S(k,l)\rangle = \sum_{x\in A} k\,|x\rangle + \sum_{x\in B} l\,|x\rangle$$

$A \subseteq \{0,1,\ldots,N-1\}$ beschreibt die Menge aller Zustände, die in der Liste gesucht werden, die also $C(x) = 1$ erfüllen. Da t Lösungen existieren, gilt $|A| = t$. Für B gilt hingegen $B = \{0,1,\ldots,N-1\} \setminus A$. Somit beinhaltet B alle Zustände, die nicht durch den Grover-Algorithmus gesucht werden, für die also $C(x) = 0$ gilt.

Durch die Normierungsbedingung gilt in dem Fall mit t Lösungen $tk^2 + (N-t)l^2 = 1$. Die Werte für k und l nach j Iterationen ergeben sich analog zu dem Grover-Algorithmus mit einer gesuchten Lösung. Sie lassen sich durch Gleichung (3.9) und Gleichung (3.10) berechnen. Hierbei muss beachtet werden, dass sich θ zu $\theta = \arcsin\sqrt{\frac{t}{N}}$ ändert.

$$k_j = \frac{1}{\sqrt{t}} \sin\left((2j+1)\theta\right) \tag{3.9}$$

$$l_j = \frac{1}{\sqrt{N-t}} \cos\left((2j+1)\theta\right) \tag{3.10}$$

Nun kann man ebenso wie in Gleichung (3.6) die Laufzeit berechnen und erhält $\mathcal{O}\left(\sqrt{\frac{N}{t}}\right)$. Die genaue Formel ist in Gleichung (3.11) angegeben.

$$\frac{\pi}{2} = (1+2x)\arcsin\sqrt{\frac{t}{N}} \quad \Leftrightarrow \quad x = \frac{\pi}{4}\frac{1}{\arcsin\sqrt{\frac{t}{N}}} - \frac{1}{2} \tag{3.11}$$

$$\Rightarrow \quad x \approx \frac{\pi}{4}\sqrt{\frac{N}{t}} - \frac{1}{2}$$

3.4.3. Beispiel für einen Grover-Algorithmus mit zwei Qubits

Man betrachte nun ein Beispiel des Grover-Algorithmus für den Fall mit zwei Qubits. Dafür wird zur Veranschaulichung die rechte Darstellung aus Abb. 3.2, 3.3 und 3.4 verwendet. Die vier Basiszustände, die sich aus zwei Qubits bilden lassen, sind in diesem Fall $|00\rangle$, $|01\rangle$, $|10\rangle$ und $|11\rangle$. Man nehme an, dass unter diesen vier Zuständen der Zustand $|10\rangle$ gesucht wird. Um diesen Zustand zu finden, wird im ersten Schritt des Grover-Algorithmus der Zustand

$$|S\rangle = \frac{1}{2} \cdot (|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

hergestellt. Dies lässt sich wie in Abb. 3.5 visualisieren.

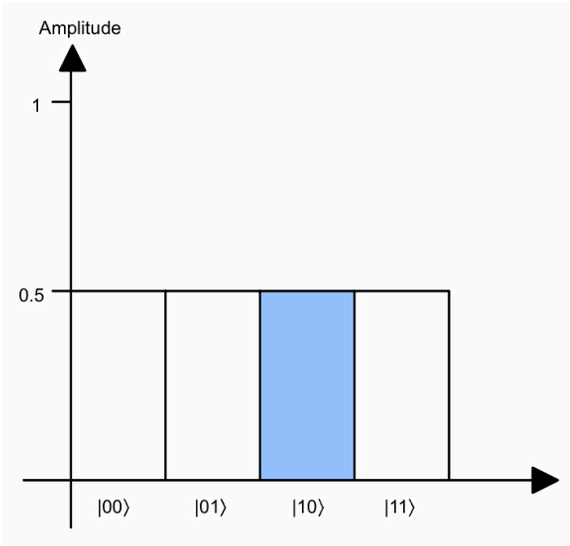


Abbildung 3.5.: Für ein Beispiel mit zwei Qubits, also vier Zuständen, wird hier der Zustand $|S\rangle$ aus dem Grover-Algorithmus dargestellt. Alle vier Zustände haben eine Amplitude von $\frac{1}{\sqrt{4}} = \frac{1}{2}$. Der gesuchte Zustand $|10\rangle$ ist blau markiert.

Im zweiten Schritt wird zuerst das Orakel angewandt und dabei das Vorzeichen der Amplitude des Zustandes $|10\rangle$ gewechselt. Daraus entsteht der Zustand

$$\begin{aligned}|\psi_S\rangle = U_{10}\,|S\rangle &= U_{10} \cdot \frac{1}{2} \cdot (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \\ &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\ &= \frac{1}{2} \cdot (|00\rangle + |01\rangle - |10\rangle + |11\rangle).\end{aligned}$$

Grafisch kann man sich die Anwendung des Orakels, wie in Abb. 3.6 dargestellt, als eine Reflexion des gesuchten Zustandes an der Abszisse vorstellen.

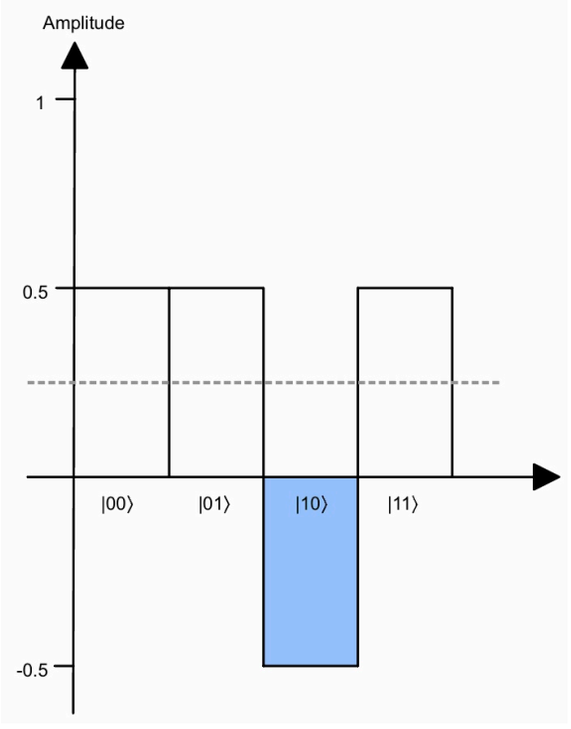


Abbildung 3.6.: Durch Anwendung des Orakels wird der gesuchte Zustand $|10\rangle$ (blau) an der Abszisse reflektiert. Der Mittelwert der neuen Amplituden (grau gestrichelt) liegt bei $\frac{1}{4}$.

Nach der Anwendung des Orakels kann der Mittelwert der Amplituden berechnet werden. Dieser liegt bei $\frac{1}{4}$. Er wurde in Abb. 3.6 grau eingezeichnet.

Im zweiten Teil des zweiten Schrittes wird der Diffusionstransformationsoperator angewandt. Dabei folgt:

$$\begin{aligned} D\left|\psi_S\right\rangle = \left|\psi'_S\right\rangle &= \left(2\left|S\right\rangle\left\langle S\right| - \mathbb{1}\right) \cdot \frac{1}{2} \cdot \left(\left|00\right\rangle + \left|01\right\rangle - \left|10\right\rangle + \left|11\right\rangle\right) \\ &= \frac{1}{4}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \\ &= \left|10\right\rangle \end{aligned}$$

Der neue Zustand entspricht dem gesuchten Zustand $|10\rangle$. Dies sieht man auch in der Abb. 3.7.

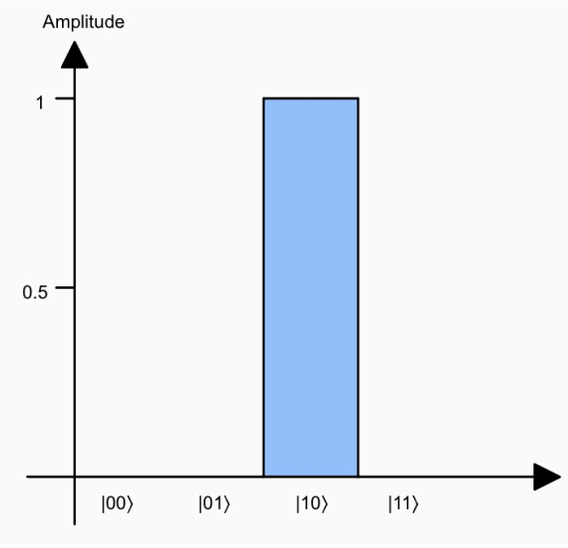


Abbildung 3.7.: Nach Anwendung des Diffusionstransformationsoperators nimmt die Amplitude des Zustandes $|10\rangle$ den Wert 1 an. Alle weiteren Amplituden liegen bei 0. Die Messwahrscheinlichkeit des Zustandes $|10\rangle$ beträgt somit 1.

Es wurde also nach nur einer Iteration des Grover-Algorithmus eine Lösung für das Problem gefunden. Die einmalige Iteration lässt sich auch mit der oben beschriebenen Gleichung (3.6), welche die Anzahl an Iterationen rechnerisch herausfindet, nachvollziehen. Dabei gilt:

$$\frac{\pi}{4}\frac{1}{\arcsin\frac{1}{\sqrt{N}}} - \frac{1}{2} \overset{N=4}{=} \frac{\pi}{4} \cdot \frac{6}{\pi} - \frac{1}{2} = 1$$

Es wird auch der Rechnung nach nur ein einzelner Iterationsschritt benötigt.

Nun muss der Zustand nur noch gemessen werden. Dabei ergibt sich, dass der gemessene Zustand mit einer Wahrscheinlichkeit von 1 dem gesuchten Zustand entspricht. Der Grover-Algorithmus hat somit seine Aufgabe, den gesuchten Zustand zu finden, erfüllt.

Es wurde hierbei ein Beispiel für einen Grover-Algorithmus gegeben. Dabei kann man den ersten Schritt durch Hadamard-Gatter und den Schritt 2.b. durch eine Kombination aus Rotationsgatter und Hadamard-Gattern implementieren. Nur das Orakel muss fallspezifisch gewählt werden. Dabei wird ein kontrolliertes Pauli-Z-Gatter eingebaut. Dieses wechselt genau dann das Vorzeichen des Zustandes, falls eine bestimmte Bedingung erfüllt ist. Das Orakel sieht allerdings bei jeder Implementierung anders aus, da je nach Fall andere Bedingungen erfüllt werden müssen. In dem eben betrachteten Beispiel wurde nach dem Zustand $|10\rangle$ gesucht. Dabei kann das Orakel wie in Abb. 3.8 aussehen [1].

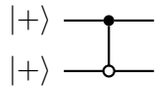


Abbildung 3.8.: Beispiel eines Orakels nach [1]. Es soll das das Vorzeichen des Zustandes $|10\rangle$ umdrehen. Da hierbei kein bestimmter Wert des Datenregisters gesucht wird, ist ein Zugriff auf dieses wie in Abb. 3.1 nicht nötig.

In dem Schritt 2.b. wird der Diffusionstransformationsoperator benötigt. Das Rotationsgatter als Teil des Diffusionstransformationsoperators wird in Abb. 3.9 abgebildet. Ziel ist es, das Vorzeichen aller Zustände bis auf das des Zustandes $|00\rangle$ zu ändern. Dabei wird zuerst jeweils ein Pauli-Z-Gatter auf die Qubits angewandt, wodurch die Zustände $|01\rangle$ und $|10\rangle$ ihr Vorzeichen ändern [19]. Da noch das Vorzeichen des Zustandes $|11\rangle$ gewechselt werden muss, wird ein kontrolliertes Pauli-Z-Gatter eingebaut [19]. Somit werden die Vorzeichen aller Zustände, bis auf das des Zustandes $|00\rangle$, durch die Rotationsmatrix gewechselt.

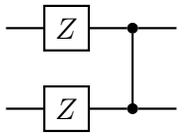


Abbildung 3.9.: Abgebildet ist wie das Rotationsgatter für zwei Qubits in einem Quantenschaltkreis dargestellt werden kann (nach [19]).

Setzt man alle Gatter zusammen, so ergibt sich für das Beispiel der Quantenschaltkreis in Abb. 3.10.

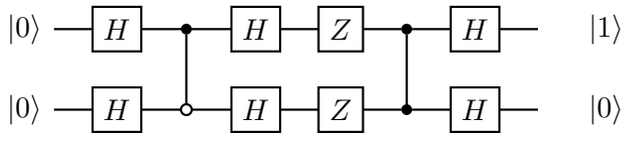


Abbildung 3.10.: Beispiel eines Quantenschaltkreises für den Grover-Algorithmus für zwei Qubits.

3.5. Grundlagen für den Quantenalgorithmus des TSP

Der Quantenalgorithmus, der das TSP lösen soll, hat die Aufgabe, den minimalen Weg über n Ecken zu finden. Dafür muss das Minimum einer Liste gefunden werden. Bei einer Suche nach bestimmten Elementen in einer Liste kann der Grover-Algorithmus angewandt werden. Der Grover-Algorithmus benötigt bei einer Liste mit N Einträgen $\mathcal{O}(\sqrt{N})$ Schritte bis zum Auffinden dieser Lösung. Wird allerdings das Minimum einer Liste gesucht, gestaltet sich die Suche mit Hilfe des Grover-Algorithmus nicht so einfach wie das Suchen nach einem bestimmten Listeneintrag, da das Minimum nicht von Anfang an bekannt ist. Stattdessen müssen die einzelnen Listeneinträge miteinander verglichen werden, damit man das Minimum identifizieren kann. Der Algorithmus für die Minimumssuche arbeitet damit, dass er einen beliebigen Wert aus der Liste aussucht und im Anschluss daran nach kleineren Listeneinträgen sucht. Die Suche ist beendet, wenn eine bestimmte Laufzeit überschritten wird. Da nicht bekannt ist, wie viele Listeneinträge kleiner als ein beliebiger Wert sind, wird die Grover-Suche mit einer unbekannten Anzahl an Lösungen benötigt. Sowohl der Algorithmus zur Minimumssuche als auch die Grover-Suche mit einer unbekannten Anzahl an Lösungen werden im Folgenden beschrieben. Ein Beispiel für beide Algorithmen findet sich im Anhang A.3 wieder. Für die Minimumssuche wird außerdem ein Abrufen der Informationen mit Hilfe des qRAM benötigt. Dieses wird in Abschnitt 3.5.3 erläutert.

3.5.1. Quantenalgorithmus zur Minimumssuche

Dürr und Hoyer beschreiben in ihrem Paper [20] einen Quantenalgorithmus, der das Minimum einer Liste mit N Elementen finden soll. Dabei soll die richtige Lösung dieses Problems mit einer Wahrscheinlichkeit von $\frac{1}{2}$ gefunden werden. Auf diesem Paper beruht dieses Kapitel.

Sei $T[0\ 1 \ldots N-1]$ eine Liste, in der jeder Eintrag einen bestimmten Wert besitzt. Aufgabe des Algorithmus ist es, x zu finden, sodass $T[x]$ minimiert wird. Der Algorithmus läuft wie folgt ab:

1. Suche einen zufälligen Schwellenindex, der durch $0 \leq x \leq N-1$ beschrieben wird. Dabei kann jeder Schwellenindex mit der gleichen Wahrscheinlichkeit gewählt werden. Dieser Index ist dem Listeneintrag $T[x]$ zugeordnet, welcher den Schwellenwert bildet.
2. Der zweite Schritt des Algorithmus wird so lange wiederholt, bis eine Gesamtlaufzeit von $22.5\sqrt{N} + 1.4 \log_2^2 N$ überschritten wird. Wenn der Algorithmus unterbrochen wird, soll zum Schritt 2.c) gesprungen werden, um diesen noch durchzuführen.
   a) Initialisiere den Speicher als $\sum_{j=0}^{N-1} \frac{1}{N} |j\rangle |x\rangle$. Diese Form der Initialisierung dient dem qRAM, welches in Abschnitt 3.5.3 beschrieben wird.
   Markiere weiterhin alle Elemente j, für die $T[j] < T[x]$ gilt.

b) Wende die Grover-Suche für eine unbekannte Anzahl an Lösungen an, um einen der markierten Einträge ausfindig zu machen. Diese Suche wird in Abschnitt 3.5.2 beschrieben.

c) Sei x' die Lösung des Grover-Algorithmus aus 2.b). Falls $T[x'] < T[x]$ erfüllt wird, setze den Schwellenindex x zu x'.

3. Gebe x aus.

Der Algorithmus sucht bei jedem Durchlauf des zweiten Schrittes Werte, die unterhalb eines bestimmten Schwellenwertes liegen. Dies wird wiederholt, bis die Gesamtlaufzeit überschritten wird. Die Schritte 2.c) und 3) sind im Algorithmus für die Überprüfung, ob x' der neue Schwellenindex ist, und zur Ausgabe dieses Schwellenindex zuständig.

Betrachte nun die Laufzeit des Algorithmus. Die Initialisierung des Speichers in 2.a) benötigt $n = \log_2 N$ Schritte [20]. Wie im nächsten Abschnitt 3.5.2 beschrieben, benötigt die Grover-Suche mit einer unbekannten Anzahl an Lösungen für das Auffinden eines kleineren Wertes mit einer Wahrscheinlichkeit von $\frac{1}{2}$ $\frac{9}{2}\sqrt{\frac{N}{t}}$ Iterationen. Die Laufzeiten der Schritte 1., 2.c) und 3. werden hier, ebenso wie in [20], nicht berücksichtigt. Zur Berechnung der benötigten Laufzeit fehlt die Wahrscheinlichkeit $p(N, t)$, die angibt, wie wahrscheinlich es ist, dass das t-kleinste Element der Liste während der Grover-Suche ein Mal zum Schwellenwert wird. In [20] wird bewiesen, dass $p(N, t) = \frac{1}{t}$ gilt. Durch die Werte kann nun ein Erwartungswert für die Laufzeit der Schritte 2.a) und 2.b) berechnet werden. Dieser ergibt sich durch die Formeln (3.12) und (3.13).

$$\sum_{r=2}^{N} p(N, r) \cdot \log_2 N \leq \frac{7}{10} \log_2^2 N \tag{3.12}$$

$$\sum_{r=2}^{N} p(N, r) \cdot \frac{9}{2} \cdot \sqrt{\frac{N}{r-1}} \leq \frac{45}{4}\sqrt{N} \tag{3.13}$$

Man kann den Erwartungswert der Laufzeit nicht als Laufzeit des zweiten Schrittes einsetzen. Würde man dies tun, so würden längere Laufzeiten als der Erwartungswert nicht mitberücksichtigt werden, was die Erfolgswahrscheinlichkeit der Minimumssuche senkt. Somit kann eine Laufzeit betrachtet werden, die doppelt so groß ist wie der Erwartungswert dieser [20]. Dadurch ergibt sich eine Laufzeit von $22.5\sqrt{N} + 1.4 \log_2^2 N$. Das Minimum wird hierbei mit einer Wahrscheinlichkeit von $\frac{1}{2}$ gefunden.

Möchte man die Erfolgswahrscheinlichkeit erhöhen, so kann der Algorithmus mehrfach ausgeführt werden. Nach c Ausführungen des Algorithmus liegt die Erfolgswahrscheinlichkeit bei $1 - \frac{1}{2^c}$ [20].

Es folgt, dass der Algorithmus in einer Laufzeit von $\mathcal{O}(\sqrt{N})$ das Minimum der Liste findet [20].

3.5.2. Grover-Algorithmus für eine unbekannte Anzahl an Lösungen

Betrachtet man den Grover-Algorithmus aus Abschnitt 3.4, so ergibt sich, dass bei der Suche nach einer einzelnen Lösung etwa $\frac{\pi}{4}\sqrt{N}$ Iterationen durchgeführt werden müssen, um eine Lösung zu erhalten. Liegen hingegen vier Lösungen vor, sind nur etwa halb so viele Iterationen nötig, und zwar etwa $\frac{\pi}{8}\sqrt{N}$. Würde man unter 2^{20} Elementen eine einzige Lösung suchen, so bräuchte man 804 Iterationen [17]. Gäbe es stattdessen vier Lösungen und man würde 804 Iterationen durchführen, so läge die Wahrscheinlichkeit, dass man das richtige Element findet, niedriger als eins zu einer Millionen [17]. In dem Algorithmus aus Abschnitt 3.5.1 ist allerdings unbekannt, wie viele Elemente der Liste kleiner sind als der Schwellenwert. Aufgrund dessen muss ein neuer Algorithmus genutzt werden, um die Grover-Suche für eine unbekannte Anzahl an Lösungen durchzuführen. Dieser wurde von Boyer et al. in [17] beschrieben und auf diesem Paper beruht dieses Kapitel.

Betrachte für den Algorithmus den Schwellenindex x und die Anzahl t an kleineren Werten als der Schwellenwert mit $1 \leq t \leq \frac{3}{4}N$. Für $t = 0$ wird der Algorithmus durch die begrenzte Laufzeit im Algorithmus aus Abschnitt 3.5.1 abgebrochen. Für $t > \frac{3}{4}N$ kann ein klassisches Sampling in einer konstanten Zeit verwendet werden, um auch diesen Fall auf den Fall $1 \leq t \leq \frac{3}{4}N$ zu reduzieren. Nun läuft der Algorithmus wie folgt ab:

1. Initialisiere $m = 1$ und $\lambda = \frac{6}{5}$. Hierbei würden auch andere Werte $1 < \lambda < \frac{4}{3}$ funktionieren.
2. Wähle $0 \leq j \leq m - 1$ mit $j \in \mathbb{N}$ zufällig aus.
3. Wende j Mal den Grover-Algorithmus auf $|S\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$ an.
4. Sei i der durch den Grover-Algorithmus gefundene Index.
5. Überprüfe, ob $T[i] < T[x]$ gilt. Falls ja, dann beende hier den Algorithmus.
6. Falls $T[i] < T[x]$ nicht erfüllt ist, setze m zu $\min(\lambda m, \sqrt{N})$. Kehre zu Schritt 2. zurück.

Der Algorithmus baut auf folgender Feststellung auf: Sei t die unbekannte Anzahl an Lösungen. Sei weiterhin m eine willkürliche Zahl mit $m \in \mathbb{N}$. Werden auf

$$|S\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

j Iterationen des Grover-Algorithmus mit $0 \leq j \leq m - 1$ durchgeführt, so liegt die Erfolgswahrscheinlichkeit des Algorithmus bei

$$P_m = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}$$

[17]. Dabei gilt $\theta = \arcsin\sqrt{\frac{t}{N}}$. Es folgt [17]:

$$m \geq \frac{1}{\sin(2\theta)} \Rightarrow P_m \geq \frac{1}{4} \tag{3.14}$$

Boyer et al. arbeiten hierbei mit einer anderen Erfolgswahrscheinlichkeit als diese Arbeit. Ihre Erfolgswahrscheinlichkeit liegt bei $P_{m_0} \geq \frac{1}{4}$ [17], während in dieser Arbeit mit einer Erfolgswahrscheinlichkeit von $P_{m_0} \geq \frac{1}{2}$ gearbeitet wird. Aufgrund dessen wird die Laufzeit des Algorithmus im Vergleich zu Boyer et al. verdoppelt. m_0 entspricht der hier genutzten Anzahl an Grover-Iterationen und $m_0' = \frac{m_0}{2}$ der von Boyer et al. genutzten Anzahl. Die folgende Rechnung basiert auf der Rechnung aus dem Paper [17], weswegen m_0' bestimmt wird.

Definiere $m_0' = \frac{1}{\sin(2\theta)}$. Dabei gilt

$$m_0' = \frac{m_0}{2} = \frac{1}{\sin(2\theta)} = \frac{N}{2\sqrt{(N-t)t}} < \sqrt{\frac{N}{t}}.$$

Nun stellen Boyer et al. fest, dass während der s-ten Iteration des Algorithmus m den Wert $m = \lambda^{s-1}$ annimmt. Da j in dem Intervall $[0, m-1]$ ausgesucht wird, liegt der Erwartungswert bei $\frac{m-1}{2} \approx \frac{m}{2}$. Nun definieren Boyer et al. einen sogenannten kritischen Zustand, falls der Algorithmus mehr als $\lceil \log_\lambda m_0' \rceil$ Iterationen durchläuft. Oberhalb des kritischen Zustandes kann die Erfolgswahrscheinlichkeit eines Iterationsschrittes nämlich angepasst werden. Bis zum Erreichen des kritischen Zustandes werden maximal

$$\frac{1}{2}\sum_{s=1}^{\lceil \log_\lambda m_0' \rceil} \lambda^{s-1} < \frac{1}{2}\frac{\lambda}{\lambda - 1} m_0' = 3m_0' \tag{3.15}$$

Grover-Iterationen erwartet. Diese Anzahl berechnet sich aus der Summation der Erwartungswerte von j. Da j bei jeder Iteration einen Erwartungswert von $\frac{m}{2} = \frac{\lambda^{s-1}}{2}$ annimmt, wurde in Gleichung (3.15) über alle s summiert.

Es ergibt sich durch Gleichung (3.14), dass für alle $m \geq m_0'$ $P_m \geq \frac{1}{4}$ gilt. Somit kann über alle Wiederholungen des Algorithmus oberhalb der kritischen Phase summiert werden, indem eine Erfolgswahrscheinlichkeit von $P_m \geq \frac{1}{4}$ genutzt wird. Daraus ergibt sich ein oberes Limit für die Anzahl an Grover-Iterationen, um daraus die Laufzeit zu berechnen. Gleichung (3.16) berechnet diese maximale Anzahl an Grover-Iterationen oberhalb der kritischen Phase, also für $m \geq m_0'$.

$$\frac{1}{2}\sum_{u=0}^{\infty} \frac{3^u}{4^{u+1}} \lambda^{u+\lceil \log_\lambda m_0' \rceil} < \frac{\lambda}{8 - 6\lambda} m_0' = \frac{3}{2} m_0' \tag{3.16}$$

$\frac{3^u}{4^u}$ beschreibt die Wahrscheinlichkeit, dass die u-te Wiederholung des obigen Algorithmus nach dem Erreichen der kritischen Phase berechnet wird, und $\frac{1}{4}$, ob diese zum Erfolg führt. Die erwartete Anzahl an Grover-Iterationen pro Iteration des obigen Algorithmus entspricht dabei $j = \frac{m}{2} = \frac{\lambda^{u+\lceil \log_\lambda m_0' \rceil}}{2}$. Im Vergleich zu s aus Gleichung (3.15) entspricht u hierbei $u = s - \lceil \log_\lambda m_0' \rceil - 1$.

Insgesamt ergibt sich durch Gleichung (3.15) und (3.16) ein oberes Limit von $\frac{9}{2}m_0'$ Grover-Iterationen, um die gesuchte Lösung mit einer Wahrscheinlichkeit von $\frac{1}{4}$ zu finden. Für eine Erfolgswahrscheinlichkeit von $\frac{1}{2}$ folgt somit eine Durchführung von bis zu $2 \cdot \frac{9}{2}m_0' = \frac{9}{2}m_0$ Grover-Iterationen. Es folgt für $t \ll N$ $\frac{9}{2}m_0 \approx \frac{9}{2}\sqrt{\frac{N}{t}}$. Das ist die Anzahl an Rechenschritten, die auch in Abschnitt 3.5.1 benutzt wurde.

3.5.3. qRAM

Für die Quantenminimumssuche ist es nötig, dass Listen mit zugehörigen Werten abgerufen werden. Hierzu wird der Direktzugriffspeicher benötigt. Dieser kann auf zwei verschiedenen Wegen genutzt werden, auf dem klassischen oder dem quantenmechanischen Weg. Der klassische Direktzugriffspeicher nennt sich auch RAM, also **r**andom **a**ccess **m**emory [21]. Er besteht aus zwei Teilen: der zentralen Verarbeitungseinheit, also der CPU, und dem Speicher mit einer Länge von $N \cdot l$ [1]. $N = 2^n$ beschreibt die Länge der Liste, also die Anzahl an Elementen, die sie beinhaltet [1]. Die Elemente $d_1, ..., d_N$ in der Liste sind im Speicherarray gespeichert [21]. Jedes Element dieses Arrays hat dabei eine Länge von l [1]. Die CPU hingegen verarbeitet die Informationen aus dem Speicher. Dabei stehen der CPU zwei Möglichkeiten zur Verfügung. Die CPU kann die Informationen aus dem Speicher abrufen oder neue Informationen im Speicher abspeichern (vgl. Abb. 3.11) [1].

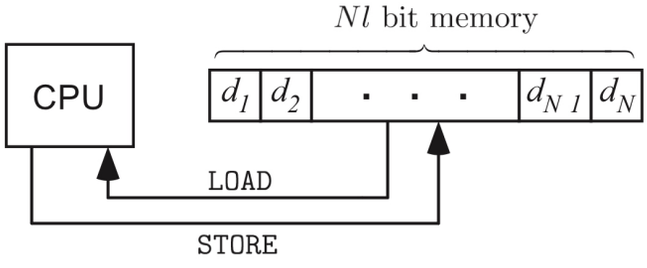


Abbildung 3.11.: In dem schematischen Aufbau des RAM aus [1] ist dargestellt, dass die CPU die Inhalte des Speichers lesen und Inhalte im Speicher speichern kann. Der Speicher enthält N Einträge $d_1, \ldots, d_N$. Jeder Eintrag benötigt l Bits zum Speichern.

Zum Laden der Speicherinhalte werden neben dem Speicherarray das Adressregister und das Ausgangsregister benötigt [21]. Das Adressregister adressiert die gesuchte Zelle in dem Speicher [21]. Jedes Element aus dem Speicher ist dabei eindeutig einer Adresse aus dem Adressregister zugeordnet. Im Ausgangsregister werden die Inhalte des Speicherarrays zwischengespeichert und am Output ausgegeben [21].

Der quantenmechanische Direktzugriffspeicher funktioniert analog zum RAM. Dabei handelt es sich um den qRAM, also den **q**uantum **r**andom **a**ccess **m**emory [21]. Dieser wird im Folgenden auf Basis des Papers von Giovanetti et al. [21] beschrieben.

Der Unterschied zwischen dem qRAM und dem RAM besteht daraus, dass sowohl das Adress- als auch das Ausgangsregister des qRAMs aus Qubits besteht. Der Speicher kann dabei sowohl klassisch als auch quantenmechanisch sein [1, 21]. Durch einen quantenmechanischen Zugriff auf den Speicher ist es möglich, durch Superposition mehrere Speicherinhalte zeitgleich abzurufen. Dabei wird der folgende Zustand injiziert [1, 21]:

$$\sum_j \phi_j \left|j\right\rangle_a \left|0\right\rangle_D$$

Der Index a markiert hierbei das Adressregister und der Index D das Ausgangsregister. Durch den Ladevorgang des qRAM wird dieser Zustand zu

$$\sum_j \phi_j \left|j\right\rangle_a \left|d_j\right\rangle_D$$

verändert. $|d_j\rangle$ beschreibt hierbei den Inhalt des Speicherarrays, der adressiert wird. Da in dem Ausgangsregister die Inhalte der Liste abgespeichert werden, kann man dieses Register auch als das Datenregister bezeichnen [1]. Für den Speicherabruf adressiert $|j\rangle$ die Einträge des Speichers, die abgerufen werden sollen. Dies wird in Abb. 3.12 in Form eines Baumdiagrammes modelliert. Dabei stellt der rote Punkt den Wurzelknoten dar. An diesem wird das Adressregister implementiert. Da Informationen in einem Computer in 0en und 1en verarbeitet werden, wird auch das Adressregister in 0en und 1en implementiert. Die Anordnung der 0en und 1en definiert dabei den Weg, der im Baumdiagramm zurückgelegt wird. Ist das k-te Bit des Adressregisters im Zustand $|0\rangle$, so wird an der k-ten Ebene der linke Pfad gewählt. Handelt es sich um den Zustand $|1\rangle$, wird der rechte Pfad genommen. Die n Qubits des Adressregisters steuern somit n Quantensteuerleitungen, die einen bestimmten Pfad im Baumdiagramm kennzeichnen. Da hier quantenmechanisch gearbeitet wird, ist auch eine Superposition mehrerer Pfade möglich.

Nun wird ein sogenannter „Quantenbus“ als $|0\rangle_D$ injiziert. Dieser folgt dem gekennzeichneten Pfad bis zu dem adressierten Speicherort, wo der innere Zustand des Busses durch beispielsweise ein CNOT-Gatter geändert wird. Nun wird der Quantenbus auf dem gleichen Weg zurück zum Wurzelknoten geleitet. Anhand des inneren Zustandes des Busses kann der Inhalt des Speichers abgelesen werden. Allerdings sind in der Praxis die Adressqubits mit $\mathcal{O}(N)$ Schaltern, die sich im Baumdiagramm an den Verzweigungspunkten befinden und die Richtung des Weges vorgeben, verschränkt, weswegen der qRAM keinen geeigneten Vorteil gegenüber dem RAM verspricht [1, 21]. Stattdessen kann man die Schalter durch Qutrits implementieren. Qutrits ähneln Qubits, haben jedoch drei mögliche Zustände, die sie annehmen können. Die Idee der Implementierung von Qutrits wird auch die „Bucket-Brigade“ genannt. Hierbei befinden sich die Schalter in einem Warte-Zustand so lange sie nicht durch Qubits zu einer Zustandsänderung angeregt

werden. Diese Idee arbeitet mit $\mathcal{O}(\log_2 N)$ statt mit $\mathcal{O}(N)$ Quantenspeicherelementen, was eine bessere Laufzeit bei Benutzung dieser Idee gegenüber dem qRAM erwarten lässt. Zum Verständnis der Funktionsweise des qRAM reicht allerdings die beschriebene Funktionsweise.

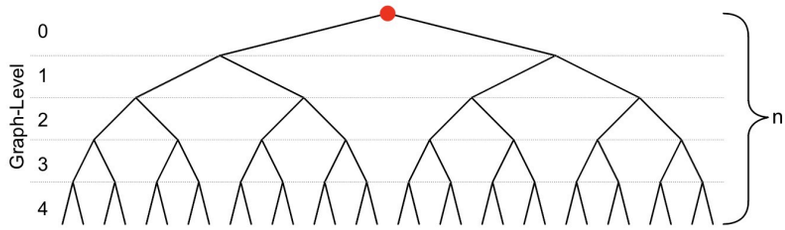


Abbildung 3.12.: Abgebildet ist ein Modell für den Speicherabruf des qRAM nach [21]. Am roten Punkt wird der „Quantenbus“ injiziert. Gibt das Adressregister beispielsweise den Zustand $|00\ldots0\rangle$ vor, wird der „Quantenbus“ zum linken Speicherort geleitet und von dort kann der Speicher ausgelesen werden.

Ein Beispiel eines Quantenschaltkreises für den qRAM wird in Abschnitt 5.3 beschrieben.

4 Kapitel 4. Quantenalgorithmus für das Travelling Salesman Problem

In diesem Kapitel wird ein Quantenalgorithmus für das TSP vorgestellt. In Abschnitt 3.2 wurde beschrieben, dass die dynamische Programmierung eine schnellere Lösung für das TSP liefert, als eine zufällige Suche nach einer Lösung. Bei der dynamischen Programmierung handelt es sich um einen klassischen Algorithmus von Held und Karp, der in Abschnitt 4.1 vorgestellt wird. Der Quantenalgorithmus von Ambainis et al. baut auf diesem auf und verspricht eine im Vergleich bessere Laufzeit. Er wird in Abschnitt 4.2 vorgestellt. Dabei ergeben sich bei dem Algorithmus nach Ambainis et al. [2] Probleme, die in Abschnitt 4.3 betrachtet und behoben werden sollen. Außerdem wird in Abschnitt 4.4 betrachtet, wieso der Algorithmus in der von Ambainis et al. [2] vorgestellten Form die schnellste Laufzeit erreicht.

4.1. Klassische dynamische Programmierung für das TSP

Die Suche nach einer optimalen Lösung für das TSP würde durch Herumprobieren eine Laufzeit von $\mathcal{O}(n!)$ benötigen. Die klassische dynamische Programmierung nach Held und Karp [4] stellt eine Methode dar, welche die Laufzeit auf $\mathcal{O}(n^2 2^n \log_2 L)$ verringert [2, 4]. Wie diese geringe Zeit zu Stande kommt, wird in diesem Abschnitt erläutert. Bei der dynamischen Programmierung handelt es sich um den bisher schnellsten klassischen Algorithmus zur Lösung des TSP [2]. Dies gilt aber nur, falls das Kantengewicht, also beispielsweise die Länge der Wege zwischen zwei Städten, nicht polylogarithmisch mit n skaliert [2]. Der Algorithmus von Björklund hat beispielsweise eine Laufzeit von $\mathcal{O}^*(1.657^n L)$, weswegen er im Falle einer polylogarithmischen Skalierung von n zu einer besseren Laufzeit führt als der Algorithmus von Held und Karp [2]. Im Folgenden wird allerdings nur der Algorithmus zur dynamischen Programmierung betrachtet. Dabei basiert dieses Kapitel auf dem Paper [4] von Held und Karp.

Betrachte n Städte, zwischen denen die kürzeste Strecke gefunden werden soll. Dabei bilden die Städte die Menge $\{1, 2, \ldots, n\}$. Die Lösung des TSP ist unabhängig davon, an welcher Stadt der Weg begonnen wird, denn sowohl der Weg $1 \to 2 \to \cdots \to n \to 1$ als auch $2 \to \cdots \to n \to 1 \to 2$ ergeben dieselbe Entfernung. Somit wird die Ecke 1 als Startpunkt festgelegt, um die mehrfache Berechnung derselben Entfernung zu vermeiden. Die Entfernung zwischen zwei Städten $i \to j$ wird als a_{ij} angegeben. Nun soll $C(S, l)$ die minimale Strecke von $1 \to l$ durch alle Elemente in $S \subseteq \{2, 3, \ldots, n\}$ beschreiben. Dabei gilt $l \in S$ und $|S| = n(S)$. Die dynamische Programmierung berechnet im ersten Schritt den kürzesten Weg über eine Stadt und nimmt in jedem Schritt eine weitere Stadt hinzu.

Beim Hinzufügen einer weiteren Stadt wird der kürzeste Weg anhand des vorherigen Schrittes berechnet. Die Rechnung wird wie in Gleichung (4.1) und (4.2) durchgeführt.

$$1.\ \text{Für } n(S) = 1: \quad C(\{l\}, l) = a_{1l} \qquad \text{mit} \quad l \in \{2, 3, \ldots, n\} \tag{4.1}$$

$$2.\ \text{Für } n(S) > 1: \quad C(S, l) = \min_{\substack{l \in S \\ m \in S \setminus \{l\}}} \{C(S \setminus \{l\}, m) + a_{ml}\} \quad \text{mit } S \subseteq \{2, 3, \ldots, n\} \tag{4.2}$$

Der zweite Schritt kann bis $n(S) = n - 1$ laufen, da $|S| \leq n - 1$ gilt. Nun fehlt für die Berechnung des gesamten Weges noch die Strecke $l \to 1$. Der minimale Gesamtweg wird durch $\mathcal{C}$ angegeben. Im dritten und somit letzten Schritt wird diese gesamte Strecke berechnet. Diese ergibt sich durch Gleichung (4.3).

$$\mathcal{C} = \min_{l \in \{2,3,\ldots,n\}} \{C(\{2, 3, \ldots, n\}, l) + a_{l1}\} \tag{4.3}$$

Nun wurde zwar die Entfernung des Weges berechnet, es wurde aber nicht festgehalten, welcher Weg die eben berechnete minimale Entfernung besitzt. Dabei sei die Permutation $(1\ i_2\ i_3 \ldots i_n)$ die Lösung für das TSP. Um diese Permutation herauszufinden, berechnet man im ersten Schritt die Stadt $i_n \in \{2, \ldots, n\}$, indem man diese mit Gleichung (4.4) bestimmt.

$$\mathcal{C} = C(\{2, 3, \ldots, n\}, i_n) + a_{i_n 1} \tag{4.4}$$

Die p-te Stadt kann durch Gleichung (4.5) herausgefunden werden. Wenn die Rechnung die richtige Lösung ergibt, wurde i_p herausgefunden. Dabei gilt $2 \leq p \leq n - 1$ und $i_p \in \{2, \ldots, n\} \setminus \{i_{p+1}, \ldots, i_n\}$.

$$C(\{i_2, i_3, \ldots, i_p, i_{p+1}\}, i_{p+1}) = C(\{i_2, i_3, \ldots, i_p\}, i_p) + a_{i_p i_{p+1}} \tag{4.5}$$

Ein Beispiel für diesen Algorithmus wird im Anhang A.1 beschrieben.

Die Laufzeit des Algorithmus kann durch die Anzahl an Additionen, die während des Algorithmus gerechnet werden, angegeben werden. Die Gleichung (4.2) berechnet die Entfernung für $n - 2$ verschiedene Größen von S. Für jede dieser Größen gibt es $\binom{|\{2,3,\ldots,n\}|}{n(S)} = \binom{n-1}{n(S)}$ verschiedene Möglichkeiten, die Menge S zusammenzusetzen. Zusätzlich gibt es $n(S)$ Möglichkeiten, die Stadt l aus Gleichung (4.2) zu wählen, an der der berechnete Weg endet. Somit gibt es pro Schritt $n(S) \cdot \binom{n-1}{n(S)}$ verschiedene Kombinationen für $C(S, l)$. In jeder Berechnung von $C(S, l)$ gibt es $n(S) - 1$ Möglichkeiten, m aus Gleichung (4.2) auszuwählen. Somit folgen $n(S) \cdot (n(S) - 1) \cdot \binom{n-1}{n(S)}$ Additionen für $2 \leq n(S) \leq n - 1$ mit Gleichung (4.2). Für $n(S) = 1$ finden keine Additionen statt, da Gleichung (4.1) keine Additionen vollführt. Im letzten Schritt mit Gleichung (4.3) gibt es $n - 1$ Möglichkeiten für die Wahl von l, wodurch $n - 1$ Additionen durchgeführt werden. Gleichung (4.6) gibt die gesamte Anzahl an Additionen für das Herausfinden der kürzesten Weglänge an.

$$\left(\sum_{k=2}^{n-1} k \cdot (k-1) \cdot \binom{n-1}{k}\right) + (n-1) = (n-1) \cdot (n-2) \cdot 2^{n-3} + (n-1) \tag{4.6}$$

Es wird, wie in Gleichung (4.6) berechnet, eine Laufzeit von

$$\mathcal{O}\left((n-1)\cdot(n-2)\cdot 2^{n-3}+(n-1)\right)=\mathcal{O}(n^2 2^n)$$

erreicht. Der Addierer, der in den Gleichungen (4.1), (4.2) und (4.3) angewandt werden muss, hat auch eine Laufzeit. Diese liegt bei $\mathcal{O}(\log_2 L)$, wobei $L = \max\limits_{\substack{\{i,j\}\in S\\ i\neq j}}\{a_{ij}\}$ die maximale Entfernung zwischen zwei Städten beschreibt [2]. Diese Laufzeit ergibt sich aus der bitweisen Darstellung der Daten, da jede Entfernung durch $\log_2 L$ Bits dargestellt wird. Es ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n^2 2^n \log_2 L)$ [2].

Im zweiten Teil des Algorithmus, in dem die Permutation herausgefunden wird, werden

$$\sum_{k=2}^{n-1} k = \frac{n\cdot(n-1)}{2}-1$$

Rechnungen durchgeführt. Auch hier wird die Laufzeit des Addierers mitberücksichtigt. Somit liegt die Laufzeit bei $\mathcal{O}(n^2 \log_2 L)$.

Möchte man die Speicheranforderung wie Bellman in seinem Paper [15] für einen Schritt des Algorithmus bestimmen, so kann man die Anzahl an Kombinationsmöglichkeiten für $C(S,l)$ für ein festes $n(S)$ betrachten. Die Anzahl an Kombinationsmöglichkeiten entspricht nämlich der Anzahl an Daten, die für einen Schritt gespeichert werden muss. Dies erschließt sich daraus, dass für jedes $C(S,l)$ eine Entfernung gespeichert wird. Im Folgenden wird dabei vernachlässigt, dass ein Ergebnis von $C(S,l)$ nicht immer einen Byte zum Speichern benötigt. Stattdessen soll nur ein Näherungswert für die Speicheranforderung gegeben werden, weswegen jedes $C(S,l)$ hier mit einem Byte zur Speicherung angegeben wird. Ein Byte entspricht 8 Bits, also einem Zeichen mit dem ASCII [12].

Die Kombinationsmöglichkeiten für $C(S,l)$ bei festem $n(S)$ sind durch $n(S)\cdot\binom{n-1}{n(S)}$ gegeben. Der Wert wird für $n(S)=\frac{n-1}{2}$ maximiert. Betrachtet man beispielsweise 41 Städte, ergeben sich für $n(S)=20$ ungefähr $2.757\cdot 10^{12}$ Kombinationsmöglichkeiten für $C(S,l)$. Dieser Zwischenschritt wird im RAM abgespeichert, damit mit dem Ergebnis weitergerechnet werden kann. Wenn man nun betrachtet, dass eine Kombinationsmöglichkeit einen Byte zum Speichern benötigt, ergeben sich ungefähr 2.757 TB zum Speichern der Ergebnisse. Als Vergleichswert hat der Mac mit dem bislang größten RAM von Apple einen RAM von ungefähr 1.5 TB [22]. Dabei handelt es sich um den Mac Pro aus dem Jahr 2019 [22]. Somit könnte dieser Mac den Zwischenschritt von 20 Städten für insgesamt 41 Städte nicht im Arbeitsspeicher abspeichern. Es gibt auch Computer mit einem besseren Arbeitsspeicher als diesem, allerdings wird hier nur ein Näherungswert betrachtet, welche Leistung ein klassischer Computer hervorbringen kann.

Insgesamt folgt, dass das Speichern des Schrittes $n(S)=20$ bei 41 Städten auf dem RAM des beschriebenen klassischen Computers nicht berechnet werden kann. Hierbei werden entweder größere RAM, Anpassungen des Algorithmus oder andere klassische Algorithmen benötigt. Allerdings hat der Algorithmus eine Laufzeit von $\mathcal{O}(n^2 2^n \log_2 L)$, was ihn zu dem bisher schnellsten klassischen Algorithmus macht [2].

Den klassischen Algorithmus von Held und Karp haben Ambainis et al. so angepasst, dass er sich auf Quantencomputern möglichst effizient und mit einer besseren Laufzeit als bei der ursprünglichen Version nutzen lassen kann. Dabei handelt es sich um den Quantenalgorithmus für das TSP auf Basis der dynamischen Programmierung.

4.2. Quantenalgorithmus für das TSP auf Basis der dynamischen Programmierung

Im Vergleich zum Algorithmus von Held und Karp kann der Quantenalgorithmus von Ambainis et al. [2] durch beispielsweise die Quantenminimumssuche aus Abschnitt 3.5.1 oder durch die Superposition der quantenmechanischen Zustände zu einem schnelleren Auffinden der Lösungen führen. Aufgrund dessen wird in diesem Abschnitt dieser Quantenalgorithmus für das TSP beschrieben. Der Algorithmus basiert auf dem Algorithmus der dynamischen Programmierung von Held und Karp. Dabei wurde der Quantenalgorithmus von Ambainis et al. [2] übernommen und auf diesem Paper basiert das folgende Kapitel. Die Grundlagen für den Algorithmus wurden bereits in Kapitel 3 behandelt.

Der Quantenalgorithmus baut auf zwei Formeln auf. Die Gleichung (4.7) funktioniert ähnlich zu den Gleichungen (4.1) und (4.2) aus Abschnitt 4.1. Bei dieser Gleichung wird zu einem berechneten Weg ein weiterer Ort hinzugefügt und somit der neue Weg berechnet. Mit der zweiten Gleichung (4.8) wird der gesamte Weg in zwei Teilwege geteilt, die addiert werden. Dabei wird das Minimum über alle möglichen Kombinationen von zwei Teilgruppen gesucht. Um die Teilwege zu berechnen, werden diese ebenfalls in zwei Teilwege geteilt. Dies geschieht rekursiv.

Sei $G = (V, E, w)$ der gegebene Graph, wobei V die Menge der Ecken, $|V| = n$ die Anzahl dieser Ecken, $E \subseteq V^2$ die Menge der Kanten zwischen zwei Ecken und $w : E \to \mathbb{N}$ das Kantengewicht beschreibt. Dabei gilt $w(u, v) = \infty$ falls $\{u, v\} \notin E$.

Sei $D = \{(S, u, v) \mid S \subseteq V, u, v \in S\}$. Definiere $f : D \to \mathbb{N}$ wie folgt: $f(S, u, v)$ ist der kürzeste Weg innerhalb des Graphen G, der in u beginnt, in v endet und jeden Eckpunkt aus S genau einmal besucht. Sei weiterhin $N(u)$ die Menge der nächsten Nachbarn von u in G.

$f(S, u, v)$ kann berechnet werden, indem an den bereits berechneten Weg $f(S \setminus \{u\}, t, v)$ die Ecke u hinzugefügt wird. u bildet dabei den neuen Startpunkt des Weges $f(S, u, v)$. Dies ist in Gleichung (4.7) dargestellt. In der Rechnung wird das Minimum über alle Möglichkeiten von t, die die nächsten Nachbarn von u sind, gesucht.

$$f(S, u, v) = \min_{\substack{t \in N(u) \cap S \\ t \neq v}} \{w(u, t) + f(S \setminus \{u\}, t, v)\}, \qquad f(\{v\}, v, v) = 0 \tag{4.7}$$

$f(S, u, v)$ kann alternativ rekursiv berechnet werden, wobei S in zwei Teilmengen geteilt wird. Sei $k \in [2, |S| - 1]$ fest gewählt. S wird in der rekursiven Berechnung in zwei Mengen der Größe k und $|S| - k + 1$ geteilt. Die Wege über diese beiden Teilmengen werden addiert, wodurch sich der Weg über die Menge S ergibt. Die Rechnung wird in Gleichung (4.8) dargestellt.

$$f(S,u,v) = \min_{\substack{X\subset S,|X|=k\\ u\in X, v\notin X}} \min_{\substack{t\in X\\ t\neq u}} \{f(X,u,t) + f((S\setminus X)\cup\{t\},t,v)\} \tag{4.8}$$

Ambainis et al. kombinieren die beiden Gleichungen zu einem Algorithmus [2]. Dieser führt Gleichung (4.7) durch, bis die einzelnen Wege über jeweils $|S| \leq \frac{(1-\alpha)n}{4}$ Ecken verlaufen. Im zweiten Schritt des Algorithmus wird Gleichung (4.8) genutzt. Dabei wird der Gesamtweg in zwei Teilwege geteilt, die über jeweils $|S| = \frac{n}{2}$ und $|(V\setminus S)\cup\{u,v\}| = \frac{n}{2}+2$ Ecken verlaufen. Die Wege dieser beiden Teilmengen werden addiert. Um die Wege der Teilmengen mit $|S| = \frac{n}{2}$ Ecken zu finden, werden diese ebenfalls in zwei Teilmengen der Größe $|X| = \frac{n}{4}$ und $|(S\setminus X)\cup\{t\}| = \frac{n}{4}+1$ geteilt und berechnet. Auch die Wege über die Teilmengen mit $|S| = \frac{n}{4}$ Ecken sind nicht bekannt, weswegen sie in zwei Teilwege über $|X| = \frac{\alpha n}{4}$ und $|(S\setminus X)\cup\{t\}| = \frac{(1-\alpha)n}{4}+1$ Ecken geteilt und addiert werden. Die Teilwege über $\frac{\alpha n}{4}$ und $\frac{(1-\alpha)n}{4}$ Ecken sind aus der Rechnung mit Gleichung (4.7) bekannt. Hierfür muss allerdings $\alpha \in \left(0, \frac{1}{2}\right]$ angenommen werden. Der Algorithmus sieht wie folgt aus:

Algorithm 1 Quantenalgorithmus für das Travelling Salesman Problem [2]

TravellingSalesman(Graph G, Kantengewichte w): Länge des kürzesten Hamiltonkreises.

1. Berechne die Werte von $f(S,u,v)$ für alle $|S| \leq \frac{(1-\alpha)n}{4}$ klassisch unter Benutzung der dynamischen Programmierung mit Gleichung (4.7) und speichere diese im Speicher.

2. Nutze die Quantenminimumsuche aus Abschnitt 3.5.1 über alle Teilmengen $S \subset V$ mit $|S| = \frac{n}{2}$, um die Lösung für

   $$\min_{\substack{S\subset V\\ |S|=n/2}} \min_{\substack{u,v\in S\\ u\neq v}} \{f(S,u,v) + f((V\setminus S)\cup\{u,v\},v,u)\}$$

   zu finden.
   Nutze Gleichung (4.8) für $k = \frac{n}{4}$ mit der Quantenminimumsuche, um $f(S,u,v)$ mit $|S| = \frac{n}{2}$ zu berechnen.
   Nutze Gleichung (4.8) für $k = \frac{\alpha n}{4}$ mit der Quantenminimumsuche, um $f(S,u,v)$ mit $|S| = \frac{n}{4}$ zu berechnen.
   Für jedes S mit $|S| = \frac{\alpha n}{4}$ oder $|S| = \frac{(1-\alpha)n}{4}$ ist das Ergebnis aus dem ersten Schritt bekannt.

Ambainis et al. geben eine Erfolgswahrscheinlichkeit von $\frac{2}{3}$ für diesen Algorithmus an. Die Quantenminimumssuche hat, wie in Abschnitt 3.5.1 beschrieben, eine Erfolgswahrscheinlichkeit von $\frac{1}{2}$. Eine zweite Anwendung der Quantenminimumssuche führt zu einer Erfolgswahrscheinlichkeit von $1 - \frac{1}{2^2} = \frac{3}{4}$. Außerdem kommt es durch die zweite Anwendung zu einer doppelten Laufzeit. In der Komplexitätstheorie gilt, dass

eine Verdoppelung der Laufzeit keine Auswirkung auf die Komplexität hat [23]. Daraus folgt, dass die Genauigkeit des Algorithmus keinen Einfluss auf die Komplexität der Laufzeit hat. Man kann durch mehrfache Wiederholung der einzelnen Algorithmen die Erfolgswahrscheinlichkeit verbessern.

Man nehme an, dass der Zugriff auf die Werte w eine Laufzeit von $poly(n)$ benötigt. Mit dieser Annahme hat die Rechnung von Ambainis ergeben, dass $\alpha \approx 0.055362$ die beste Laufzeit erzielt. Diese liegt bei $\mathcal{O}^*\left(1.727391^n\right)$. Das Erreichen des Wertes für α und die Berechnung der Laufzeit werden in Abschnitt 4.4 beschrieben. Weiterhin geben Ambainis et al. an, dass arithmetische Operationen die Laufzeit des Algorithmus um $\mathcal{O}(\log_2 L)$ erhöhen. Dies kommt daher, dass die Laufzeit einer Addition von der Größe der Zahlen abhängig ist. $L = \max\limits_{\{u,v\}\in E}\{w(u,v)\}$ beschreibt das maximale Kantengewicht, also beispielsweise die größte Entfernung zwischen zwei Städten. Die Darstellung der Kantengewichte im Binärsystem benötigt $\log_2 L$ Qubits. Somit folgt, dass auch die Addition zweier ganzer Zahlen eine Laufzeit von $\mathcal{O}(\log_2 L)$ benötigt. Dies kann beispielsweise an den Quantenschaltkreisen in Abschnitt 5.1 und Anhang A.4 beobachtet werden, indem man sich anschaut, wie sich die Größe der Quantenschaltkreise mit der Anzahl der einkommenden Qubits verhält. Es folgt eine Gesamtlaufzeit von $\mathcal{O}^*\left(1.727391^n \log_2 L\right)$ für den Quantenalgorithmus.

Damit der Algorithmus wie beschrieben funktioniert, wird in Abschnitt 4.3 eine Anpassung des Algorithmus vorgenommen.

4.3. Anpassung des Algorithmus

Beim Betrachten des Algorithmus von Ambanis et al., der in Abschnitt 4.2 beschrieben wurde, stellt sich heraus, dass der Algorithmus das Hinzufügen einiger Ecken nicht betrachtet. Im zweiten Schritt des Algorithmus addiert man zum Beispiel zwei Wege, die durch $f(S,u,v)$ und $f((V\setminus S)\cup\{u,v\},v,u)$ beschrieben werden. Die Wege setzen sich dabei aus jeweils $|S| = \frac{n}{2}$ und $|(V\setminus S)\cup\{u,v\}| = \frac{n}{2}+2$ Ecken zusammen. $|S| = \frac{n}{2}$ wird durch den Algorithmus vorgegeben und $|(V\setminus S)\cup\{u,v\}| = \frac{n}{2}+2$ berechnet sich daraus, dass $|V| = n$, $S \subset V$, also $|V\setminus S| = \frac{n}{2}$, und $u,v \in S$ gilt. Ein Weg über $\frac{n}{2}+2$ Ecken wurde in der vorherigen Rechnung nicht berechnet. Analoges gilt auch für die Wege über $\frac{n}{4}+1$ und $\frac{(1-\alpha)n}{4}+1$ Ecken. Somit wird eine Anpassung des Algorithmus vorgenommen, damit der Algorithmus richtig funktioniert.

Im Folgenden wird sich mit drei Anpassungsmöglichkeiten und welche dieser Anpassungen das TSP mit der besten Laufzeit lösen kann auseinandergesetzt. Bei den Anpassungen und der Laufzeitberechnung wird nur der zweite Schritt des Quantenalgorithmus betrachtet, da sich der erste Schritt, also der, welcher dem klassischen Algorithmus von Held und Karp ähnelt, nicht ändert. Nur der zweite Teil des Algorithmus von Ambainis et al. aus Abschnitt 4.2 soll hier optimiert werden.

4.3.1. Vorstellung der Änderungsmöglichkeiten

Die drei Vorschläge für eine Änderung des zweiten Schrittes des Algorithmus von Ambainis et al., die hier vorgestellt werden, passen die Größen der Teilmengen an, in die der Gesamtweg im Quantenalgorithmus geteilt wird. Dabei wird die Unterteilung der Wege mit $k = \frac{n}{2}$, $k = \frac{n}{4}$ und $k = \frac{\alpha n}{4}$ von Ambainis et al. im zweiten Schritt ihres Algorithmus im Folgenden zur besseren Übersicht mit a), b) und c) benannt. Die Vorschläge funktionieren wie folgt:

1. In der ersten Anpassung des Quantenalgorithmus wird die Anzahl an Ecken im Vergleich zu Ambainis et al. Algorithmus bei jedem Rechenschritt angepasst. Dabei wird der Weg bei den Schritten a) und b) jeweils halbiert. Der Schritt c) wird so angepasst, dass sich der erste Schritt des Quantenalgorithmus aus Abschnitt 4.2 nicht ändert. Es wird folgende Unterteilung vorgenommen:

   a) $\min\limits_{\substack{S\subset V\\ |S|=n/2+1}} \min\limits_{\substack{u,v\in S\\ u\neq v}} \{f(S,u,v) + f((V\setminus S)\cup\{u,v\},v,u)\}$

   b) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{2}+1$ mit Gleichung (4.8), wobei $k = \frac{n}{4}+1$ genutzt wird.

   c) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{4}+1$ mit Gleichung (4.8), wobei $k = \frac{\alpha n}{4}+2$ genutzt wird.
   Da im ersten Schritt bis $|S| \leq \frac{(1-\alpha)n}{4}$ gerechnet wird, also $|S|$ auf die nächste ganze Zahl abgerundet wird, wird k in diesem Fall auf die nächste ganze Zahl aufgerundet.

   Dieser Algorithmus funktioniert für $n \geq 13$, da in diesem Fall $f(S,u,v)$ mit $|S| = \lceil \frac{\alpha n}{4} + 2 \rceil$ aus dem ersten Schritt des Algorithmus bekannt ist. Für $n = 13$ gilt nämlich $\frac{(1-\alpha)n}{4} = 3.07$, weswegen mit dem ersten Schritt bis $|S| = 3$ gerechnet wird. Damit kann Gleichung (4.8) mit $k = \frac{\alpha n}{4} + 2 = 2.18$, also aufgerundet $k = 3$, berechnet werden.

   Für $n < 13$ muss hingegen in c) die Gleichung (4.7) anstatt Gleichung (4.8) angewandt werden.

2. Die zweite mögliche Anpassung des Quantenalgorithmus betrachtet keine Veränderung der Größe der Teilmengen S im zweiten Schritt. Stattdessen wird der Algorithmus von Ambainis et al. beibehalten und die Wege über $\frac{n}{4}+1$ und $\frac{(1-\alpha)n}{4}+1$ Ecken werden durch Gleichung (4.7) berechnet. Zur Berechnung des Weges über $\frac{n}{2}+2$ Ecken wird Gleichung (4.8) genutzt.

   a) $\min\limits_{\substack{S\subset V\\ |S|=n/2+2}} \min\limits_{\substack{u,v\in S\\ u\neq v}} \{f(S,u,v) + f((V\setminus S)\cup\{u,v\},v,u)\}$
   Berechne dafür $f(S,u,v)$ mit $|S| = \frac{n}{2}+2$ mit Gleichung (4.8), wobei $k = \frac{n}{2}$ genutzt wird.

   b) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{2}$ mit Gleichung (4.8), wobei $k = \frac{n}{4}+1$ genutzt wird.
   Berechne dafür $f(S,u,v)$ mit $|S| = \frac{n}{4}+1$ durch Anwenden von Gleichung (4.7).

c) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{4}$ mit Gleichung (4.8), wobei $k = \frac{\alpha n}{4} + 1$ genutzt wird.
Wie in 1.c) wird auch hier k aufgerundet.

Der beschriebene Änderungsvorschlag funktioniert für c) erst ab $n \geq 9$. Dies liegt daran, dass Gleichung (4.8) mit $k = \lceil \frac{\alpha n}{4} + 1 \rceil$ berechnet wird. Analog zu der Rechnung im ersten Änderungsvorschlag ergibt sich, dass dies nur für $n \geq 9$ möglich ist, da in diesem Fall der Weg mit $k = \lceil \frac{\alpha n}{4} + 1 \rceil$ im ersten Schritt berechnet wurde.

Für den Fall $n < 9$ muss die Gleichung (4.7) statt Gleichung (4.8) in c) angewandt werden.

Zur Berechnung des Schrittes a) ist es nötig, dass zuvor der Weg über drei Ecken berechnet wurde. Dies geschieht im ersten Schritt, falls $\frac{(1-\alpha)n}{4} \geq 3$ gilt. Diese Bedingung wird für $n \geq 13$ erfüllt.

Für $n < 13$ muss in a) Gleichung (4.7) statt Gleichung (4.8) angewandt werden.

3. Im letzten Änderungsvorschlag wird ähnlich wie im zweiten Änderungsvorschlag vorgegangen. Allerdings werden im ersten Teil beide Wege über $\frac{n}{2}$ Ecken durch Gleichung (4.7) verlängert, sodass man zwei Wege über $\frac{n}{2} + 1$ Ecken zu einem Weg über n Ecken zusammensetzen kann.

   a) $\min\limits_{\substack{S \subset V \\ |S| = n/2+1}} \min\limits_{\substack{u,v \in S \\ u \neq v}} \{f(S,u,v) + f((V \setminus S) \cup \{u,v\}, v, u)\}$
   Berechne dafür $f(S,u,v)$ mit $|S| = \frac{n}{2} + 1$ und $f((V \setminus S) \cup \{u,v\}, v, u)$ mit $|(V \setminus S) \cup \{u,v\}| = \frac{n}{2} + 1$ durch Anwenden von Gleichung (4.7).

   b) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{2}$ mit Gleichung (4.8), wobei $k = \frac{n}{4} + 1$ genutzt wird.
   Berechne dafür $f(S,u,v)$ mit $|S| = \frac{n}{4} + 1$ durch Anwenden von Gleichung (4.7).

   c) Berechne $f(S,u,v)$ mit $|S| = \frac{n}{4}$ mit Gleichung (4.8), wobei $k = \frac{\alpha n}{4} + 1$ genutzt wird.
   Analog zu 1.c) und 2.c) wird hier k aufgerundet.

   Wie beim zweiten Änderungsvorschlag beschrieben, ist die Rechnung in c) für $n \geq 9$ möglich. Falls $n < 9$ gilt, muss Gleichung (4.7) statt Gleichung (4.8) in 3.c) angewandt werden.

Um aufzuzeigen, wie die drei Änderungsvorschläge funktionieren, wird im Folgenden ein Beispiel mit $|V| = n = 64$ Ecken betrachtet. Dadurch soll verdeutlicht werden, wie die Aufteilung in die Teilmengen abläuft.

1. Erste Anpassung des Algorithmus von Ambainis

   a) Es werden Wege über $|S| = \frac{n}{2} + 1 = 33$ und $|(V \setminus S) \cup \{u,v\}| = \frac{n}{2} + 1 = 33$ Ecken zu einem Weg über $|V| = 64$ Ecken zusammengefügt.

   b) Nun wird ein Weg über $|S| = \frac{n}{2} + 1 = 33$ Ecken aus Teilwegen, die über jeweils $|X| = \frac{n}{4} + 1 = 17$ und $|(S \setminus X) \cup \{t\}| = \frac{n}{4} + 1 = 17$ Ecken verlaufen, zusammengesetzt.

c) Um die Wege über 17 Ecken zu berechnen, werden im letzten Schritt Wege über $|X| = \lceil \frac{\alpha n}{4} + 2 \rceil = 3$ und $|(S \setminus X) \cup \{t\}| = \lfloor \frac{(1-\alpha)n}{4} \rfloor = 15$ Ecken zu einem Weg, der über $|S| = \frac{n}{4} + 1 = 17$ Ecken geht, zusammengefügt.

2. Zweite Anpassung des Algorithmus von Ambainis

a) Es werden Wege über $|S| = \frac{n}{2} + 2 = 34$ und $|(V \setminus S) \cup \{u, v\}| = \frac{n}{2} = 32$ Ecken zu einem Gesamtweg, der über $|V| = 64$ Ecken verläuft, zusammengefügt.
Der Weg über $|S| = \frac{n}{2} + 2 = 34$ Ecken wird durch Anwendung von Gleichung (4.8) mit $|(S \setminus X) \cup \{t\}| = 3$ auf Wege mit $|X| = \frac{n}{2} = 32$ Ecken bestimmt.

b) Nun werden Wege über $|X| = \frac{n}{4} + 1 = 17$ und $|(S \setminus X) \cup \{t\}| = \frac{n}{4} = 16$ Ecken zu einem Weg über $|S| = \frac{n}{2} = 32$ Ecken zusammengefügt.
Der Weg über $|X| = \frac{n}{4} + 1 = 17$ Ecken wird durch Anwendung von Gleichung (4.7) auf Wege mit $\frac{n}{4} = 16$ Ecken bestimmt.

c) Im letzten Teil wird der Weg über $|S| = \frac{n}{4} = 16$ Ecken durch das Zusammenfügen von Wegen über $|X| = \lceil \frac{\alpha n}{4} + 1 \rceil = 2$ und $|(S \setminus X) \cup \{t\}| = \lfloor \frac{(1-\alpha)n}{4} \rfloor = 15$ Ecken berechnet.

3. Dritte Anpassung des Algorithmus von Ambainis

a) Aus Wegen, die über $|S| = \frac{n}{2} + 1 = 33$ und $|(V \setminus S) \cup \{u, v\}| = \frac{n}{2} + 1 = 33$ Ecken verlaufen, wird ein Weg über $|V| = 64$ Ecken berechnet.
Die Wege über $|S| = \frac{n}{2} + 1 = 33$ und $|(V \setminus S) \cup \{u, v\}| = \frac{n}{2} + 1 = 33$ Ecken werden dabei durch Anwendung von Gleichung (4.7) auf Wege über $\frac{n}{2} = 32$ Ecken bestimmt.

b) Im zweiten Teil werden Wege über $|X| = \frac{n}{4} + 1 = 17$ und $|(S \setminus X) \cup \{t\}| = \frac{n}{4} = 16$ Ecken zu einem Weg, der über $|S| = \frac{n}{2} = 32$ Ecken verläuft, zusammengefügt.
Dabei wird durch Anwendung von Gleichung (4.7) auf Wege über $|S| = \frac{n}{4} = 16$ Ecken der Weg über $|X| = \frac{n}{4} + 1 = 17$ Ecken bestimmt.

c) Nun kann der Weg über $|S| = \frac{n}{4} = 16$ Ecken durch das Zusammenfügen von Wegen über $|X| = \lceil \frac{\alpha n}{4} + 1 \rceil = 2$ und $|(S \setminus X) \cup \{t\}| = \lfloor \frac{(1-\alpha)n}{4} \rfloor = 15$ Elemente bestimmt werden.

4.3.2. Laufzeitbestimmung der drei Quantenalgorithmen

Die drei Änderungsmöglichkeiten für den Quantenalgorithmus bieten verschiedene Lösungswege für das TSP an. Dabei stellt sich die Frage, welcher dieser Algorithmen die beste Laufzeit hat, um der Suche nach der Lösung für das TSP zu optimieren. Im Folgenden wird die Laufzeit der drei Algorithmen bestimmt. Dafür wird berechnet, wie viele Wege $f(S, u, v)$ betrachtet und wie viele Additionen zur Berechnung dieser benötigt werden. Die Anzahl an Additionen gibt an, über wie viele Elemente der Liste die Minimumssuche berechnet wird. Aus der Anzahl und Länge der Listen setzt sich die Laufzeit zusammen.

Um die Komplexitäten der Algorithmen zu senken, wird der Startpunkt des Weges festgelegt. Würde man dies nicht tun, so würden Wege doppelt berechnet werden. Bei einer Rechnung über vier Ecken $V = \{a, b, c, d\}$ wären unter anderem die folgenden Wege über die vier Ecken bei der Berechnung möglich:

$$a \to b \to c \to d \to a$$
$$b \to c \to d \to a \to b$$
$$c \to d \to a \to b \to c$$
$$d \to a \to b \to c \to d$$

Hierbei sind alle vier Wege identisch, da die Handelsreise einer Permutation entspricht und diese zyklisch ist. Somit kann man beispielsweise $a \in V$ als Startpunkt für die Rechnung festlegen. Bei der Berechnung der Komplexitäten mit dieser Bedingung erhält man folgende Lösungen:

1. Für den ersten Algorithmus kann man die folgenden Komplexitäten festhalten:
   a) $\binom{n-1}{\frac{n}{2}}$ gibt an, wie viele Möglichkeiten es gibt, S mit $|S| = \frac{n}{2} + 1$ zu bilden. Dabei ist der Anfangsort u festgelegt und wird nicht berücksichtigt. Außerdem gibt es für die Wahl der Ecke v, welche die Verbindung zwischen den beiden Mengen S und $(V \setminus S) \cup \{u, v\}$ bildet, $\frac{n}{2}$ Möglichkeiten. Somit werden in diesem Rechenschritt $\binom{n-1}{\frac{n}{2}} \cdot \frac{n}{2}$ verschiedene Additionen von Wegen berücksichtigt.

   b) Für einen Weg $f(S, u, v)$ mit $|S| = \frac{n}{2} + 1$ gibt es $\binom{\frac{n}{2}-1}{\frac{n}{4}}$ Möglichkeiten, eine Menge X mit $|X| = \frac{n}{4} + 1$ Ecken aus S auszuwählen. Dabei sind die Ecken u und v vorgegeben. In dieser Rechnung wird ein Übergangsort t benötigt, an dem zwei Wege über jeweils $|X| = \frac{n}{4} + 1$ Ecken verbunden werden. Für die Wahl von t gibt es $\frac{n}{4}$ Möglichkeiten. Somit werden $\binom{\frac{n}{2}-1}{\frac{n}{4}} \cdot \frac{n}{4}$ verschiedene Rechnungen durchgeführt.

   c) Für den Weg $f(S, u, v)$ mit $|S| = \frac{n}{4} + 1$ gibt es $\binom{\frac{n}{4}-1}{\lceil \frac{\alpha n}{4}+1 \rceil}$ Möglichkeiten, eine Menge X mit $|X| = \lceil \frac{\alpha n}{4} + 2 \rceil$ Elementen aus S auszuwählen. Auch in diesem Schritt sind die Ecken u und v festgelegt, weswegen der Binomialkoeffizient nicht $\binom{\frac{n}{4}+1}{\lceil \frac{\alpha n}{4}+2 \rceil}$ ist. Für die Verbindung zweier Wege über jeweils $|X| = \lceil \frac{\alpha n}{4} + 2 \rceil$ und $|(S \setminus X) \cup \{t\}| = \lfloor \frac{(1-\alpha)n}{4} \rfloor$ Ecken wird ein Übergangsort t benötigt. Dabei gibt es $\lceil \frac{\alpha n}{4} + 1 \rceil$ Möglichkeiten, diesen zu wählen. Es werden also insgesamt $\binom{\frac{n}{4}-1}{\lceil \frac{\alpha n}{4}+1 \rceil} \cdot (\lceil \frac{\alpha n}{4} + 1 \rceil)$ verschiedene Rechnungen durchgeführt.

   Insgesamt ergibt sich, dass

$$\binom{n-1}{\frac{n}{2}} \cdot \frac{n}{2} \cdot 2 \cdot \binom{\frac{n}{2}-1}{\frac{n}{4}} \cdot \frac{n}{4} \cdot 2 \cdot \binom{\frac{n}{4}-1}{\frac{\alpha n}{4}+1} \cdot \left(\frac{\alpha n}{4} + 1\right)$$
$$= \frac{4 \cdot (n-1)!}{(\frac{n}{2}-1)! \cdot (\frac{n}{4}-1)! \cdot \frac{\alpha n}{4}! \cdot \left(\frac{(1-\alpha)n}{4} - 2\right)!}$$

Rechenschritte durchgeführt werden. Die Schritte b) und c) werden dabei doppelt ausgeführt, da diese für jeweils zwei Mengen berechnet werden. Beispielsweise wird der Schritt b) für die Wege $f(S,u,v)$ und $f((V \setminus S) \cup \{u,v\},v,u)$ aus Schritt a) angewandt. Da zur Minimumssuche der Quantenalgorithmus aus Abschnitt 3.5.1 genutzt wird, ist die Laufzeit durch

$$\mathcal{O}\left(\sqrt{\frac{4\cdot(n-1)!}{(\frac{n}{2}-1)!\cdot(\frac{n}{4}-1)!\cdot\frac{\alpha n}{4}!\cdot\left(\frac{(1-\alpha)n}{4}-2\right)!}}\right)$$

gegeben.

2. Für den zweiten Algorithmus kann man die Laufzeit folgendermaßen berechnen:
   a) $\binom{n-1}{\frac{n}{2}+1}$ gibt an, wie viele Möglichkeiten es gibt, S mit $|S| = \frac{n}{2}+2$ zu bilden. Der Anfangsort u ist hierbei festgelegt und wird nicht berücksichtigt. Weiterhin gibt es für die Wahl des Verbindungsortes t, der die Wege über S und $(V \setminus S) \cup \{u,v\}$ verbindet, $\frac{n}{2}+1$ Möglichkeiten. Somit werden in diesem Rechenschritt $\binom{n-1}{\frac{n}{2}+1}\cdot(\frac{n}{2}+1)$ verschiedene Additionen von Wegen berücksichtigt. Außerdem werden in diesem Schritt mit Gleichung (4.8) zwei weitere Ecken in Form des Weges $f((S \setminus X) \cup \{t\},t,v)$ hinzugefügt. Der Endort ist durch v gegeben. Für die Wahl der zweiten hinzugefügten Ecke und dem Verbindungsort gibt es $\binom{\frac{n}{2}}{2} = \frac{1}{2}\cdot(\frac{n}{2}-1)\cdot\frac{n}{2}$ Möglichkeiten, diese aus $|S|$ zu wählen. Dabei existieren zwei Möglichkeiten für die Wahl des Verbindungsortes t.
   Es werden insgesamt $\binom{n-1}{\frac{n}{2}+1}\cdot(\frac{n}{2}+1)\cdot\frac{n}{2}\cdot(\frac{n}{2}-1)$ Rechenschritte berücksichtigt.
   b) Betrachtet man nun die Berechnung des Weges $f(S,u,v)$ mit $|S| = \frac{n}{2}$, so gibt es $\binom{\frac{n}{2}-2}{\frac{n}{4}}$ Möglichkeiten, eine Menge X mit $|X| = \frac{n}{4}+1$ Elementen aus S auszuwählen. Dabei sind die Ecken u und v durch $f(S,u,v)$ festgelegt. Für die Ecke t, an der zwei Wege über $|X| = \frac{n}{4}+1$ und $|(S \setminus X) \cup \{t\}| = \frac{n}{4}$ Ecken verbunden werden, gibt es $\frac{n}{4}$ Wahlmöglichkeiten. Es werden also $\binom{\frac{n}{2}-2}{\frac{n}{4}}\cdot\frac{n}{4}$ verschiedene Rechnungen durchgeführt.
   Da in diesem Schritt die Ecke u zu einer Menge mit $\frac{n}{4}$ Ecken durch Anwendung der Gleichung (4.7) hinzugefügt wird, müssen die Wahlmöglichkeiten für die Verbindungsecke bei der Rechnung ebenfalls mitberücksichtigt werden. Dabei gibt es $\frac{n}{4}-1$ Möglichkeiten, diese Ecke aus $X \setminus \{u,t\}$ auszuwählen.
   Insgesamt werden in diesem Schritt $\binom{\frac{n}{2}-2}{\frac{n}{4}}\cdot\frac{n}{4}\cdot(\frac{n}{4}-1)$ Rechenschritte benötigt.
   c) Für den Weg $f(S,u,v)$, wobei $|S| = \frac{n}{4}$ gilt, gibt es hingegen $\binom{\frac{n}{4}-2}{\lceil\frac{\alpha n}{4}\rceil}$ Möglichkeiten, eine Menge X mit $|X| = \lceil\frac{\alpha n}{4}+1\rceil$ aus S zu wählen. Wie auch in den Schritten zuvor wurde hierbei berücksichtigt, dass die Orte u und v festgelegt sind. Um nun zwei Wege über jeweils $|X| = \lceil\frac{\alpha n}{4}+1\rceil$ und $|(S \setminus X) \cup \{t\}| = \lfloor\frac{(1-\alpha)n}{4}\rfloor$ Ecken zu verbinden, wird ein Übergangsort t benötigt. Für t gibt es $\lceil\frac{\alpha n}{4}\rceil$ Wahlmöglichkeiten. Es werden also insgesamt $\binom{\frac{n}{4}-2}{\lceil\frac{\alpha n}{4}\rceil}\cdot\lceil\frac{\alpha n}{4}\rceil$ verschiedene Rechnungen durchgeführt.

Es folgt, dass

$$\binom{n-1}{\frac{n}{2}+1} \cdot \left(\frac{n}{2}+1\right) \cdot \frac{n}{2} \cdot \left(\frac{n}{2}-1\right) \cdot 2 \cdot \binom{\frac{n}{2}-2}{\frac{n}{4}} \cdot \frac{n}{4} \cdot \left(\frac{n}{4}-1\right) \cdot 2 \cdot \binom{\frac{n}{4}-2}{\frac{\alpha n}{4}} \cdot \frac{\alpha n}{4}$$
$$= \frac{4 \cdot (n-1)!}{(\frac{n}{2}-2)! \cdot (\frac{n}{4}-2)! \cdot (\frac{\alpha n}{4}-1)! \cdot \left(\frac{(1-\alpha)n}{4}-2\right)!}$$

Rechenschritte durchgeführt werden. Aus dem gleichen Grund wie im ersten Änderungsvorschlag werden die Laufzeiten von b) und c) mit 2 multipliziert. Da die Quantenminimumsuche zum Auffinden des Minimums benötigt wird, ergibt sich eine Laufzeit von

$$\mathcal{O}\left(\sqrt{\frac{4 \cdot (n-1)!}{(\frac{n}{2}-2)! \cdot (\frac{n}{4}-2)! \cdot (\frac{\alpha n}{4}-1)! \cdot \left(\frac{(1-\alpha)n}{4}-2\right)!}}\right).$$

3. Für den dritten Änderungsvorschlag des Algorithmus kann man die Laufzeit analog zum zweiten Änderungsvorschlag berechnen. Der einzige Unterschied ergibt sich durch die Berechnung des Schrittes a). Es folgt:

   a) Durch $\binom{n-1}{\frac{n}{2}}$ wird angegeben, wie viele Möglichkeiten es gibt, S mit $|S| = \frac{n}{2}+1$ zu bilden. Dabei ist der Anfangsort u festgelegt und wird für die Wahl von S nicht berücksichtigt. Außerdem werden die beiden Wege über die Mengen S und $(V \setminus S) \cup \{u, v\}$ durch die Ecke v verbunden. v kann für eine festgelegte Menge S insgesamt $\frac{n}{2}$ verschiedene Ecken beschreiben. Somit werden in diesem Rechenschritt $\binom{n-1}{\frac{n}{2}} \cdot \frac{n}{2}$ verschiedene Wege berücksichtigt.
   Außerdem wird in diesem Schritt mit Gleichung (4.7) eine weitere Ecke zu den Wegen $f(S, u, v)$ und $f((V \setminus S) \cup \{u, v\}, v, u)$ hinzugefügt. Es gibt hierbei $\frac{n}{2}-1$ Möglichkeiten, diese Ecke aus $S \setminus \{u, v\}$ zu wählen.
   Es werden in diesem Schritt $\binom{n-1}{\frac{n}{2}} \cdot \frac{n}{2} \cdot (\frac{n}{2}-1)$ Rechenschritte durchgeführt.

   b) Analog zu 2.b) werden in diesem Schritt des Algorithmus $\binom{\frac{n}{2}-2}{\frac{n}{4}} \cdot \frac{n}{4} \cdot (\frac{n}{4}-1)$ Rechenschritte benötigt.

   c) Auch im letzten Schritt kann die Anzahl an Rechenschritten aus 2.c) übernommen werden. Diese beträgt $\binom{\frac{n}{4}-2}{\lceil \frac{\alpha n}{4} \rceil} \cdot \lceil \frac{\alpha n}{4} \rceil$.

   Durch die Multiplikation aller drei Anzahlen an Rechenschritten folgt, dass

$$\binom{n-1}{\frac{n}{2}} \cdot \frac{n}{2} \cdot \left(\frac{n}{2}-1\right) \cdot 2 \cdot \binom{\frac{n}{2}-2}{\frac{n}{4}} \cdot \frac{n}{4} \cdot \left(\frac{n}{4}-1\right) \cdot 2 \cdot \binom{\frac{n}{4}-2}{\frac{\alpha n}{4}} \cdot \frac{\alpha n}{4}$$
$$= \frac{4 \cdot (n-1)!}{(\frac{n}{2}-1)! \cdot (\frac{n}{4}-2)! \cdot (\frac{\alpha n}{4}-1)! \cdot \left(\frac{(1-\alpha)n}{4}-2\right)!}$$

Rechenschritte durchgeführt werden. Wie auch im ersten und zweiten Algorithmus werden die Schritte b) und c) zweifach ausgeführt. Durch die Quantenminimumssuche ergibt sich für den zweiten Schritt des dritten Änderungsvorschlages eine Laufzeit von

$$\mathcal{O}\left(\sqrt{\frac{4\cdot(n-1)!}{(\frac{n}{2}-1)!\cdot(\frac{n}{4}-2)!\cdot(\frac{\alpha n}{4}-1)!\cdot\left(\frac{(1-\alpha)n}{4}-2\right)!}}\right).$$

Durch einen Abgleich der Laufzeiten lässt sich beobachten, dass der erste Algorithmus die schnellste Anpassung darstellt. Bei dem Änderungsvorschlag mit der zweitbesten Laufzeit handelt es sich um den dritten Algorithmus. Die schlechteste Laufzeit hat der zweite Änderungsvorschlag. Es gilt nämlich:

$$\begin{aligned}
&\mathcal{O}\left(\sqrt{\frac{4\cdot(n-1)!}{(\frac{n}{2}-1)!\cdot(\frac{n}{4}-1)!\cdot\frac{\alpha n}{4}!\cdot\left(\frac{(1-\alpha)n}{4}-2\right)!}}\right) && \text{Algorithmus 1}\\
&<\mathcal{O}\left(\sqrt{\frac{4\cdot(n-1)!}{(\frac{n}{2}-1)!\cdot(\frac{n}{4}-2)!\cdot(\frac{\alpha n}{4}-1)!\cdot\left(\frac{(1-\alpha)n}{4}-2\right)!}}\right) && \text{Algorithmus 3}\\
&<\mathcal{O}\left(\sqrt{\frac{4\cdot(n-1)!}{(\frac{n}{2}-2)!\cdot(\frac{n}{4}-2)!\cdot(\frac{\alpha n}{4}-1)!\cdot\left(\frac{(1-\alpha)n}{4}-2\right)!}}\right) && \text{Algorithmus 2}
\end{aligned}$$

Im Folgenden wird nur der erste Änderungsvorschlag betrachtet. Um mit der Laufzeit des angepassten Quantenalgorithmus aus Abschnitt 4.3 besser arbeiten zu können, kann man in der Laufzeit die polynomiellen Komplexitäten vernachlässigen. Dafür kann die Anzahl an Rechenschritten des angepassten Algorithmus wie folgt umgestellt werden:

$$\begin{aligned}
&\binom{n-1}{\frac{n}{2}}\cdot\frac{n}{2}\cdot 2\cdot\binom{\frac{n}{2}-1}{\frac{n}{4}}\cdot\frac{n}{4}\cdot 2\cdot\binom{\frac{n}{4}-1}{\frac{\alpha n}{4}+1}\cdot\left(\frac{\alpha n}{4}+1\right)\\
=&\binom{n}{\frac{n}{2}}\cdot\frac{1}{2}\cdot\frac{n}{2}\cdot 2\cdot\binom{\frac{n}{2}}{\frac{n}{4}}\cdot\frac{1}{2}\cdot\frac{n}{4}\cdot 2\cdot\binom{\frac{n}{4}}{\frac{\alpha n}{4}}\cdot\frac{(1-\alpha)n-4}{n}\cdot\frac{(1-\alpha)n}{\alpha n+4}\cdot\left(\frac{\alpha n}{4}+1\right)\\
=&\binom{n}{\frac{n}{2}}\cdot\frac{n}{2}\cdot\binom{\frac{n}{2}}{\frac{n}{4}}\cdot\frac{n}{4}\cdot\binom{\frac{n}{4}}{\frac{\alpha n}{4}}\cdot\left(\frac{(1-\alpha)n}{4}-1\right)\cdot(1-\alpha)
\end{aligned}$$

Nun wird die Quantenminimumssuche durchgeführt. Durch die Nutzung von $\mathcal{O}^*$ fallen alle polynomiellen Terme weg und die Laufzeit lässt sich umschreiben zu

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\frac{n}{2}}\binom{\frac{n}{2}}{\frac{n}{4}}\binom{\frac{n}{4}}{\frac{\alpha n}{4}}}\right). \tag{4.9}$$

Diese Laufzeit geben auch Ambainis et al. in ihrem Quantenalgorithmus für den zweiten Schritt des Algorithmus an [2]. Somit sind diese identisch, egal ob man den angepassten oder den originalen Quantenalgorithmus von Ambainis et al. betrachtet. Daraus lässt sich vermuten, dass Ambainis et al. die Anpassung des Algorithmus nicht betrachtet haben, da diese für die Laufzeit irrelevant ist.

Für den ersten Algorithmus wird im Anhang A.2 ein Beispiel berechnet.

Bei den Algorithmen gilt es zu beachten, dass mit $\frac{n}{2}$ und $\frac{n}{4}$ gerechnet wird. Es wurde somit davon ausgegangen, dass n ein Vielfaches von 4 ist. Falls dies nicht der Fall ist, muss der Algorithmus ebenfalls angepasst werden.

4.4. Vergleich mit weiteren Veränderungen des Algorithmus

Ambainis et al. schreiben in ihrem Text [2], dass die Unterteilung der Wege in mehr oder weniger als zwei Teile sowie mehr oder weniger als drei Rekursionsschritte eine schlechtere Laufzeit hervorbringt. Dies wird im Folgenden bewiesen.

Ambainis et al. geben eine Laufzeit von

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{4}}\right) = \sum_{i=1}^{\frac{(1-\alpha)n}{4}} \mathcal{O}^*\left(\binom{n}{i}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{4}\right)n}\right) \tag{4.10}$$

für den ersten Schritt ihres Algorithmus an [2]. Diese ergibt sich, da in jedem Schritt $\binom{n}{|S|}$ Möglichkeiten zur Wahl von S bestehen. Dabei gilt $|S| \leq \frac{(1-\alpha)n}{4}$. Für den zweiten Schritt des Algorithmus ergibt sich die Laufzeit aus Gleichung (4.9). Diese schreiben Ambainis et al. zu

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\frac{n}{2}}\binom{\frac{n}{2}}{\frac{n}{4}}\binom{\frac{n}{4}}{\frac{\alpha n}{4}}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{1}{2}+\frac{H(\alpha)}{4}\right)n}\right) \tag{4.11}$$

um [2]. Dabei gilt $H(\epsilon) = -(\epsilon \log_2(\epsilon) + (1-\epsilon)\log_2(1-\epsilon))$ mit $\epsilon \in [0,1]$ [2].

Die beiden Laufzeiten aus Gleichung (4.10) und (4.11) lassen sich daraus herleiten, dass für alle $1 \leq k \leq \frac{n}{2}$

$$\binom{n}{\leq k} = \sum_{i=0}^{k} \binom{n}{i} \leq 2^{H\left(\frac{k}{n}\right)\cdot n} \tag{4.12}$$

gilt [2]. Falls hingegen $k > \frac{n}{2}$ gilt, kann man die Näherung

$$\binom{n}{\leq k} \leq 2^n \tag{4.13}$$

nutzen [2]. Die Näherung aus Gleichung (4.12) wurde dabei für Gleichung (4.10) angewandt. Zur Herleitung von Gleichung (4.11) wurde hingegen Gleichung (4.14) genutzt [2]. Diese Formel lässt sich ebenfalls aus Gleichung (4.12) herleiten, indem die Summation über mehrere Binomialkoeffizienten vernachlässigt wird und stattdessen nur ein einzelner

Binomialkoeffizient betrachtet wird.

$$\binom{n}{k} \leq 2^{H\left(\frac{k}{n}\right)\cdot n} \tag{4.14}$$

Die Laufzeit wird minimiert, falls beide Laufzeiten aus Gleichung (4.10) und (4.11) gleich groß sind [2]. Dabei muss wie in Abschnitt 4.2 beschrieben $\alpha \in \left(0, \frac{1}{2}\right]$ gelten. Es ergibt sich, dass beide Laufzeiten für $\alpha \approx 0.055362$ gleich groß sind [2]. Somit folgt eine Gesamtlaufzeit von ungefähr $\mathcal{O}^*\left(1.727391^n\right)$ [2].

4.4.1. Unterschiedliche Anzahl an Rekursionsschritten

Für unterschiedliche Anzahlen an Rekursionsschritten im zweiten Teil des Algorithmus kann man sowohl mehr als drei als auch weniger als drei Rekursionsschritte betrachten. Dabei stellt sich die Frage, wieso überhaupt eine Verknüpfung des ersten Schrittes mit dem zweiten Schritt geschehen muss. Wieso weder eine reine Betrachtung des ersten Schrittes, eine Betrachtung von einem, zwei oder mehr als drei Rekursionsschritten, worunter auch eine reine Betrachtung des zweiten Schrittes zur Lösung fällt, noch eine Variation in den Unterteilungen der Menge pro Rekursionsschritt sinnvoll ist, wird in diesem und folgenden Abschnitten geklärt.

Keine Rekursionsschritte

Betrachtet man eine reine Berechnung des Algorithmus anhand von Gleichung (4.7), so gibt es für jeden Schritt mit $|S| = k$ insgesamt $\binom{n}{k}$ Möglichkeiten, die Menge S zu wählen. Die Wahlmöglichkeiten für u und v zur Berechnung von $f(S, u, v)$ werden ignoriert, da diese einen polynomiellen Beitrag zur Laufzeit beisteuern. Darüber hinaus führt die Wahl der Verbindungsecke t, über welche die Gleichung (4.7) das Minimum sucht, ebenfalls zu einem polynomiellen Beitrag in der Laufzeit. Im Folgenden wird mit $\mathcal{O}^*$ gerechnet, weswegen die polynomiellen Beiträge irrelevant sind. Im Anschluss an die Berechnung aller Wege über die Mengen S mit $|S| = k$ wird eine weitere Ecke hinzugefügt. Es finden somit Rechnungen über alle möglichen Mengen S mit $1 \leq k \leq n$ statt. Würde man den Algorithmus von Ambainis et al. aus Abschnitt 4.2 ohne Durchführung der Rekursionsschritte im zweiten Schritt durchführen, so müsste man mit $\sum_{k=1}^{n} \binom{n}{k} = 2^n - 1$ Möglichkeiten zur Wahl von S rechnen, um den kürzesten Weg zu finden. Es ergibt sich also eine Laufzeit von $\mathcal{O}^*(2^n)$, was keinen Speedup gegenüber dem Algorithmus von Ambainis et al. bringt.

Ein Rekursionsschritt

In den Rechnungen für einen oder mehrere Rekursionsschritte wird die Berechnung der Laufzeit analog zu der durch Ambainis et al. beschriebenen Rechnung in [2] durchgeführt.

Für die Durchführung von nur einem Rekursionsschritt folgt für den ersten Schritt des Algorithmus die folgende Laufzeit:

$$\mathcal{O}^*\left(\binom{n}{\leq (1-\alpha)n}\right)$$

Dabei gilt es zu beachten, dass hierbei Gleichung (4.13) angewandt werden muss, da $(1-\alpha) \geq \frac{1}{2}$ gilt. Es folgt:

$$\mathcal{O}^*\left(\binom{n}{\leq (1-\alpha)n}\right) = \mathcal{O}^*\left(2^n\right)$$

Für den zweiten Schritt gilt:

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\alpha n}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}H(\alpha)n}\right)$$

Die Laufzeit wird in diesem Fall durch den ersten Rekursionsschritt bestimmt und ergibt für alle α einen Wert von $\mathcal{O}^*\left(2^n\right)$. Diese Laufzeit ist schlechter als $\mathcal{O}^*\left(1.727391^n\right)$. Somit ist die Nutzung von einem Rekursionsschritt nicht zielführend.

Zwei Rekursionsschritte

Für zwei Rekursionsschritte gilt die folgende Laufzeit für den ersten Teil des Algorithmus:

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{2}}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{2}\right)n}\right)$$

Für den zweiten Teil ist die Laufzeit wie folgt gegeben:

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\frac{n}{2}}\binom{\frac{n}{2}}{\frac{\alpha n}{2}}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{H(\alpha)}{2}\right)n}\right)$$

Beide Laufzeiten sind identisch für $\alpha \approx 0.575368$. Hierbei ergibt sich eine Gesamtlaufzeit von $\mathcal{O}^*\left(1.677005^n\right)$. Diese Laufzeit ist besser als die des originalen Algorithmus mit $\mathcal{O}^*\left(1.727391^n\right)$. Allerdings fällt auf, dass in diesem Fall $\alpha > \frac{1}{2}$ gilt. Dies lässt sich mit dem Algorithmus nicht berechnen, da $\alpha \in \left(0, \frac{1}{2}\right]$ angenommen wurde. Aufgrund dessen muss die minimale Laufzeit auf einem anderen Weg gefunden werden. Dabei wird das Minimum der beiden Laufzeiten für $0 < \alpha \leq \frac{1}{2}$ gesucht. Die Gesamtlaufzeit entspricht der größeren minimalen Laufzeit beider Algorithmen, da die größere Laufzeit einen größeren

Einfluss auf die Gesamtlaufzeit hat. Es ergeben sich die folgenden beiden Minima:

$$\min_{\alpha\in\left(0,\frac{1}{2}\right]}\{\mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{2}\right)n}\right)\} = \mathcal{O}^*\left(1.754765^n\right) \qquad \text{für } \alpha = \frac{1}{2}$$

$$\min_{\alpha\in\left(0,\frac{1}{2}\right]}\{\mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{H(\alpha)}{2}\right)n}\right)\} = \mathcal{O}^*\left(1.414214^n\right) \qquad \text{für } \alpha \to 0$$

Für den ersten Teil des Algorithmus folgt eine Laufzeit von bestenfalls $\mathcal{O}^*\left(1.754765^n\right)$ für $\alpha = \frac{1}{2}$. Um eine endgültige Laufzeit für den ganzen Algorithmus zu bestimmen, muss noch der zweite Schritt des Algorithmus für $\alpha = \frac{1}{2}$ betrachtet werden. Dabei muss geprüft werden, ob die Laufzeit des zweiten Schrittes die vom ersten Schritt übersteigt oder nicht. Es folgt:

$$\mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{H(\alpha)}{2}\right)n}\right) \overset{\alpha=0.5}{=} \mathcal{O}^*\left(1.681793^n\right) < \mathcal{O}^*\left(1.754765^n\right)$$

Somit zeigt sich, dass die Laufzeit $\mathcal{O}^*\left(1.754765^n\right)$ die beste Laufzeit ist, die mit zwei Rekursionsschritten erreicht werden kann. Da sie langsamer als $\mathcal{O}^*\left(1.727391^n\right)$ ist, sind zwei Rekursionsschritte für eine bessere Laufzeit nicht geeignet.

Vier Rekursionsschritte

Für die Laufzeit des ersten Schrittes des Algorithmus gilt bei vier Rekursionsschritten:

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{8}}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{8}\right)n}\right)$$

Für den zweiten Schritt gilt hingegen:

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\frac{n}{2}}\binom{\frac{n}{2}}{\frac{n}{4}}\binom{\frac{n}{4}}{\frac{n}{8}}\binom{\frac{n}{8}}{\frac{\alpha n}{8}}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{1}{2}+\frac{1}{4}+\frac{H(\alpha)}{8}\right)n}\right)$$

Hierbei reicht es alleine das Minimum für den zweiten Schritt zu betrachten. Das Minimum für $\alpha \in \left(0, \frac{1}{2}\right]$ liegt dabei bei

$$\min_{\alpha\in\left(0,\frac{1}{2}\right]} \mathcal{O}^*\left(2^{\frac{1}{2}\left(1+\frac{1}{2}+\frac{1}{4}+\frac{H(\alpha)}{8}\right)n}\right) = \mathcal{O}^*\left(1.834008^n\right).$$

Das Minimum für den zweiten Schritt des Quantenalgorithmus gibt zeitgleich ein Minimum für die Gesamtlaufzeit von diesem an. Dabei folgt für vier Rekursionsschritte eine schlechtere Laufzeit als für drei Rekursionsschritte mit $\mathcal{O}^*\left(1.727391^n\right)$. Dies lässt sich auch allgemein für alle Algorithmen mit mehr als drei Rekursionsschritten zeigen.

Mehr als drei Rekursionsschritte

Für mehr als drei Rekursionsschritte gilt die folgende Formel für den ersten Teil des Algorithmus:

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{2^{r-1}}}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{2^{r-1}}\right)n}\right)$$

Dabei beschreibt r die Anzahl an Rekursionsschritten.

Die Laufzeit des zweiten Teiles ergibt sich durch:

$$\mathcal{O}^*\left(\sqrt{\left[\prod_{k=0}^{r-2}\binom{\frac{n}{2^k}}{\frac{n}{2^{k+1}}}\right]\binom{\frac{n}{2^{r-1}}}{\frac{\alpha n}{2^{r-1}}}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(\left[\sum_{k=0}^{r-2}\frac{1}{2^k}\right]+\frac{H(\alpha)}{2^{r-1}}\right)n}\right)$$

Für $\alpha \in \left(0, \frac{1}{2}\right]$ nimmt $H(\alpha)$ Werte innerhalb von $(0, 1]$ an. Betrachtet man die Laufzeit des zweiten Schrittes, so folgt, dass diese minimal für $H(\alpha) \to 0$ ist. Somit folgt für den zweiten Schritt:

$$\mathcal{O}^*\left(2^{\frac{1}{2}\left(\left[\sum_{k=0}^{r-2}\frac{1}{2^k}\right]+\frac{H(\alpha)}{2^{r-1}}\right)n}\right) \overset{H(\alpha)\to 0}{=} \mathcal{O}^*\left(2^{\frac{1}{2}\left(\sum_{k=0}^{r-2}\frac{1}{2^k}\right)n}\right)$$

Für $r > 3$ gilt $\frac{1}{2}\left(\sum_{k=0}^{r-2}\frac{1}{2^k}\right) \geq \frac{7}{8}$. Dadurch folgt $\mathcal{O}^*\left(\geq 1.834008^n\right)$. Somit verbessern mehr als drei Rekursionsschritte die Laufzeit nicht. Insgesamt folgt, dass drei Rekursionsschritte zu der schnellsten Laufzeit mit $\mathcal{O}^*\left(1.727391^n\right)$ führen.

4.4.2. Unterteilung eines Weges in Teilwege über verschieden viele Ecken

Neben der Variation der Anzahl an Rekursionsschritten in dem Algorithmus von Ambainis et al. kann geprüft werden, ob eine Unterteilung des Weges im zweiten Schritt des Algorithmus in die beschriebenen Mengen sinnvoll ist. Ambainis et al. beschreiben eine Unterteilung der Wege in zwei kürzere Wege, die über die gleichen Anzahlen an Ecken verlaufen [2]. Im Folgenden soll gezeigt werden, ob eine Unterteilung in zwei Wege über die gleiche Anzahl an Ecken sinnvoll ist.

Betrachte eine Unterteilung des Weges über n Ecken in $x \cdot n$ und $(1-x) \cdot n$ Ecken pro Weg. Dass wie in Abschnitt 4.3 beschrieben einer der Wege über eine Ecke mehr verlaufen muss, wird vernachlässigt. Es gilt $x \in (0, 1)$. $x = 1$ oder $x = 0$ wird ausgeschlossen, da in diesem Fall keine Unterteilung vorgenommen wird. Dies würde nämlich der Nutzung von ausschließlich dem ersten Schritt aus Ambainis et al. Algorithmus entsprechen und die Laufzeit davon wurde in Abschnitt 4.4.1 bei der Nutzung von keinem Rekursionsschritt berechnet. Im Folgenden wird die Laufzeit zur Berechnung des längsten Teilweges betrachtet, da die Rechnung über mehr Ecken zu einer längeren Laufzeit führt. Das bedeutet, dass, wenn $x \geq 1 - x$ gilt, der Weg über $x \cdot n$ Ecken betrachtet werden soll,

da dieser eine längere Rechendauer benötigt als der Weg über $(1-x)\cdot n$ Ecken. Für $1-x \geq x$ kann man $x \to 1-x$ umschreiben und erhält ein analoges Ergebnis. Es wird somit $x \geq \frac{1}{2}$ angenommen, weswegen $x \in \left[\frac{1}{2}, 1\right)$ gelten soll.

Für den zweiten Schritt des Algorithmus folgt die Laufzeit

$$\mathcal{O}^*\left(\sqrt{\binom{n}{xn}\binom{xn}{x^2n}\binom{x^2n}{\alpha x^2 n}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(H(x)+H(x)\cdot x+H(\alpha)\cdot x^2\right)n}\right).$$

Für den ersten Schritt des Algorithmus folgt hingegen folgende Laufzeit:

$$\mathcal{O}^*\left(\binom{n}{\leq (1-\alpha)x^2 n}\right)$$

Hier muss eine Fallbetrachtung vorgenommen werden. Für $(1-\alpha)x^2 > \frac{1}{2}$ wird Gleichung (4.13) angewandt. Dabei folgt eine Laufzeit von

$$\mathcal{O}^*\left(\binom{n}{\leq (1-\alpha)x^2 n}\right) = \mathcal{O}^*(2^n),$$

die keinen Speedup gegenüber dem von Ambainis et al. präsentierten Algorithmus liefert [2]. Für $(1-\alpha)x^2 \leq \frac{1}{2}$ gilt hingegen:

$$\mathcal{O}^*\left(\binom{n}{\leq (1-\alpha)x^2 n}\right) = \mathcal{O}^*\left(2^{H((1-\alpha)\cdot x^2)n}\right)$$

Nun kann man zum Auffinden des Minimums beide Laufzeiten gleichsetzen. Daraus folgt:

$$\mathcal{O}^*\left(2^{\frac{1}{2}\left(H(x)+H(x)\cdot x+H(\alpha)\cdot x^2\right)n}\right) = \mathcal{O}^*\left(2^{H((1-\alpha)\cdot x^2)n}\right)$$

Das minimale Ergebnis unter den Bedingungen $\alpha \in \left(0, \frac{1}{2}\right]$, $x \in \left[\frac{1}{2}, 1\right)$ und $(1-\alpha)x^2 \leq \frac{1}{2}$ ergibt sich bei $x = \frac{1}{2}$ und $\alpha \approx 0.055362$. Dies kann auch grafisch in Abb. 4.1 vermutet werden.

Eine Aufteilung der Wege durch eine Halbierung der Anzahl an Ecken ergibt die beste Laufzeit. Diese wurde auch von Ambainis et al. genutzt. Somit folgt eine identische Laufzeit zu der, die von Ambainis et al. beschrieben wurde [2].

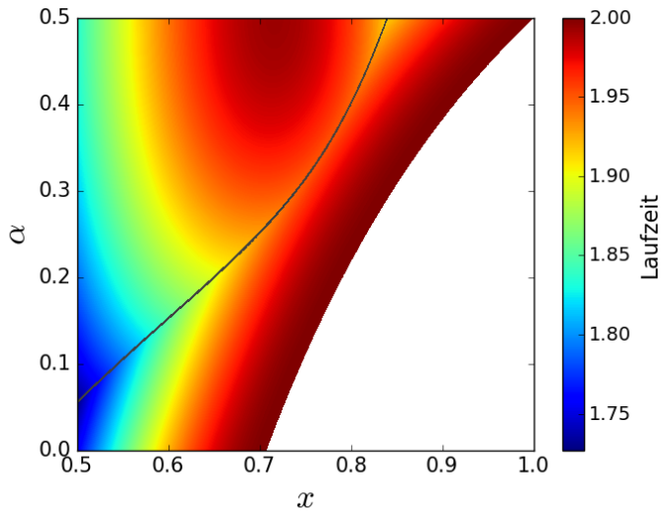


Abbildung 4.1.: In der Grafik sind die Laufzeiten für die jeweiligen Werte von x und α dargestellt. Der Ausschnitt unten rechts erfüllt die Bedingung $(1-\alpha)x^2 \leq \frac{1}{2}$ nicht und ist somit weiß dargestellt. Der diagonale Teil, der von unten links nach oben rechts verläuft, stellt die Laufzeit dar, die vom ersten Schritt des Algorithmus begrenzt wird. Der Abschnitt oben links bildet hingegen die vom zweiten Schritt des Algorithmus beschränkte Laufzeit ab. Die schwarze Linie zwischen den Abschnitten der beiden Algorithmen stellt die Stellen dar, an denen die Laufzeiten beider Schritte identisch sind. Das Minimum dieser entspricht dem Minimum der gesamten Laufzeit in dem dargestellten Bereich. Es liegt bei $x = \frac{1}{2}$ und $\alpha \approx 0.055362$.

4.4.3. Unterteilung eines Weges in mehr als zwei Teilwege

Nun soll die Aussage von Ambainis et al., dass sich eine Unterteilung des Weges in mehr als zwei Teilwege nicht lohnt [2], überprüft werden. Hierfür wird eine Unterteilung in Wege über die gleiche Anzahl an Ecken betrachtet, da sich durch das Beispiel mit einer Unterteilung in zwei Mengen aus Abschnitt 4.4.2 vermuten lässt, dass dies die Unterteilung mit der besten Laufzeit ist.

Im Folgenden wird sowohl die Unterteilung von V wie auch die von S aus Gleichung (4.8) in mehr als zwei Teilmengen betrachtet. Dabei soll k die Anzahl der Wege beschreiben, in die der Gesamtweg unterteilt wird. Im ersten Schritt wird eine Unterteilung in $k = 3$ Mengen betrachtet und im Anschluss eine allgemeine Unterteilung in mehr als zwei Mengen.

Unterteilung in drei Wege

Betrachte eine Unterteilung des Weges in drei Teilmengen. Dabei verändert sich die Laufzeit des ersten Schrittes vom Algorithmus. Für diesen folgt die folgende Laufzeit:

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{9}}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{9}\right)n}\right)$$

Auch die Laufzeit des zweiten Schrittes verändert sich wie folgt:

$$\mathcal{O}^*\left(\sqrt{\binom{n}{\frac{n}{3}}\binom{\frac{2n}{3}}{\frac{n}{3}}\binom{\frac{n}{3}}{\frac{n}{9}}\binom{\frac{2n}{9}}{\frac{n}{9}}\binom{\frac{n}{9}}{\frac{\alpha n}{9}}}\right) = \mathcal{O}^*\left(2^{\frac{1}{2}\left(H\left(\frac{1}{3}\right)+\frac{2}{3}+\frac{H\left(\frac{1}{3}\right)}{3}+\frac{2}{9}+\frac{H(\alpha)}{9}\right)n}\right)$$

Bei dieser Berechnung der Laufzeit des zweiten Schrittes des Algorithmus wird im ersten Teil, also bei der Unterteilung von V in drei Teile, erst eine Menge mit $\frac{n}{3}$ Elementen aus V gewählt. Daraus resultiert der Term $\binom{n}{\frac{n}{3}}$. Aus den restlichen $\frac{2n}{3}$ Elementen wird eine zweite Menge mit $\frac{n}{3}$ Elementen gewählt. Dabei existieren $\binom{\frac{2n}{3}}{\frac{n}{3}}$ Möglichkeiten, die zweite Menge zu wählen. Insgesamt folgen für den ersten Rekursionsschritt $\binom{n}{\frac{n}{3}}\binom{\frac{2n}{3}}{\frac{n}{3}}$ Kombinationsmöglichkeiten. Für den zweiten Schritt wird analog vorgegangen. Im letzten Schritt wird nur eine Unterteilung in zwei Mengen vorgenommen, da in diesem Schritt die fehlenden, im ersten Schritt nicht berechneten Ecken bis zu einem Weg über $\frac{n}{9}$ Ecken dazuaddiert werden.

Es reicht die Laufzeit für den zweiten Schritt des Algorithmus zu betrachten. Diese ist minimal für $\alpha \to 0$. Dabei ergibt sich eine Laufzeit von $\mathcal{O}^*\left(2.080084^n\right)$. Diese Laufzeit ist nicht besser als $\mathcal{O}^*(1.727391^n)$, weswegen eine Unterteilung eines langen Weges in zwei Wege sich mehr eignet als in drei Wege.

Unterteilung in mehr als zwei Wege

Betrachte nun allgemein eine Unterteilung der Wege in mehr als zwei Teilwege. Hierbei stellt r die Anzahl der Teilmengen dar, in die der Weg jeweils unterteilt wird. Für den ersten Schritt des Algorithmus gilt:

$$\mathcal{O}^*\left(\binom{n}{\leq \frac{(1-\alpha)n}{r^2}}\right) = \mathcal{O}^*\left(2^{H\left(\frac{1-\alpha}{r^2}\right)n}\right)$$

Für den zweiten Schritt folgt hingegen analog zu der Unterteilung in drei Teilmengen:

$$\begin{aligned}\mathcal{O}^*\left(\sqrt{\left[\prod_{k=2}^{r}\binom{\frac{kn}{r}}{\frac{n}{r}}\right]\left[\prod_{k=2}^{r}\binom{\frac{kn}{r^2}}{\frac{n}{r^2}}\right]\binom{\frac{n}{r^2}}{\frac{\alpha n}{r^2}}}\right) &= \mathcal{O}^*\left(2^{\frac{1}{2}\left(\sum_{k=2}^{r}H\left(\frac{1}{k}\right)\frac{k}{r}+\sum_{k=2}^{r}H\left(\frac{1}{k}\right)\frac{k}{r^2}+\frac{H(\alpha)}{r^2}\right)n}\right)\\ &= \mathcal{O}^*\left(2^{\frac{1}{2}\left(\sum_{k=2}^{r}\left[H\left(\frac{1}{k}\right)\frac{k}{r}+H\left(\frac{1}{k}\right)\frac{k}{r^2}\right]+\frac{H(\alpha)}{r^2}\right)n}\right)\end{aligned}$$

Auch in dieser Aufteilung reicht es die Laufzeit für den zweiten Schritt zu betrachten. Es muss beachtet werden, dass $\alpha \in \left(0, \frac{1}{2}\right]$ gilt, wodurch $H(\alpha) \in (0, 1]$ folgt. Die schnellste Laufzeit wird für $H(\alpha) \to 0$ erzielt. Es folgt:

$$\mathcal{O}^*\left(2^{\frac{1}{2}\left(\sum_{k=2}^{r}\left[H\left(\frac{1}{k}\right)\frac{k}{r}+H\left(\frac{1}{k}\right)\frac{k}{r^2}\right]+\frac{H(\alpha)}{r^2}\right)n}\right) \overset{H(\alpha)\to 0}{=} \mathcal{O}^*\left(2^{\frac{1}{2}\sum_{k=2}^{r}\left(H\left(\frac{1}{k}\right)\frac{k}{r}+H\left(\frac{1}{k}\right)\frac{k}{r^2}\right)n}\right)$$

Dabei gilt $\frac{1}{2}\sum_{k=2}^{r}\left(H\left(\frac{1}{k}\right)\frac{k}{r}+H\left(\frac{1}{k}\right)\frac{k}{r^2}\right)\geq 1.056642$ für $k\geq 3$. Somit ergibt sich eine Laufzeit von $\mathcal{O}^*\left(\geq 2.080084^n\right)$. Die Unterteilung in mehr als zwei Teilmengen lohnt sich nicht, da diese Laufzeit größer als die von Ambainis et al. berechnete Laufzeit ist.

Es folgt insgesamt, dass sich weder eine Unterteilung der Wege in mehr als zwei Teilmengen, eine Unterteilung in verschieden große Mengen noch eine Abweichung in der Anzahl an Rekursionsschritten lohnt, um die Laufzeit zu verbessern. Eine Einteilung in zwei Teilmengen und drei Rekursionsschritte bildet die schnellste Lösung, die auch von Ambainis et al. genutzt wurde. Hierbei ergibt sich $\alpha\approx 0.055362$ und $\mathcal{O}^*\left(1.727391^n\right)$ [2].

5 Kapitel 5. Quantenschaltkreise

Für das Implementieren des Quantenalgorithmus, der auf der dynamischen Programmierung basiert, sind mehrere Quantenschaltkreise nötig. In diesem Kapitel soll ein Überblick über drei verschiedene Schaltkreise gegeben werden, die auch für den Quantenalgorithmus benutzt werden müssen. Bei den Schaltkreisen handelt es sich um einen Addierer, einen Komparator und um einen Schaltkreis für den qRAM. Dabei wird der qRAM zum Abrufen des Speichers genutzt. Eine Kombination von Addierer und Komparator kann für das Orakel im Grover-Algorithmus aus Abschnitt 3.4 verwendet werden.

5.1. Addierer

Seien $|d_x\rangle$ und $|d_y\rangle$ zwei Datenregistereinträge, die addiert werden sollen. Dabei beschreiben die beiden Einträge beispielsweise zwei Längen von Wegen. Die Addition wird im Quantenalgorithmus benutzt, um aus zwei kurzen Wegen einen langen Weg zusammenzusetzen. Das Ziel des Addierers ist somit, zwei Werte aus dem Datenregister zu addieren, um mit der Summe weiterarbeiten zu können.

Sei n_x die Anzahl an Qubits, die zur Darstellung von $|d_x\rangle$ benötigt wird, und n_y die zur Darstellung von $|d_y\rangle$. Man kann also auch $|d_x\rangle = |d_{x_{n_x-1}} \dots d_{x_1} d_{x_0}\rangle$ und analoges für $|d_y\rangle$ schreiben. In die oberen Qubits des Schaltkreises werden die Datenregister $|d_x\rangle$ und $|d_y\rangle$ injiziert. Dazu werden pro Datenregistereintrag $\max\{n_x, n_y\}$ Qubits gebraucht, da beide Datenregistereinträge die gleiche Anzahl an Qubits zum Addieren benötigen. Ist dies nicht erfüllt, gilt also beispielsweise $n_x < n_y$, werden bei dem Datenregistereintrag mit weniger Qubits weitere Qubits benötigt. Es kann dann für den Fall $n_x < n_y$ $|d_x\rangle$ als $|0 \dots 0\ d_{x_{n_x-1}} \dots d_{x_1} d_{x_0}\rangle$ mit $n_y - n_x$ 0-en injiziert werden. Das Ergebnis der Addition dieser beiden Einträge, beschrieben durch den Zustand $|d\rangle = |d_{n-1} \dots d_0\rangle$, hat eine Länge von $n = \max\{n_x, n_y\} + 1$. Um in dem Schaltkreis das Ergebnis berechnen zu können, werden unterhalb der Qubits für die Datenregistereinträge $\max\{n_x, n_y\} + 1$ Qubits im Zustand $|0\rangle$ präpariert.

Der Addierer setzt sich für den Fall $n_x = n_y = 1$ aus zwei CNOT-Gattern und einem Toffoli-Gatter zusammen. In die ersten beiden Qubits werden die Datenregistereinträge injiziert und die weiteren beiden Qubits werden im Zustand $|0\rangle$ präpariert, damit auf diesen die Addition vollzogen werden kann. Die CNOT-Gatter addieren dabei die beiden Zustände modulo 2. Das bedeutet, dass wenn beide Eingangsqubits, die durch die Zustände $|d_x\rangle$ und $|d_y\rangle$ beschrieben werden, im Zustand $|1\rangle$ vorliegen, am Ende durch die beiden CNOT-Gatter am dritten Qubit das Ergebnis $|0\rangle$ erreicht wird. Es wird ist ein Übertrag benötigt, um das gewünschte Ergebnis zu erhalten. Dies kann durch ein Toffoli-Gatter erreicht werden. Bei zwei Eingangssignalen im Zustand $|1\rangle$ wird der Zustand des letzten

Qubits von $|0\rangle$ zu $|1\rangle$ geändert. Für die Addition von zwei Ein-Qubit-Informationen aus dem Datenregister ergibt sich ein Quantenschaltkreis, der in Abb. 5.1 dargestellt ist. Dieser Quantenschaltkreis wird auch „Halbaddierer“ genannt [1].

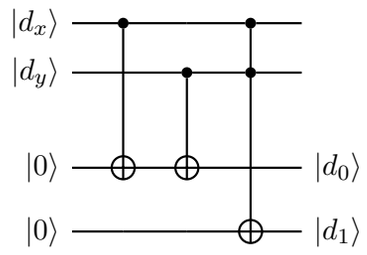


Abbildung 5.1.: Der Quantenschaltkreis addiert zwei Ein-Qubit-Informationen. Diese sind durch die Zustände der Datenregister gegeben. Das Ergebnis der Addition, das dem Zustand $|d\rangle = |d_1 d_0\rangle$ entspricht, wird in den unteren beiden Qubits berechnet.

Nun tritt bei einer Addition von Informationen aus dem Datenregister, die $n_x > 1$ oder $n_y > 1$ erfüllen, ein Problem auf, da bei der Addition auch mit dem Übertrag gerechnet werden muss. Somit wird statt der Addition von zwei Qubits eine Addition von drei Qubits durchgeführt. Ein Addierer, der den Übertrag mitaddiert und diesen dann mit dem Ergebnis überspeichert, wurde von Quantum Inspire [24] vorgestellt. Dieser wird in Abb. 5.2 dargestellt. Die ersten beiden Qubits, die in den Zuständen $|a\rangle$ und $|b\rangle$ vorliegen, stellen die Einträge dar, die addiert werden sollen. Das dritte Qubit beschreibt den Übertrag im Zustand $|u\rangle$, der im Anschluss an die Addition die zweite Ziffer des Ergebnisses ausgeben soll. Das vierte Qubit wird im Zustand $|0\rangle$ präpariert und stellt beim Output die erste Ziffer des Ergebnisses dar. Nun bewirkt das erste Gatter, dass sich der Zustand der ersten Ziffer des Output zu $|1\rangle$ ändert, falls sowohl $|a\rangle$ als auch $|b\rangle$ den Zustand $|1\rangle$ annehmen. Die nächsten beiden Gatter ändern den Output, falls der Zustand des Übertrags $|u\rangle$ und nur einer der Zustände $|a\rangle$ und $|b\rangle$ $|1\rangle$ annehmen. Das vierte Gatter ändert den Zustand der zweiten Ziffer des Ergebnisses und das letzte Gatter kehrt die Veränderungen am Zustand $|b\rangle$ des Inputs um. Somit addiert dieser Teil auch den Übertrag mit und kann als „Volladdierer“ bezeichnet werden [24]. Bei den Zuständen ist außerdem eine Superposition möglich, weswegen auch eine Zusammensetzung der Zustände $|0\rangle$ und $|1\rangle$ die Zustände des Ergebnisses ändert.

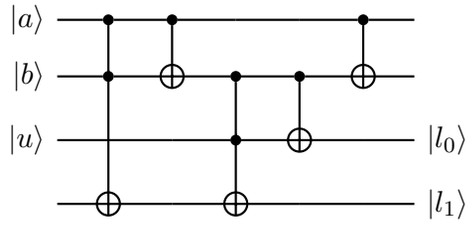


Abbildung 5.2.: Dieser Schaltkreis eines Addierers nach [24] berücksichtigt den Zustand $|u\rangle$ des Übertrags mit. Dabei werden die Zustände $|a\rangle$ und $|b\rangle$ der Datenregistereinträge mit dem Zustand $|u\rangle$ addiert. Das Ergebnis wird durch den Zustand $|l_1 l_0\rangle$ der unteren beiden Qubits beschrieben.

Um einen Addierer zu erhalten, der Daten mit $n_x > 1$ oder $n_y > 1$ addieren kann, können beide Quantenschaltkreise kombiniert werden. Ein Beispiel eines solchen Schaltkreises für eine Addition zweier Zwei-Qubit-Informationen ist in Abb. 5.3 zu sehen. Der erste Teil beschreibt hierbei den „Halbaddierer“ und der zweite Teil des Quantenschaltkreises den „Volladdierer“. Um aufzuzeigen, wie sich dieser Schaltkreis bei größeren Systemen, also beispielsweise der Addition zweier Drei-Qubit-Informationen, verhält, wurde ein größerer Quantenschaltkreis im Anhang A.4 dargestellt.

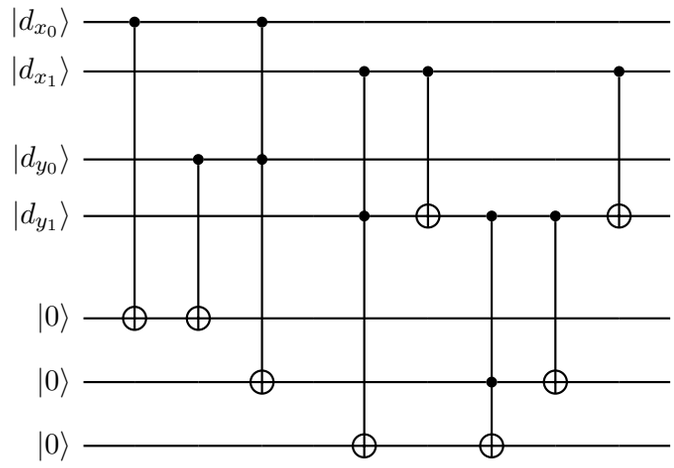


Abbildung 5.3.: Zu sehen ist ein Addierer für zwei Zwei-Qubit-Informationen. Die ersten zwei Qubits beschreiben den Zustand $|d_x\rangle$, die weiteren zwei Qubits den Zustand $|d_y\rangle$. Der Zustand des Ergebnisses wird durch die letzten drei Qubits beschrieben. Bei dem Schaltkreis handelt es sich um eine Kombination des „Halbaddierers“ (linke drei Gatter) mit dem „Volladdierer“ (rechte fünf Gatter).

Für den dargestellten Addierer werden neben den Qubits für die Datenregistereinträge weitere $\max\{n_x, n_y\} + 1$ Qubits für das Ergebnis benötigt. Neben diesem Addierer gibt es weitere Addierer, die weniger Qubits benötigen. Eines davon ist der QFT-Addierer, also der „**Q**uantum **F**ourier **T**ransform“-Addierer. Dieser benötigt statt $\max\{n_x, n_y\} + 1$ zusätzlicher Qubits nur einen weiteren Qubit [25].

5.2. Komparator

Im Folgenden wird ein Quantenschaltkreis für den Komparator vorgestellt. Die Aufgabe eines Komparators ist der Vergleich von Werten, die im Datenregister vorzufinden sind. Dabei soll durch den benötigten Komparator das Minimum wie in dem Algorithmus aus Abschnitt 3.5.1 gefunden werden. In diesem Algorithmus soll ein Wert genau dann als neuer Schwellenwert gesetzt werden, wenn dieser kleiner ist als der aktuelle Schwellenwert. Somit wird der Wert aus dem Datenregister, beschrieben durch den Zustand $|d\rangle$, mit dem aktuellen Schwellenwert, der durch den Zustand $|s\rangle$ beschrieben wird, verglichen. Ziel ist es, dass wenn der Wert des Zustandes $|d\rangle$ kleiner als der von $|s\rangle$ ist, $|1\rangle$ als Output ausgegeben wird. Dies entspricht dem Ergebnis der Bedingung C des Grover-Algorithmus aus Abschnitt 3.4. In dem Paper von Wang et al. werden zwei Formeln angegeben, die

für den digitalen Komparator angewandt werden können [26]. Dabei handelt es sich um die Formeln (5.1) und (5.2).

$$(A > B)_{out} = \sum_{i=0}^{n-1} \left(A_i \overline{B_i} \left(\prod_{j=i+1}^{n-1} P_j \right) \right) \tag{5.1}$$

$$P_j = \overline{A_j \oplus B_j} \tag{5.2}$$

Im Gegensatz zu Abschnitt 3.4 bedeutet $\overline{B_i}$ hier nicht die binäre Darstellung von B_i, sondern die Negation dieses Wertes. Da mit den Zuständen $|0\rangle$ und $|1\rangle$ gearbeitet wird, bedeutet $\overline{0} = 1$ und $\overline{1} = 0$.

In dem Fall, der in dieser Arbeit betrachtet wird, wird überprüft, ob $d < s$ gilt. Aufgrund dessen lassen sich die Formeln anpassen zu Gleichung (5.3) und (5.4).

$$\begin{aligned}(s > d)_{out} &= \sum_{i=0}^{n-1} \left(s_i \overline{d_i} \left(\prod_{j=i+1}^{n-1} P_j \right) \right) \\ &= s_{n-1}\overline{d_{n-1}} + P_{n-1}s_{n-2}\overline{d_{n-2}} + \cdots + P_{n-1} \cdots P_1 s_0 \overline{d_0} \qquad (5.3) \\ P_j &= \overline{s_j \oplus d_j} \\ &= s_j d_j + \overline{s_j}\,\overline{d_j} \qquad (5.4)\end{aligned}$$

s und d setzen sich auch in diesem Fall aus mehreren Qubits zusammen, weswegen $|s\rangle = |s_{n-1} \dots s_1 s_0\rangle$ und $|d\rangle = |d_{n-1} \dots d_1 d_0\rangle$ gilt. Wie auch in Abschnitt 5.1 müssen $|s\rangle$ und $|d\rangle$ hierbei aus der gleichen Anzahl an Qubits, beschrieben durch n, zusammengesetzt sein. Wenn dies nicht der Fall ist kann man wie in Abschnitt 5.1 vorgehen. Nun funktioniert die Berechnung von Gleichung (5.3) wie folgt: $s_i\overline{d_i}$ ergibt genau dann 1, wenn $s_i = 1$ und $d_i = 0$ gilt. P_j ergibt immer genau dann 1, wenn $s_j = d_j$ gilt. Aus diesen beiden Rechenschritten kann die Formel für den digitalen Komparator zusammengesetzt werden. Zuerst wird die erste Ziffer von s und d miteinander verglichen. Hierbei kann man drei verschiedene Fälle betrachten.

Der erste Fall ist $s_{n-1} < d_{n-1}$, also $s_{n-1} = 0$ und $d_{n-1} = 1$ und somit auch $s < d$. In diesem Fall gilt $s_{n-1}\overline{d_{n-1}} = 0$ und auch $P_{n-1} = s_{n-1}d_{n-1} + \overline{s_{n-1}}\,\overline{d_{n-1}} = 0$. Somit folgt für Gleichung (5.3), dass der erste Term durch $s_{n-1}\overline{d_{n-1}} = 0$ 0 wird und alle weiteren Terme durch $P_{n-1} = 0$ ebenfalls 0 werden. Es folgt also insgesamt $(s > d)_{out} = 0$. Dies entspricht dem erwarteten Ergebnis, da $s < d$ angenommen wurde.

Der zweite Fall betrachtet $s_{n-1} > d_{n-1}$, also $s_{n-1} = 1$ und $d_{n-1} = 0$. Aus diesem Fall folgt $s > d$, weswegen das Ergebnis $(s > d)_{out} = 1$ erwartet wird. Es gilt $s_{n-1}\overline{d_{n-1}} = 1$ und $P_{n-1} = s_{n-1}d_{n-1} + \overline{s_{n-1}}\,\overline{d_{n-1}} = 0$. Der erste Term der Gleichung (5.3) ist hierbei 1, da $s_{n-1}\overline{d_{n-1}} = 1$ gilt. Die weiteren Terme entfallen durch $P_{n-1} = 0$. Somit folgt insgesamt wie erwartet $(s > d)_{out} = 1$.

Der dritte Fall betrachtet $s_{n-1} = d_{n-1}$, also $s_{n-1} = d_{n-1} = 0$ oder $s_{n-1} = d_{n-1} = 1$. In diesem Fall gilt $s_{n-1}\overline{d_{n-1}} = 0$ und $P_{n-1} = s_{n-1}d_{n-1} + \overline{s_{n-1}}\overline{d_{n-1}} = 1$. Setzt man diese Ergebnisse in Gleichung (5.3) ein, so folgt für den ersten Term eine 0 und die Formel ist von den weiteren Stellen von s und d abhängig. Es folgt Gleichung (5.5).

$$\begin{aligned}(s > d)_{out} &= \sum_{i=0}^{n-2} \left(s_i \overline{d_i} \left(\prod_{j=i+1}^{n-2} P_j \right) \right) \\ &= s_{n-2}\overline{d_{n-2}} + P_{n-2}s_{n-3}\overline{d_{n-3}} + \cdots + P_{n-2} \cdots P_1 s_0 \overline{d_0} \end{aligned} \tag{5.5}$$

In diesem Fall wird die Rechnung mit s_{n-2} und d_{n-2} wieder von vorne begonnen. Gilt hingegen $s = d$, so wird die Rechnung bis s_0 und d_0 fortgesetzt. Für den letzten Schritt gilt $s_0\overline{d_0} = 0$, weswegen $(s > d)_{out} = 0$ erzielt wird.

Somit erfüllt Gleichung (5.3) die Bedingungen, die der digitale Komparator erfüllen muss. Es kann also anhand der Gleichung ein Quantenschaltkreis aufgebaut werden. In die ersten n Qubits wird dabei $|d\rangle = |d_{n-1} \dots d_1 d_0\rangle$ und in die weiteren n Qubits der Zustand des Schwellenwertes $|s\rangle = |s_{n-1} \dots s_1 s_0\rangle$ injiziert. Die folgenden $n-1$ Qubits werden für die Berechnung von $|P_j\rangle$ genutzt und dafür im Zustand $|0\rangle$ präpariert. Durch eine Anwendung von zwei Toffoli-Gattern kann der Zustand der Qubits zu $|P_j\rangle$ geändert werden. Dies geschieht analog zu Gleichung (5.4). Die nächsten n Qubits werden ebenfalls im Zustand $|0\rangle$ präpariert. Durch Anwendung von Toffoli-Gattern werden ihre Zustände zu den Summanden aus Gleichung (5.3) geändert. Hierbei wird der Zustand des i-ten Qubits dieser n Qubits genau dann von $|0\rangle$ zu $|1\rangle$ geändert, wenn $|d_{n-i}\rangle = |0\rangle$, $|s_{n-i}\rangle = |1\rangle$ und $|P_{n-i+1}\rangle = \cdots = |P_{n-2}\rangle = |P_{n-1}\rangle \overset{!}{=} |1\rangle$ gilt. Das letzte Qubit, das ebenfalls im Zustand $|0\rangle$ präpariert wird, stellt die Addition der Terme aus Gleichung (5.3) dar. Dafür werden mehrere CNOT-Gatter angewandt. Ein Beispiel für diesen Schaltkreis findet sich in Abb. 5.4 wieder. Hierbei wird $|d\rangle$ mit $|s\rangle$ verglichen, wobei $n = 2$ gilt. Dabei handelt es sich um die kleinste Anzahl an Qubits, die in dem Schaltkreis betrachtet wird. Dies liegt daran, dass der Vergleich in dem Algorithmus aus Abschnitt 4.2 erst nach einer Addition von Wegen vorgenommen wird. In der Addition wird als kleinste Anzahl an Qubits $n_x = 1$ und $n_y = 1$ genutzt. Das führt dazu, dass das Ergebnis sich aus mindestens $n = 2$ Qubits zusammensetzt. Aufgrund dessen wird dieser minimale Fall hier abgebildet.

In dem gesamten Schaltkreis gilt es zu beachten, dass die Qubits auch mehrere Zustände in Superposition annehmen können. Die Rechnung anhand von Gleichung (5.3) wird somit für mehrere Vergleiche zeitgleich angewandt.

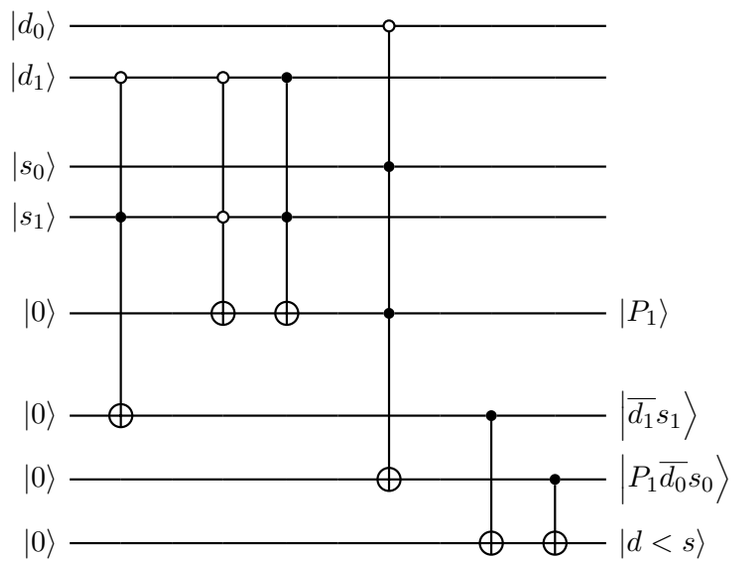


Abbildung 5.4.: Dargestellt ist ein Komparator zweier Zwei-Qubit-Informationen. Es soll geprüft werden, ob die Zustände der ersten beiden Qubits einem kleineren Wert als die Zustände der weiteren beiden Qubits entsprechen. Das erste Gatter berechnet $\left|\overline{d_1}s_1\right\rangle$, die weiteren beiden Gatter $|P_1\rangle$ und das vierte Gatter $\left|P_1\overline{d_0}s_0\right\rangle$. Die letzten Gatter addieren die Ergebnisse und berechnen, ob $d < s$ gilt.

Nun kann man den Addierer und den Komparator zusammensetzen. Wie dies aussieht wird im Anhang A.5 gezeigt.

5.3. qRAM

Die Funktionsweise des qRAM wurde bereits in Abschnitt 3.5.3 beschrieben. Dabei beschreibt $|j\rangle = |j_{n-1} \dots j_1 j_0\rangle$ die Zustände des Adressregisters. Dieses Adressregister nimmt die ersten n Qubits des Quantenschaltkreises ein. Es aktiviert die Quantensteuerleitungen, die die 2^n Speicherorte beschreiben, die in Abb. 3.12 den unteren Enden des Baumdiagrammes entsprechen. Die folgenden 2^n Qubits beschreiben somit die Speicherorte und werden im Zustand $|0\rangle$ präpariert. Allerdings wechseln sie ihren Zustand, wenn das Adressregister sie abruft. Das erste der 2^n Qubits beschreibt den Zustand $|0 \dots 00\rangle$ des Adressregisters, das zweite den Zustand $|0 \dots 01\rangle$ bis zum 2^n-ten Qubit, das den Zustand $|1 \dots 11\rangle$ beschreibt. Wie diese Adressierung bei 2 Qubits ausschauen kann ist in Abb. 5.5 dargestellt.

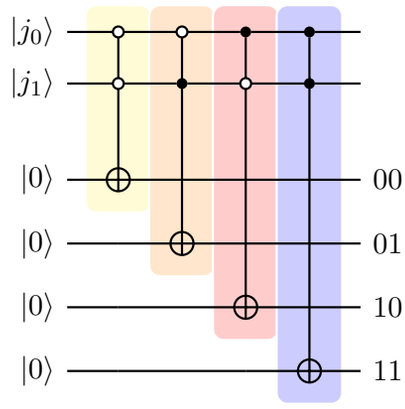


Abbildung 5.5.: Abgebildet ist die Adressierung der Speicherorte durch das Adressregister für n=2. Dabei stellen die ersten beiden Qubits das Adressregister dar und die weiteren vier Qubits die Speicherorte. Der erste Speicherort stellt beispielsweise den Wert 00 dar. Wenn das Adressregister nun im Zustand $|00\rangle$ vorliegt, wird der Zustand des ersten Speicherorts durch ein Toffoli-Gatter von $|0\rangle$ zu $|1\rangle$ geändert. Analog verläuft es mit den weiteren Speicherorten.

Allerdings setzen sich die invertierten Toffoli-Gatter im Vergleich zu normalen Toffoli-Gattern aus mehreren Gattern zusammen (siehe Abschnitt 2.2.2). Um die Anzahl an Gattern zu minimieren, kann man den Schaltkreis wie in Abb. 5.6 ändern. Hierbei wurde farblich gekennzeichnet, welche Gatter welche Aufgabe im Vergleich zu Abb. 5.5 übernehmen.

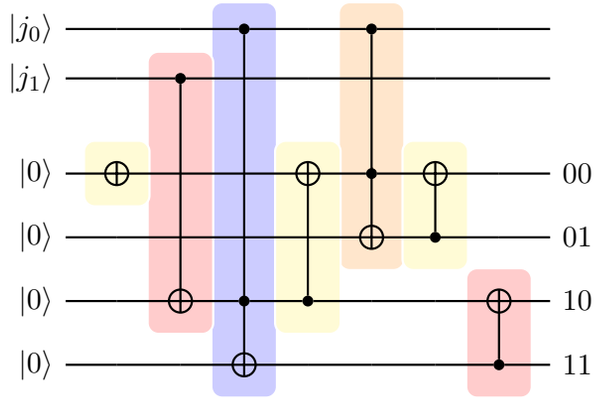


Abbildung 5.6.: Zu sehen ist die verbesserte Adressierung der Speicherorte durch das Adressregister für n=2. Hierbei werden im Vergleich zu Abb. 5.5 weniger Gater benötigt.

Alle weiteren Gatter sind hierbei von dem ersten rot gekennzeichneten Gatter abhängig. Das bedeutet, dass eine Änderung des Zustandes durch das erste rote Gatter Auswirkungen auf alle weiteren Gatter hat. Somit kann eine Änderung dieses Gatters dazu genutzt werden, um Schaltkreise mit mehr Qubits im Adressregister aufzustellen. Ein Beispiel hierzu ist in Abb. 5.7 dargestellt. Dabei wurde der Quantenschaltkreis aus Abb. 5.6 zwei Mal hintereinander aufgebaut, was durch die grauen Kästen gekennzeichnet ist. Die rot gekennzeichneten Gatter stellen die Änderung im Vergleich zu dem ersten rot

gekennzeichneten Gatter aus Abb. 5.6 dar. Beim zweiten grauen Kasten fehlt das Q-Not-Gatter, da in diesem Fall zwei Q-NOT-Gatter hintereinander angewandt werden. Dies liegt daran, dass der erste graue Kasten $|j_3\rangle = |0\rangle$ betrachtet und der zweite graue Kasten bei $|j_3\rangle = |1\rangle$ zu einer Zustandsänderung führt.

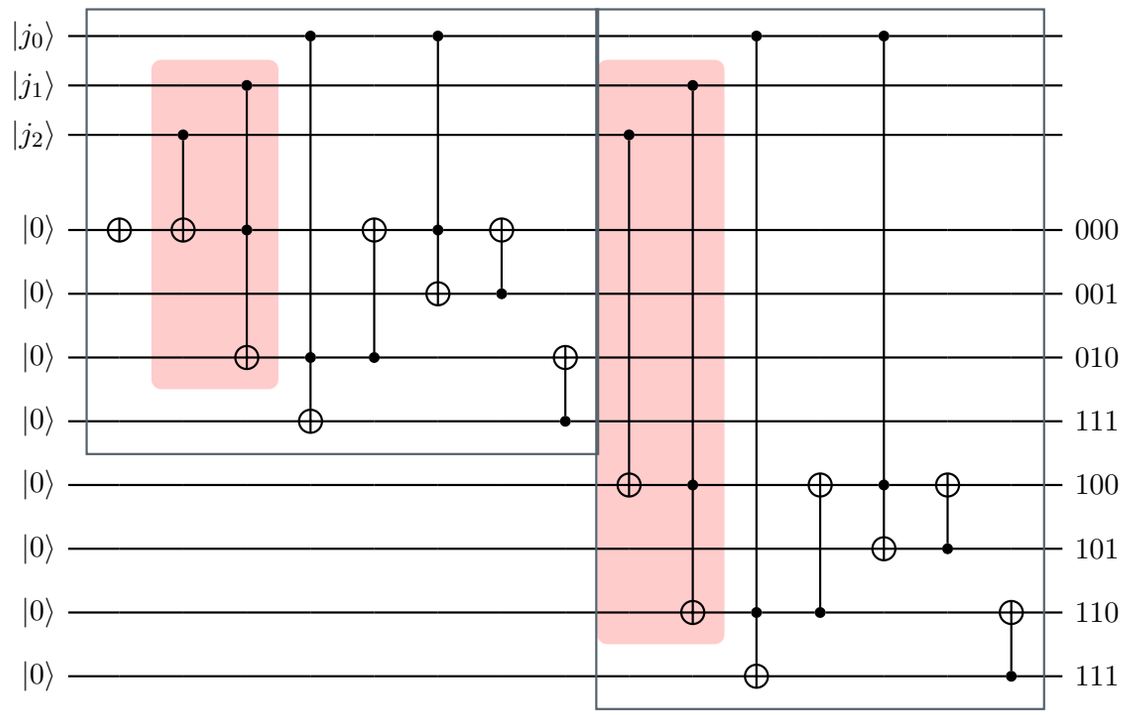


Abbildung 5.7.: Zu sehen ist die Adressierung der Speicherorte durch das Adressregister bei n=3. Die ersten drei Qubits stellen das Adressregister und die weiteren acht Qubits die Speicherorte dar. In der Abbildung wurden zwei Schaltkreise aus Abb. 5.6 hintereinander angewandt (grau markiert) und die Gatter für die Adressierung durch den Zustand $|j_2\rangle$ wurden hinzugefügt. Die Änderung im Vergleich zu dem roten Gatter aus Abb. 5.6 wurde hierbei rot markiert.

Nun muss ein Zugriff auf den Speicher geschehen. Hierbei gibt es 2^n verschiedene Speicherorte. Jeder Inhalt eines Speicherortes besteht dabei aus l Qubits. Somit lässt sich der Inhalt des Speichers an j-ter Stelle als $|d_j\rangle = |d_{jl-1} \dots d_{j1} d_{j0}\rangle$ schreiben. Der Speicher besteht insgesamt aus $2^n l$ Qubits. Es werden also $2^n l$ Qubits unterhalb des Quantenschaltkreises für die Adressierung aus Abb. 5.6 und 5.7 für den Speicher genutzt. Mit Toffoli-Gattern lässt sich aus den Speicherorten der Speicher ablesen und auf den „Quantenbus“ übertragen. Dazu werden unten im Schaltkreis l Qubits im Zustand $|0\rangle$ für den „Quantenbus“ präpariert. Insgesamt lässt sich der qRAM wie in Abb. 5.8 darstellen. In diesem Beispiel wird mit $n = l = 2$ gearbeitet.

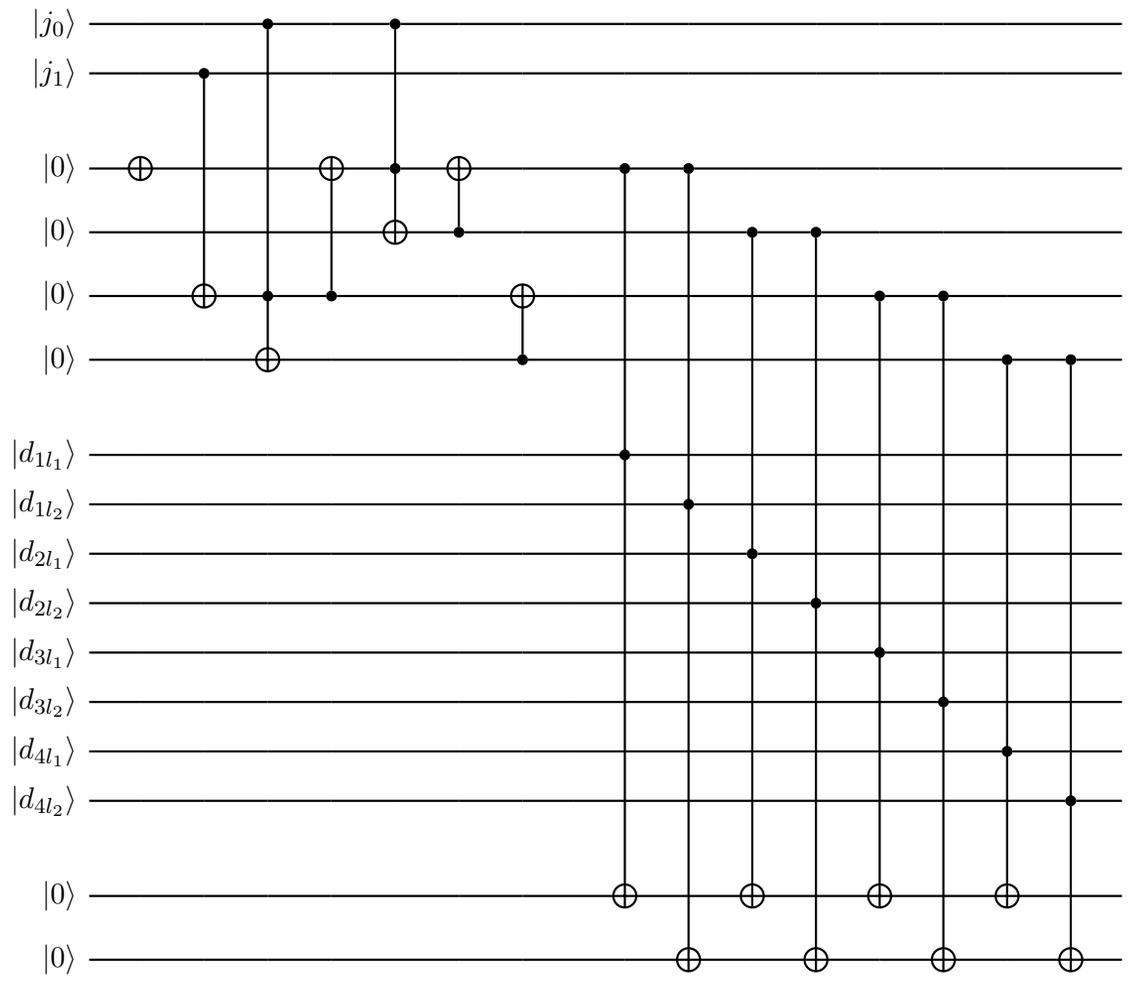


Abbildung 5.8.: Dargestellt ist ein Schaltkreis für den qRAM mit n=2 und l=2. Die ersten zwei Qubits werden für das Adressregister verwendet, die weiteren vier Qubits beschreiben die Spicherorte. Die nächsten acht Qubits beschrieben den Speicher und die letzten beiden werden für den „Quantenbus“ im Zustand $|0\rangle$ präpariert. Ein Abgriff des „Quantenbusses“ von Speicherort und Speicherinhalt durch Toffoli-Gatter führt dazu, dass dieser den Speicher auslesen kann. Durch das „No-Cloning-Theorem“ wird der Speicherinhalt nicht exakt auf den„Quantenbus“ kopiert.

6 Kapitel 6. Fazit und Ausblick

In der Bachelorarbeit wurde sich mit Quantenalgorithmen zur Lösung des TSP auseinandergesetzt. Dabei wurde in Kapitel 2 auf die Grundlagen zur Arbeit mit Quantencomputern eingegangen. Es wurde dargestellt, dass Quantencomputer gegenüber klassischen Computern einen Vorteil in der Laufzeit von verschiedenen Algorithmen bewirken können. Außerdem kann das Hinzufügen von nur einem Qubit zu einem Quantencomputer die Speichermöglichkeit verdoppeln. Bei einem klassischen Computer müsste hingegen die Anzahl an Bits verdoppelt werden (vgl. Abschnitt 2.1). In Kapitel 3 wurde betrachtet, was **NP**-schwere Probleme sind und dass Quantenalgorithmen wie der Grover-Algorithmus einen Speedup der Laufzeit bei der Lösung dieser realisieren können. Auch der Algorithmus zur Quantenminimumssuche führt zu einem Speedup, da für die Suche des Minimums einer Liste mit dem Algorithmus eine Laufzeit von $\mathcal{O}(\sqrt{N})$ benötigt wird. Es wird also ein quadratischer Speedup erreicht. Für die Lösung des TSP wurde in Abschnitt 4.1 weiterhin ein klassischer Algorithmus von Held und Karp vorgestellt, der die beste klassische Laufzeit zur Lösung dieses Problems erreicht. Die Erkenntnisse über die verschiedenen Quantenalgorithmen und der in Abschnitt 4.1 beschriebene klassische Algorithmus wurden genutzt, um den Quantenalgorithmus von Ambainis et al. in Abschnitt 4.2 zu verstehen. Dabei wurde herausgefunden, dass der Algorithmus von Ambainis et al. in der beschriebenen Form einige Rechenschritte vernachlässigt. Um das Problem zu beheben, wurde in Abschnitt 4.3 eine Anpassung des Algorithmus beschrieben. Es wurden drei mögliche Anpassungsmöglichkeiten für den Algorithmus vorgeschlagen und die mit der besten Laufzeit bestimmt. Daraus ergab sich, dass die Laufzeit ohne Betrachtung polynomieller Komplexitäten im Vergleich zu dem Algorithmus von Ambainis et al. unverändert bleibt. Außerdem wurde bestätigt, dass die im Algorithmus von Ambainis et al. genutzte Halbierung der Wege und Wiederholung durch drei Rekursionsschritte die beste Laufzeit erzielt. Im letzten Kapitel wurde vorgestellt, wie sich ein Addierer, ein Komparator und der qRAM in einen Quantencomputer implementieren lassen. Dafür wurden für diese Quantenschaltkreise vorgestellt.

Um den betrachteten Quantenalgorithmus weiter anzupassen, kann eine Unterscheidung zwischen den Anzahlen an Ecken gemacht werden. In dem betrachteten Fall wurde angenommen, dass die Anzahl an Ecken n durch vier teilbar ist. Der Algorithmus kann auf die vier Fälle $n \mod 4 = \{0, 1, 2, 3\}$ angepasst werden. In dem in Abschnitt 4.2 betrachteten Fall handelt es sich um den Fall $n \mod 4 = 0$. Außerdem kann auf Basis des angepassten Algorithmus aus Abschnitt 4.3 und der in Kapitel 5 beschriebenen Quantenschaltkreise eine Implementierung des Quantenalgorithmus vorgenommen werden.

Insgesamt folgt, dass der Quantenalgorithmus einen Speedup in der Laufzeit gegenüber den klassischen Algorithmen geben kann. Dies zeigt die Nützlichkeit von Quantencomputern zur Lösung von **NP**-schweren Problemen auf. Dabei kann das Travelling Salesman Problem beispielsweise zur Optimierung von Transportwegen von Unternehmen genutzt werden. Diese Optimierung kann zu einer Senkung von Kosten oder anderen Faktoren für Unternehmen führen.

A Anhang

Anhang A.

A.1. Beispiel für den klassischen Algorithmus der dynamischen Programmierung

In diesem Beispiel wird der klassische Algorithmus von Held und Karp aus Abschnitt 4.1 an einem Beispiel über fünf Städte durchgeführt. Betrachte die Matrix A, welche die Entfernungen zwischen den einzelnen Städten $\{1, 2, 3, 4, 5\}$ angibt:

$$A = \begin{bmatrix} 0 & 5 & 10 & 16 & 11 \\ 9 & 0 & 7 & 12 & \infty \\ 12 & 5 & 0 & 15 & \infty \\ 15 & 14 & 9 & 0 & 9 \\ 9 & \infty & \infty & 12 & 0 \end{bmatrix}$$

Im ersten Schritt wird $C(S, l)$ mit $n(S) = 1$ anhand Gleichung (4.1) berechnet.

$$\begin{aligned} C(\{2\}, 2) &= a_{12} = 5 \\ C(\{3\}, 3) &= a_{13} = 10 \\ C(\{4\}, 4) &= a_{14} = 16 \\ C(\{5\}, 5) &= a_{15} = 11 \end{aligned}$$

Nun kann mit Gleichung (4.2) der Weg mit einer weiteren Stadt berechnet werden. Dabei folgen die folgenden Wege:

$$\begin{aligned} C(\{2,3\}, 3) &= \min\{C(\{2\}, 2) + a_{23}\} = \min\{5 + 7\} = 12 \\ C(\{2,4\}, 4) &= \min\{C(\{2\}, 2) + a_{24}\} = \min\{5 + 12\} = 17 \\ C(\{2,5\}, 5) &= \min\{C(\{2\}, 2) + a_{25}\} = \min\{5 + \infty\} = \infty \end{aligned}$$

$$\begin{aligned} C(\{2,3\}, 2) &= \min\{C(\{3\}, 3) + a_{32}\} = \min\{10 + 5\} = 15 \\ C(\{3,4\}, 4) &= \min\{C(\{3\}, 3) + a_{34}\} = \min\{10 + 15\} = 25 \\ C(\{3,5\}, 5) &= \min\{C(\{3\}, 3) + a_{35}\} = \min\{10 + \infty\} = \infty \end{aligned}$$

$$\begin{aligned} C(\{2,4\}, 2) &= \min\{C(\{4\}, 4) + a_{42}\} = \min\{16 + 14\} = 30 \\ C(\{3,4\}, 3) &= \min\{C(\{4\}, 4) + a_{43}\} = \min\{16 + 9\} = 25 \\ C(\{4,5\}, 5) &= \min\{C(\{4\}, 4) + a_{45}\} = \min\{16 + 9\} = 25 \end{aligned}$$

$$
\begin{aligned}
C(\{2,5\},2) &= \min\{C(\{5\},5) + a_{52}\} = \min\{11+\infty\} = \infty \\
C(\{3,5\},3) &= \min\{C(\{5\},5) + a_{53}\} = \min\{11+\infty\} = \infty \\
C(\{4,5\},4) &= \min\{C(\{5\},5) + a_{54}\} = \min\{11+12\} = 23
\end{aligned}
$$

Auf Basis der Ergebnisse kann man diese Berechnung mit drei Städten neben der Stadt 1 durchführen. Dabei gilt:

$$
\begin{aligned}
C(\{2,3,4\},4) &= \min\{C(\{2,3\},2) + a_{24}, C(\{2,3\},3) + a_{34}\} \\
&= \min\{15+12, 12+15\} = 27 \\
C(\{2,3,5\},5) &= \min\{C(\{2,3\},2) + a_{25}, C(\{2,3\},3) + a_{35}\} \\
&= \min\{15+\infty, 12+\infty\} = \infty
\end{aligned}
$$

$$
\begin{aligned}
C(\{2,3,4\},3) &= \min\{C(\{2,4\},2) + a_{23}, C(\{2,4\},4) + a_{43}\} \\
&= \min\{30+7, 17+9\} = 26 \\
C(\{2,4,5\},5) &= \min\{C(\{2,4\},2) + a_{25}, C(\{2,4\},4) + a_{45}\} \\
&= \min\{30+\infty, 17+9\} = 26
\end{aligned}
$$

$$
\begin{aligned}
C(\{2,3,5\},3) &= \min\{C(\{2,5\},2) + a_{23}, C(\{2,5\},5) + a_{53}\} \\
&= \min\{\infty+7, \infty+\infty\} = \infty \\
C(\{2,4,5\},4) &= \min\{C(\{2,5\},2) + a_{24}, C(\{2,5\},5) + a_{54}\} \\
&= \min\{\infty+12, \infty+12\} = \infty
\end{aligned}
$$

$$
\begin{aligned}
C(\{2,3,4\},2) &= \min\{C(\{3,4\},3) + a_{32}, C(\{3,4\},4) + a_{42}\} \\
&= \min\{25+5, 25+14\} = 30 \\
C(\{3,4,5\},5) &= \min\{C(\{3,4\},3) + a_{35}, C(\{3,4\},4) + a_{45}\} \\
&= \min\{25+\infty, 25+9\} = 34
\end{aligned}
$$

$$
\begin{aligned}
C(\{2,3,5\},2) &= \min\{C(\{3,5\},3) + a_{32}, C(\{3,5\},5) + a_{52}\} \\
&= \min\{\infty+5, \infty+\infty\} = \infty \\
C(\{3,4,5\},4) &= \min\{C(\{3,5\},3) + a_{34}, C(\{3,5\},5) + a_{54}\} \\
&= \min\{\infty+15, \infty+12\} = \infty
\end{aligned}
$$

$$
\begin{aligned}
C(\{2,4,5\},2) &= \min\{C(\{4,5\},4) + a_{42}, C(\{4,5\},5) + a_{52}\} \\
&= \min\{23+14, 25+\infty\} = 37 \\
C(\{3,4,5\},3) &= \min\{C(\{4,5\},4) + a_{43}, C(\{4,5\},5) + a_{53}\} \\
&= \min\{23+9, 25+\infty\} = 32
\end{aligned}
$$

Nun kann man alle Städte hinzufügen, die sich mit Gleichung (4.2) berechnen lassen.

$$\begin{aligned} C(\{2,3,4,5\},5) &= \min\{C(\{2,3,4\},2)+a_{25}, C(\{2,3,4\},3)+a_{35}, C(\{2,3,4\},4)+a_{45}\} \\ &= \min\{30+\infty, 26+\infty, 27+9\} = 36 \end{aligned}$$

$$\begin{aligned} C(\{2,3,4,5\},4) &= \min\{C(\{2,3,5\},2)+a_{24}, C(\{2,3,5\},3)+a_{34}, C(\{2,3,5\},5)+a_{54}\} \\ &= \min\{\infty+12, \infty+15, \infty+12\} = \infty \end{aligned}$$

$$\begin{aligned} C(\{2,3,4,5\},3) &= \min\{C(\{2,4,5\},2)+a_{23}, C(\{2,4,5\},4)+a_{43}, C(\{2,4,5\},5)+a_{53}\} \\ &= \min\{37+7, \infty+9, 26+\infty\} = 44 \end{aligned}$$

$$\begin{aligned} C(\{2,3,4,5\},2) &= \min\{C(\{3,4,5\},3)+a_{32}, C(\{3,4,5\},4)+a_{42}, C(\{3,4,5\},5)+a_{52}\} \\ &= \min\{32+5, \infty+14, 34+\infty\} = 37 \end{aligned}$$

Für die Berechnung des letzten Schrittes ist die Gleichung (4.3) nötig. Hier wird an die vier verschiedenen Wege der Rückweg zur Stadt 1 hinzugefügt. Dabei ergibt sich der folgende kürzeste Weg:

$$\begin{aligned} \mathcal{C} &= \min\{C(\{2,3,4,5\},2)+a_{21}, C(\{2,3,4,5\},3)+a_{31}, \\ &\qquad C(\{2,3,4,5\},4)+a_{41}, C(\{2,3,4,5\},5)+a_{51}\} \\ &= \min\{37+9, 44+12, \infty+15, 36+9\} = 45 \end{aligned}$$

Es muss herausgefunden werden, welcher Weg für diese Entfernung zurückgelegt wurde. Der Weg wird durch $1 \to i_2 \to i_3 \to i_4 \to i_5 \to 1$ beschrieben. Die letzte Stadt kann durch Gleichung (4.4)herausgefunden werden. Dabei ergeben sich folgende Möglichkeiten:

$$\begin{aligned} \mathcal{C} &= C(\{2,3,4,5\},2)+a_{21} = 37+9 = 46 \\ \mathcal{C} &= C(\{2,3,4,5\},3)+a_{31} = 44+12 = 56 \\ \mathcal{C} &= C(\{2,3,4,5\},4)+a_{41} = \infty+15 = \infty \\ \mathcal{C} &= C(\{2,3,4,5\},5)+a_{51} = 36+9 = 45 \end{aligned}$$

Es folgt, dass für die letzte Stadt $i_5 = 5$ gilt. Berechne nun $i_4 \in \{2,3,4\}$ durch Gleichung (4.5). Dabei wird mit $p = 4$ gerechnet.

$$\begin{aligned} C(\{2,3,4,5\},5) &= C(\{2,3,4\},2)+a_{25} = 30+\infty = \infty \\ C(\{2,3,4,5\},5) &= C(\{2,3,4\},3)+a_{35} = 26+\infty = \infty \\ C(\{2,3,4,5\},5) &= C(\{2,3,4\},4)+a_{45} = 27+9 = 36 \end{aligned}$$

Durch die Rechnung wurde i_4 als Stadt 4 identifiziert. Nun kann die gleiche Rechnung zum Bestimmen von $i_3 \in \{2, 3\}$ verwendet werden. Hierbei wird mit $p = 3$ gerechnet.

$$C(\{2,3,4\},4) = C(\{2,3\},2) + a_{24} = 15 + 12 = 27$$
$$C(\{2,3,4\},4) = C(\{2,3\},3) + a_{34} = 12 + 15 = 27$$

Es ergibt sich, dass sowohl die Stadt 2 als auch die Stadt 3 eine Lösung für i_3 ergeben können. Somit ergeben sich zwei Lösungswege, welche die gleiche Entfernung haben. Berechne im letzten Schritt nun noch i_2. Rechne wieder mit Gleichung (4.5).

Für $i_3 = 3$:
$$C(\{2,3\},3) = C(\{2\},2) + a_{23} = 5 + 7 = 12$$
Für $i_3 = 2$:
$$C(\{2,3\},2) = C(\{3\},3) + a_{32} = 10 + 5 = 15$$

Es ergibt sich falls $i_3 = 2$ $i_2 = 3$ und falls $i_3 = 3$ gilt $i_2 = 2$. Es folgen folgende Permutationen als mögliche Wege:

$$(1\ 2\ 3\ 4\ 5)$$
$$(1\ 3\ 2\ 4\ 5)$$

Somit wurde mit der klassischen Programmierung ein Weg gefunden, um das TSP zu lösen.

A.2. Beispiel für den angepassten Quantenalgorithmus auf Basis der dynamischen Programmierung

Im Folgenden wird anhand des ersten Algorithmus aus Abschnitt 4.3 eine Lösung für das TSP berechnet, die die Weglänge einer Rundreise zwischen vier Städten minimieren soll. Betrachte die Matrix A, welche die Entfernungen zwischen den einzelnen Städten $\{1, 2, 3, 4\}$ angibt. Diese ist ähnlich zu der aus Anhang A.1. Allerdings funktioniert der Algorithmus nur für $\frac{n}{4} \in \mathbb{N}$, weswegen statt fünf Städten in diesem Beispiel vier Städte betrachtet werden.

$$A = \begin{bmatrix} 0 & 5 & 10 & 16 \\ 9 & 0 & 7 & 12 \\ 12 & 5 & 0 & 15 \\ 15 & 14 & 9 & 0 \end{bmatrix}$$

Wie im Algorithmus oben beschrieben, funktioniert dieser Algorithmus für $n \geq 13$. Für $n < 13$ muss Schritt c) angepasst werden,wie auch in diesem Fall, da in diesem Beispiel $n = 4$ gilt.

Im ersten Schritt wird die dynamische Programmierung bis $|S| \leq \frac{(1-0.055)n}{4}$, also $|S| < 1$, durchgeführt. Dabei erfüllt nur $|S| = 0$ diese Bedingung. Dieser Fall ist trivial und muss nicht berechnet werden.

Nun wird Schritt a) der Anpassung des zweiten Schrittes von Ambainis et al. Algorithmus durchgeführt. Dabei wird mit

$$\min_{\substack{S \subset V \\ |S|=n/2+1=3}} \min_{\substack{u,v \in S \\ u \neq v}} \{f(S,u,v) + f((V \setminus S) \cup \{u,v\}, v, u)\}$$

gerechnet. Es folgt:

$$\begin{aligned}\min\{&f(\{1,2,3\},1,2) + f(\{1,2,4\},2,1),\\ &f(\{1,2,3\},1,3) + f(\{1,3,4\},3,1),\\ &f(\{1,2,4\},1,2) + f(\{1,2,3\},2,1),\\ &f(\{1,2,4\},1,4) + f(\{1,3,4\},4,1),\\ &f(\{1,3,4\},1,3) + f(\{1,2,3\},3,1),\\ &f(\{1,3,4\},1,4) + f(\{1,2,4\},4,1)\}\end{aligned}$$

Anhand von Schritt b) muss $f(S,u,v)$ mit $|S| = 3$ berechnet werden. Dafür wird Gleichung (4.8) mit $k = \frac{n}{4} + 1 = 2$ angewandt.

$$\begin{aligned}f(\{1,2,3\},1,2) &= \min\{f(\{1,3\},1,3) + f(\{2,3\},3,2)\}\\ f(\{1,2,3\},1,3) &= \min\{f(\{1,2\},1,2) + f(\{2,3\},2,3)\}\\ f(\{1,2,3\},2,1) &= \min\{f(\{2,3\},2,3) + f(\{1,3\},3,1)\}\\ f(\{1,2,3\},3,1) &= \min\{f(\{2,3\},3,2) + f(\{1,2\},2,1)\}\end{aligned}$$

$$\begin{aligned}f(\{1,2,4\},1,2) &= \min\{f(\{1,4\},1,4) + f(\{2,4\},4,2)\}\\ f(\{1,2,4\},1,4) &= \min\{f(\{1,2\},1,2) + f(\{2,4\},2,4)\}\\ f(\{1,2,4\},2,1) &= \min\{f(\{2,4\},2,4) + f(\{1,4\},4,1)\}\\ f(\{1,2,4\},4,1) &= \min\{f(\{2,4\},4,2) + f(\{1,2\},2,1)\}\end{aligned}$$

$$\begin{aligned}f(\{1,3,4\},1,3) &= \min\{f(\{1,4\},1,4) + f(\{3,4\},4,3)\}\\ f(\{1,3,4\},1,4) &= \min\{f(\{1,3\},1,3) + f(\{3,4\},3,4)\}\\ f(\{1,3,4\},3,1) &= \min\{f(\{3,4\},3,4) + f(\{1,4\},4,1)\}\\ f(\{1,3,4\},4,1) &= \min\{f(\{3,4\},4,3) + f(\{1,3\},3,1)\}\end{aligned}$$

Diese Rechnung muss nicht weiter aufgeteilt werden, da die Entfernungen für $k = 2$ aus der Matrix A abgelesen werden können. Schritt c) entfällt somit. Für Schritt b) folgt:

$$\begin{aligned}
f(\{1,2,3\},1,2) &= \min\{f(\{1,3\},1,3) + f(\{2,3\},3,2)\} \\
&= \min\{10+5\} = 15 \\
f(\{1,2,3\},1,3) &= \min\{f(\{1,2\},1,2) + f(\{2,3\},2,3)\} \\
&= \min\{5+7\} = 12 \\
f(\{1,2,3\},2,1) &= \min\{f(\{2,3\},2,3) + f(\{1,3\},3,1)\} \\
&= \min\{7+12\} = 19 \\
f(\{1,2,3\},3,1) &= \min\{f(\{2,3\},3,2) + f(\{1,2\},2,1)\} \\
&= \min\{5+9\} = 14
\end{aligned}$$

$$\begin{aligned}
f(\{1,2,4\},1,2) &= \min\{f(\{1,4\},1,4) + f(\{2,4\},4,2)\} \\
&= \min\{16+14\} = 30 \\
f(\{1,2,4\},1,4) &= \min\{f(\{1,2\},1,2) + f(\{2,4\},2,4)\} \\
&= \min\{5+12\} = 17 \\
f(\{1,2,4\},2,1) &= \min\{f(\{2,4\},2,4) + f(\{1,4\},4,1)\} \\
&= \min\{12+15\} = 27 \\
f(\{1,2,4\},4,1) &= \min\{f(\{2,4\},4,2) + f(\{1,2\},2,1)\} \\
&= \min\{14+9\} = 23
\end{aligned}$$

$$\begin{aligned}
f(\{1,3,4\},1,3) &= \min\{f(\{1,4\},1,4) + f(\{3,4\},4,3)\} \\
&= \min\{16+9\} = 25 \\
f(\{1,3,4\},1,4) &= \min\{f(\{1,3\},1,3) + f(\{3,4\},3,4)\} \\
&= \min\{10+15\} = 25 \\
f(\{1,3,4\},3,1) &= \min\{f(\{3,4\},3,4) + f(\{1,4\},4,1)\} \\
&= \min\{15+15\} = 30 \\
f(\{1,3,4\},4,1) &= \min\{f(\{3,4\},4,3) + f(\{1,3\},3,1)\} \\
&= \min\{9+12\} = 21
\end{aligned}$$

Diese Ergebnisse können nun in Schritt a) eingesetzt werden.

$$\begin{aligned}
&\min\{f(\{1,2,3\},1,2)+f(\{1,2,4\},2,1),\\
&\quad f(\{1,2,3\},1,3)+f(\{1,3,4\},3,1),\\
&\quad f(\{1,2,4\},1,2)+f(\{1,2,3\},2,1),\\
&\quad f(\{1,2,4\},1,4)+f(\{1,3,4\},4,1),\\
&\quad f(\{1,3,4\},1,3)+f(\{1,2,3\},3,1),\\
&\quad f(\{1,3,4\},1,4)+f(\{1,2,4\},4,1),\}\\
&=\min\{15+27,12+30,30+19,17+21,25+14,25+23\}\\
&=\min\{42,42,49,38,39,48\}\\
&=38
\end{aligned}$$

Es ergibt sich, dass der kürzeste Weg eine Länge von 38 besitzt.

A.3. Beispiel für die Minimumssuche

Anhand der Liste $T=\{42,42,49,38,39,48\}$ kann gezeigt werden, wie die Quantenminimumssuche funktioniert. Diese setzt sich aus den Algorithmen aus Abschnitt 3.5.1 und Abschnitt 3.5.2 zusammen. Nummeriere dabei die Listenelemente von 0 bis 5 durch. Nun kann man den Algorithmus aus Abschnitt 3.5.1 ausführen.

1. Suche $0 \leq y \leq 5$ zufällig aus. Sei dabei $y=0$.
2. a) Der Zustand wird als $\frac{1}{\sqrt{6}}(|0\rangle|0\rangle+|1\rangle|0\rangle+|2\rangle|0\rangle+|3\rangle|0\rangle+|4\rangle|0\rangle+|5\rangle|0\rangle)$ präpariert.
   Die Zustände $|3\rangle|0\rangle$ und $|4\rangle|0\rangle$ werden markiert, da sie $T[3]=38<T[0]=42$ und $T[4]=39<T[0]=42$ erfüllen.

   b) Nun wird die Grover-Suche aus Abschnitt 3.5.2 angewandt. Diese funktioniert wie folgt:

   1. $m=1$ und $\lambda=\frac{6}{5}$ werden initialisiert.
   2. Es gilt $0 \leq j < 1$, woraus $j=0$ folgt.
   3. $|\psi_0\rangle=\frac{1}{\sqrt{6}}(|0\rangle+|1\rangle+|2\rangle+|3\rangle+|4\rangle+|5\rangle)$
      $\rightarrow$ Der Grover-Algorithmus wird j=0 Mal angewandt. Aufgrund dessen wird $|3\rangle$ oder $|4\rangle$ mit einer Wahrscheinlichkeit von $\frac{1}{3}$ gefunden.
   4. Sei $|2\rangle$ das Ergebnis.
   5. $T[2]=49>T[0]=42$ $\lightning$
   6. Setze $m=\min\left\{\frac{6}{5}\cdot 1,\sqrt{6}\right\}=\frac{6}{5}$ und kehre zu Schritt 2 zurück.

2. Wähle aus $0 \leq j < \frac{6}{5}$ $j = 1$ zufällig aus.
3. Wende nun ein Mal den Grover-Algorithmus auf

   $|\psi_0\rangle = \frac{1}{\sqrt{6}} (|0\rangle + |1\rangle + |2\rangle + |3\rangle + |4\rangle + |5\rangle)$

   an. Es folgt für k_1 und l_1:

   $k_1 = \frac{1}{\sqrt{t}} \sin((2 \cdot 1 + 1)\theta)$, wobei t=2 gilt, da zwei Einträge in 1.

   markiert wurden. Es folgt $k_1 = \frac{1}{\sqrt{2}} \sin\left(3 \arcsin\left(\sqrt{\frac{2}{6}}\right)\right) \approx 0.68$.

   $l_1 = \frac{1}{\sqrt{N-t}} \cos((2 \cdot 1 + 1)\theta)$, wobei durch t=2 folgt:

   $l_1 = \frac{1}{\sqrt{4}} \cos\left(3 \arcsin\left(\sqrt{\frac{2}{6}}\right)\right) \approx -0.14$

   Somit folgt der Gesamtzustand

   $|\psi\rangle = 0.68 (|3\rangle + |4\rangle) - 0.14 (|0\rangle + |1\rangle + |2\rangle + |5\rangle)$.

   → Es wird zu $\approx 92.6\%$ der Zustand $|3\rangle$ oder $|4\rangle$ gefunden.
4. Sei $|4\rangle$ das Ergebnis.
5. $T[4] = 39 < T[0] = 42$

   → Hierbei wurde eine mögliche Lösung gefunden.

   Eine Lösung nach maximal zwei Wiederholungen wird zu

   $\frac{1}{3} + \frac{2}{3} \cdot \frac{1}{2} \cdot 0.926 + \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{1}{3} \approx 75.3\%$ gefunden.

c) Es gilt $T[4] = 39 < T[0] = 42$. Somit wurde eine mögliche Lösung gefunden und $y = 4$ wird zum neuen Schwellenindex.

a) Initialisiere $\frac{1}{\sqrt{6}} (|0\rangle|4\rangle + |1\rangle|4\rangle + |2\rangle|4\rangle + |3\rangle|4\rangle + |4\rangle|4\rangle + |5\rangle|4\rangle)$.
Der Zustand $|3\rangle|4\rangle$ wird markiert, da $T[3] = 38 < T[4] = 39$ gilt.

b) Wende wieder die Grover-Suche für eine unbekannte Anzahl an Lösungen an.

1. $m = 1$ und $\lambda = \frac{6}{5}$ werden initialisiert.
2. Es gilt $0 \leq j < 1$, woraus $j = 0$ folgt.
3. $|\psi_0\rangle = \frac{1}{\sqrt{6}} (|0\rangle + |1\rangle + |2\rangle + |3\rangle + |4\rangle + |5\rangle)$

   → Der Grover-Algorithmus wird j=0 Mal angewandt. Dadurch

   wird $|3\rangle$ mit einer Wahrscheinlichkeit von $\frac{1}{6}$ gefunden.
4. Sei $|5\rangle$ das Ergebnis.

5. $T[5] = 48 > T[4] = 39$ ↯
6. Setze $m = \min\left\{\frac{6}{5} \cdot 1, \sqrt{6}\right\} = \frac{6}{5}$ und kehre zu Schritt 2 zurück.

2. Wähle aus $0 \leq j < \frac{6}{5}$ $j = 1$ zufällig aus.
3. Wende nun ein Mal den Grover-Algorithmus auf

   $|\psi_0\rangle = \frac{1}{\sqrt{6}}(|0\rangle + |1\rangle + |2\rangle + |3\rangle + |4\rangle + |5\rangle)$

   an. Es folgt für k_1 und l_1:

   $k_1 = \frac{1}{\sqrt{t}} \sin((2 \cdot 1 + 1)\theta)$, wobei t=1 gilt, da $|3\rangle$ in 1.

   markiert wurde. Es folgt $k_1 = \sin\left(3 \arcsin\left(\frac{1}{\sqrt{6}}\right)\right) \approx 0.95$.

   $l_1 = \frac{1}{\sqrt{N-t}} \cos((2 \cdot 1 + 1)\theta)$, wobei durch t=1 folgt:

   $l_1 = \frac{1}{\sqrt{5}} \cos\left(3 \arcsin\left(\frac{1}{\sqrt{6}}\right)\right) \approx 0.14$

   Somit folgt der Gesamtzustand

   $|\psi\rangle = 0.95\,|3\rangle + 0.14\,(|0\rangle + |1\rangle + |2\rangle + |4\rangle + |5\rangle)\,.$

   → Es wird zu $\approx 90.7\%$ der Zustand $|3\rangle$ gefunden.
4. Sei $|3\rangle$ das Ergebnis.
5. $T[3] = 38 < T[4] = 39$

   → Hierbei wurde eine mögliche Lösung gefunden.

   Diese Lösung nach maximal zwei Wiederholungen wird zu

   $\frac{1}{6} + \frac{5}{6} \cdot \frac{1}{2} \cdot 0.907 + \frac{5}{6} \cdot \frac{1}{2} \cdot \frac{1}{6} \approx 61.4\%$ gefunden.

c) Es gilt $T[3] < T[4] = 39$. Somit wurde eine mögliche Lösung gefunden und $y = 3$ wird zum neuen Schwellenindex.

Nun kann der Algorithmus wiederholt werden. Allerdings wird keine neue Lösung gefunden. $T[3]$ entspricht also dem Minimum der Liste. Der Algorithmus wird abgebrochen, sobald die Laufzeit von $22.5\sqrt{6} + 1.4 \log_2^2 6 \approx 64.5$ überschritten wird.

3. Gebe 3 aus.

A.4. Quantenschaltkreis eines Addierers

In Abb. A.1 ist ein Addierer von zwei Drei-Qubit-Informationen dargestellt. Der erste Teil stellt den „Halbaddierer“ dar und die letzten beiden Teile die „Volladdierer“. Das Vorgehen wurde in Abschnitt 5.1 beschrieben.

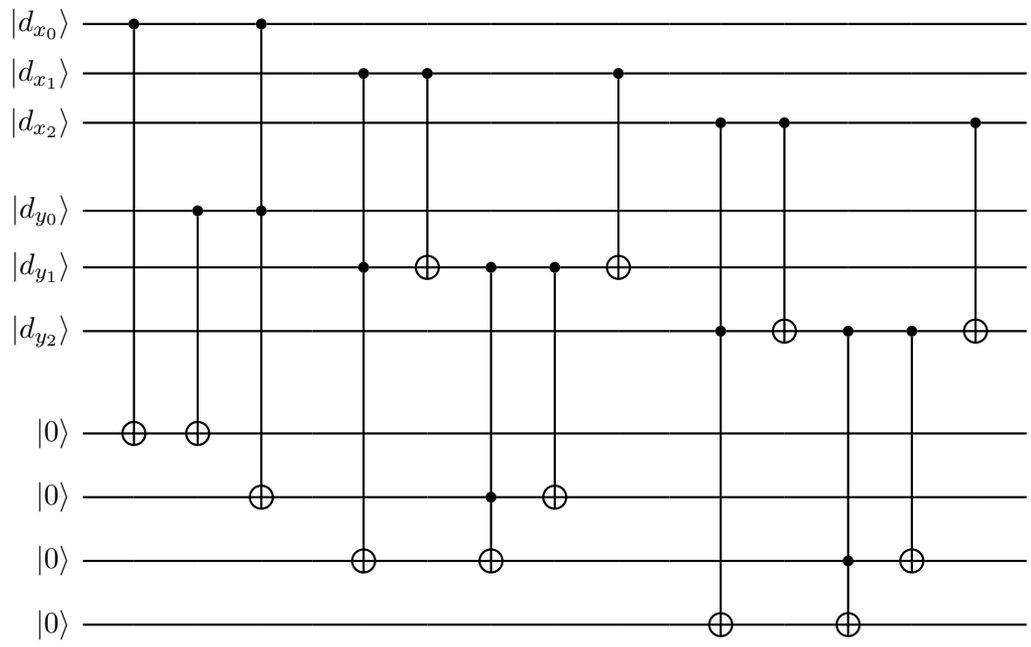


Abbildung A.1.: Abgebildet ist ein Addierer für zwei Drei-Qubit-Informationen. Dabei entsprechen die ersten sechs Zustände der Qubits den Datenregistereinträgen, die addiert werden sollen. Die letzten vier Qubits sind für die Darstellung des Ergebnisses zuständig und werden im Zustand $|0\rangle$ präpariert. Die ersten drei Gatter beschreiben den „Halbaddierer“ und die weiteren zehn Gatter zwei „Volladdierer“.

A.5. Kombination der Quantenschaltkreise des Addierers und Komparators

Die Aufgabe des Algorithmus aus Abschnitt 4.2 ist es, nach der Addition der Weglängen diese mit einem bestimmten Schwellenwert zu vergleichen. Falls das addierte Ergebnis dann kleiner ist als der Schwellenwert, wird dieses im Algorithmus aus Abschnitt 3.5.1 zum neuen Schwellenwert. Dafür muss also das Ergebnis der Addition mit dem Schwellenwert verglichen werden. Es muss somit zuerst die Addition durchgeführt werden. Dies geschieht mit dem in Abschnitt 5.1 beschriebenen Addierer. Das Ergebnis dieses Addierers wird nun für den Komparator aus Abschnitt 5.2 genutzt. Dazu wird die Summe der Zustände $|d_x\rangle$ und $|d_y\rangle$, die in dem folgenden Quantenschaltkreis $|d\rangle$ genannt wird, in die oberen Qubits des digitalen Komparators injiziert. Ein Beispiel dafür ist in Abb. A.2 dargestellt. Hierbei bestehen die Summanden $|d_x\rangle$ und $|d_y\rangle$ aus jeweils $n_x = n_y = 3$ Qubits. Dementsprechend benötigt die Summe $|d\rangle$ insgesamt $n = 4$ Qubits. Auch der Schwellenwert hat $n = 4$ Qubits. Somit ergibt sich in der Abb. A.2 ein Addierer für zwei Drei-Qubit-Informationen und ein digitaler Komparator von Vier-Qubit-Informationen.

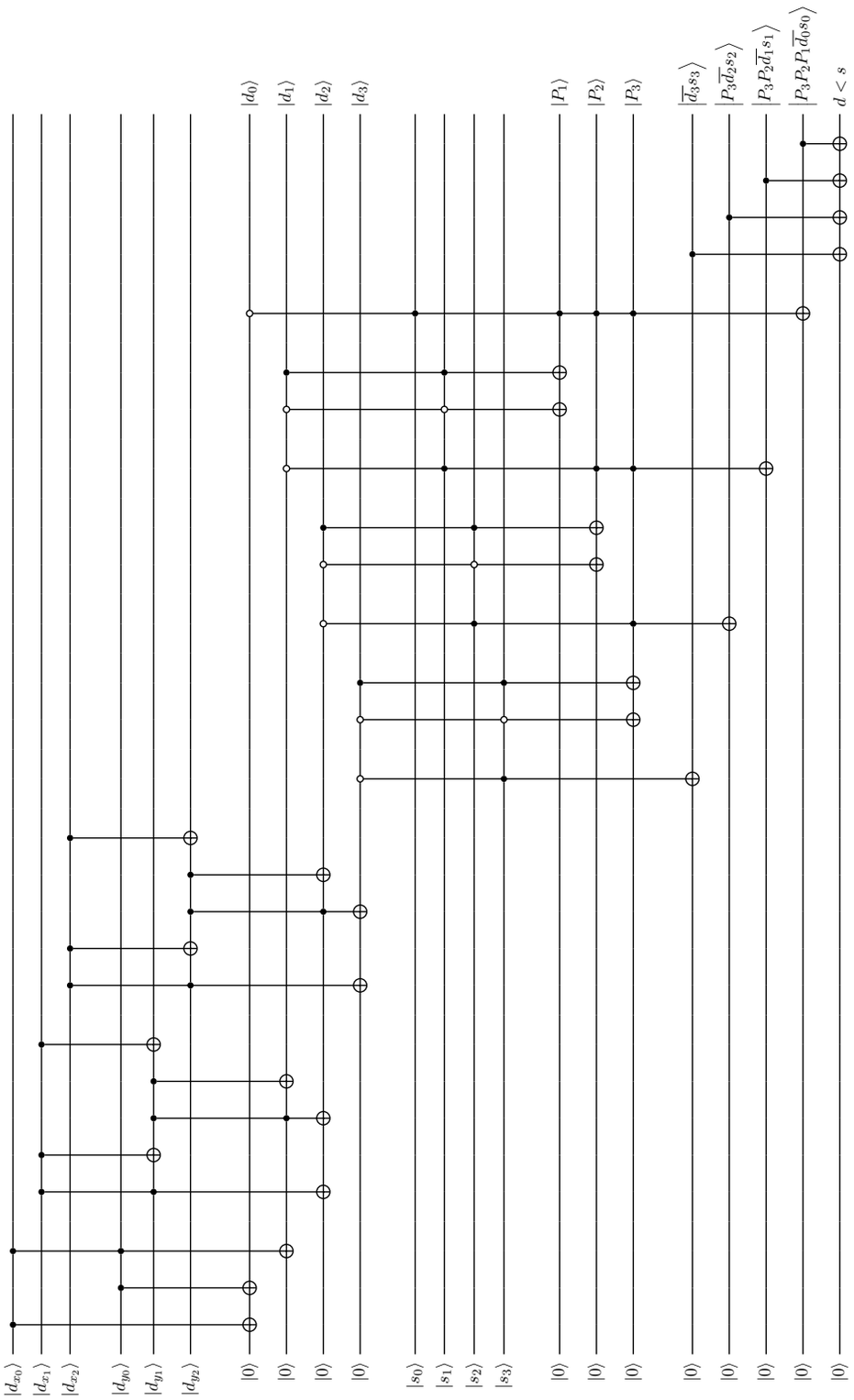


Abbildung A.2.: In der Abbildung ist ein Quantenschaltkreis für die Addition zweier Drei-Qubit-Informationen und im Anschluss daran ein Komperator zweier Vier-Qubit-Informationen zu sehen. Die ersten 13 Gatter beschreiben wie auch Abb. A.1 den Addierer. Die restlichen Gatter beschrieben den Komparator ähnlich zu Abb. 5.4. Dabei wird das Ergebnis der Addition im Zustand $|d\rangle$ für den Vergleich in die ersten vier Qubits des Komparators eingefügt.

Literatur

[1] M. A. Nielsen und I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[2] A. Ambainis et al. *Quantum Speedups for Exponential-Time Dynamic Programming Algorithms*. 2018. DOI: 10.48550/ARXIV.1807.05209.

[3] G. Pospiech. *Quantencomputer & Co: Grundideen und zentrale Begriffe der Quanteninformation verständlich erklärt*. Springer, 2021.

[4] M. Held und R. M. Karp. »A Dynamic Programming Approach to Sequencing Problems«. *Journal of the Society for Industrial and Applied Mathematics* 10.1 (1962), S. 196–210. ISSN: 03684245. URL: http://www.jstor.org/stable/2098806 (besucht am 27. 05. 2022).

[5] E. Farhi, J. Goldstone und S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. DOI: 10.48550/ARXIV.1411.4028.

[6] A. Montanaro. »Quantum speedup of branch-and-bound algorithms«. *Phys. Rev. Research* 2 (1 Jan. 2020), S. 013056. DOI: 10.1103/PhysRevResearch.2.013056.

[7] M. Hahsler und K. Hornik. »TSP—Infrastructure for the Traveling Salesperson Problem«. *Journal of Statistical Software* 23.2 (2007), S. 1–21. DOI: 10.18637/jss.v023.i02.

[8] B. Just. »Quantengatter auf einem QBit«. In: *Quantencomputing kompakt*. Springer, 2020, S. 73–87.

[9] W. Scherer. »Quantengatter und Schaltkreise für elementare Rechenoperationen«. In: *Mathematik der Quanteninformatik*. Springer, 2016, S. 111–188.

[10] B. M. Ellerhoff. *Mit Quanten rechnen: Quantencomputer für Neugierige*. 1. Aufl. essentials. Springer, 2020.

[11] T. A. Heilmann. »Reciprocal materiality and the body of code«. *Digital Culture & Society* 1.1 (2015), S. 39–52. DOI: 10.25969/mediarep/678.

[12] G. Pommranz. »Einige technische Begriffe: Byte, ASCII und Hexadezimal«. In: *Aufbaukurs MS-DOS: Das Microsoft-Handbuch zum professionellen Programmieren für den fortgeschrittenen Anwender*. Hrsg. von G. Pommranz. Wiesbaden: Vieweg+Teubner Verlag, 1988, S. 3–7. ISBN: 978-3-322-83691-5. DOI: 10.1007/978-3-322-83691-5_1.

[13] T. Moritz, H.-J. Steffens und P. Steffens. »Schaltkreise, Schaltwerke«. In: *Prüfungstrainer Informatik: 500 Fragen und Antworten für das Bachelor-Studium*. Heidelberg: Spektrum Akademischer Verlag, 2010, S. 229–247. ISBN: 978-3-8274-2567-6. DOI: 10.1007/978-3-8274-2567-6_13.

[14] F. Rapp. »Hintergrund und Bedeutung des P≠NP Problems« (2012).

[15] R. Bellman. »Dynamic Programming Treatment of the Travelling Salesman Problem«. *J. ACM* 9.1 (Jan. 1962), S. 61–63. ISSN: 0004-5411. DOI: 10.1145/321105.321111.

[16] L. K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. DOI: 10.48550/ARXIV.QUANT-PH/9605043.

[17] M. Boyer et al. »Tight Bounds on Quantum Searching«. *Fortschritte der Physik* 46.4-5 (Juni 1998), S. 493–505. DOI: 10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p.

[18] K. R. Ahmed Nasirand Rao. »Walsh-Hadamard Transform«. In: *Orthogonal Transforms for Digital Signal Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, S. 99–152. ISBN: 978-3-642-45450-9. DOI: 10.1007/978-3-642-45450-9_6.

[19] Qiskit. *Grover´s Algorithm*. URL: https://qiskit.org/textbook/ch-algorithms/grover.html.

[20] C. Durr und P. Hoyer. *A Quantum Algorithm for Finding the Minimum*. 1996. DOI: 10.48550/ARXIV.QUANT-PH/9607014.

[21] V. Giovannetti, S. Lloyd und L. Maccone. »Quantum Random Access Memory«. *Physical Review Letters* 100.16 (Apr. 2008). DOI: 10.1103/physrevlett.100.160501.

[22] Apple Inc. *Mac Pro*. 2019. URL: http://www.apple.com/de/mac-pro/.

[23] I. Wegener. *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Springer-Verlag, 2013.

[24] Quantum Inspire. *Code example: Quantum full adder*. 2022. URL: https://www.quantum-inspire.com/kbase/full-adder/.

[25] L. Ruiz-Perez und J. C. Garcia-Escartin. »Quantum arithmetic with the quantum Fourier transform«. *Quantum Information Processing* 16.6 (Apr. 2017). DOI: 10.1007/s11128-017-1603-1.

[26] G. Wang, S. Sheng und L. Ji. »New efficient design of digital comparator«. In: *2nd International Conference on ASIC*. IEEE. 1996, S. 263–266.

Die Bachelorarbeit wurde mit LaTeX erstellt. Ein Großteil des Layouts wurde von der Seite https://www.uni-giessen.de/fbz/fb07/fachgebiete/physik/institute/ipi/raumfahrtphysik/Ionentriebwerke/Richtlinien/LaTeX-Vorlage/view der Universität Giessen übernommen.

Abbildungsverzeichnis

Tabellenverzeichnis