2013•2014
FACULTEIT WETENSCHAPPEN
master in de informatica

Masterproef
Quantum Computing

Promotor :
Prof. dr. Jan VAN DEN BUSSCHE

Jan Anthonissen

Proefschrift ingediend tot het behalen van de graad van master in de informatica

2013•2014
FACULTEIT WETENSCHAPPEN
master in de informatica

Masterproef

Quantum Computing

Promotor :
Prof. dr. Jan VAN DEN BUSSCHE

Jan Anthonissen
Proefschrift ingediend tot het behalen van de graad van master in de informatica

universiteit hasselt | Maastricht University

Samenvatting

Hoewel de nood aan meer rekenkracht steeds blijft toenemen, wordt het tegenwoordig als maar moeilijker om aan deze vraag te voldoen. Zelfs met de indrukwekkend snelle evolutie van de computerchips wordt dit haast een onmogelijke taak. We merken dat er problemen bestaan die zoveel rekenkracht vragen in vorm van tijd en geheugenruimte dat we er geen efficiënte oplossing voor kunnen vinden. Bovendien stelt zich ook nog het probleem dat we bijna de fysieke grens hebben bereikt van het aantal transistoren dat op een chip past. Deze oorzaken hebben er voor gezorgd dat we uitkijken naar andere manieren om over computers na te denken dan degene die we momenteel gebruiken zoals DNA Computing, Parallel Computing, Quantum Computing,.... In deze masterproef gaan we dieper in op deze laatste. We proberen eerst met behulp van complexiteit duidelijk te maken dat sommige problemen moeilijk oplosbaar blijven onafhankelijk van het aantal resources dat we gebruiken. Hierna proberen we duidelijk te maken hoe een Quantumcomputer werkt en gebruikt kan worden om deze problemen op te lossen. Dit doen we door enkele algoritmes te bespreken en uit te leggen. Als laatste kijken we kort naar realisaties die al verwezenlijkt zijn met het bouwen van Quantumcomputers, waaronder de controversiële D-Wave computers.

Voorwoord

Het onderwerp voor mijn masterproef vindt vooral zijn oorsprong in mijn eigen interesse voor Quantumcomputers. Reeds geruime tijd zocht ik in mijn vrije tijd artikels en video's op over dit onderwerp. Dit gebeurde steeds vaker omdat in die periode Quantumcomputers enkele malen het nieuws hadden gehaald na de aankoop van Google bij D-Wave. Ik ben mij beginnen te verdiepen in dit onderwerp. Toen ik zag dat dit onderwerp ook beschikbaar was om als masterproef te doen heb ik niet langer getwijfeld. Tijdens het maken van mijn masterproef heb ik veel bijgeleerd over dit onderwerp en hoop ik dan ook deze interessante maar ingewikkelde Quantumwereld te kunnen delen met anderen.

Mijn masterproef heb ik voor het grootste deel gebaseerd op twee boeken, aangeraden door mijn promotor, namelijk:

- Quantum Computing since Democritus - Scott Aaronson [1];
- Quantum Computation and Quantum Information - Michael A. Nielsen en Isaac L. Chuang [18].

Verder werd ze vaak aangevuld met informatie die ik opzocht op het internet, om de meer ingewikkelde delen te begrijpen. Uit deze boeken heb ik niet enkel de informatie gehaald om mijn masterproef te kunnen maken maar ook de structuur ontleende ik er grotendeels aan. Zo kan men merken dat deze boeken, net als de meeste werken over Quantumcomputers, beginnen met een stuk over complexiteit, om zo een beeld te kunnen scheppen van de moeilijkheden die we willen oplossen met Quantumcomputer. De informatie voor de hoofdstukken over de Fouriertransformatie en de Quantumzoekalgoritmes heb ik vooral uit het boek van Nielsen en Chuang gehaald.
Graag wil ik langs deze weg mijn promotor Jan Van den Bussche bedanken, voor de goede raad en alle tijd die hij in deze masterproef heeft gestoken. Ook mijn mama wil ik bedanken om mij steeds te steunen.

Inhoudsopgave

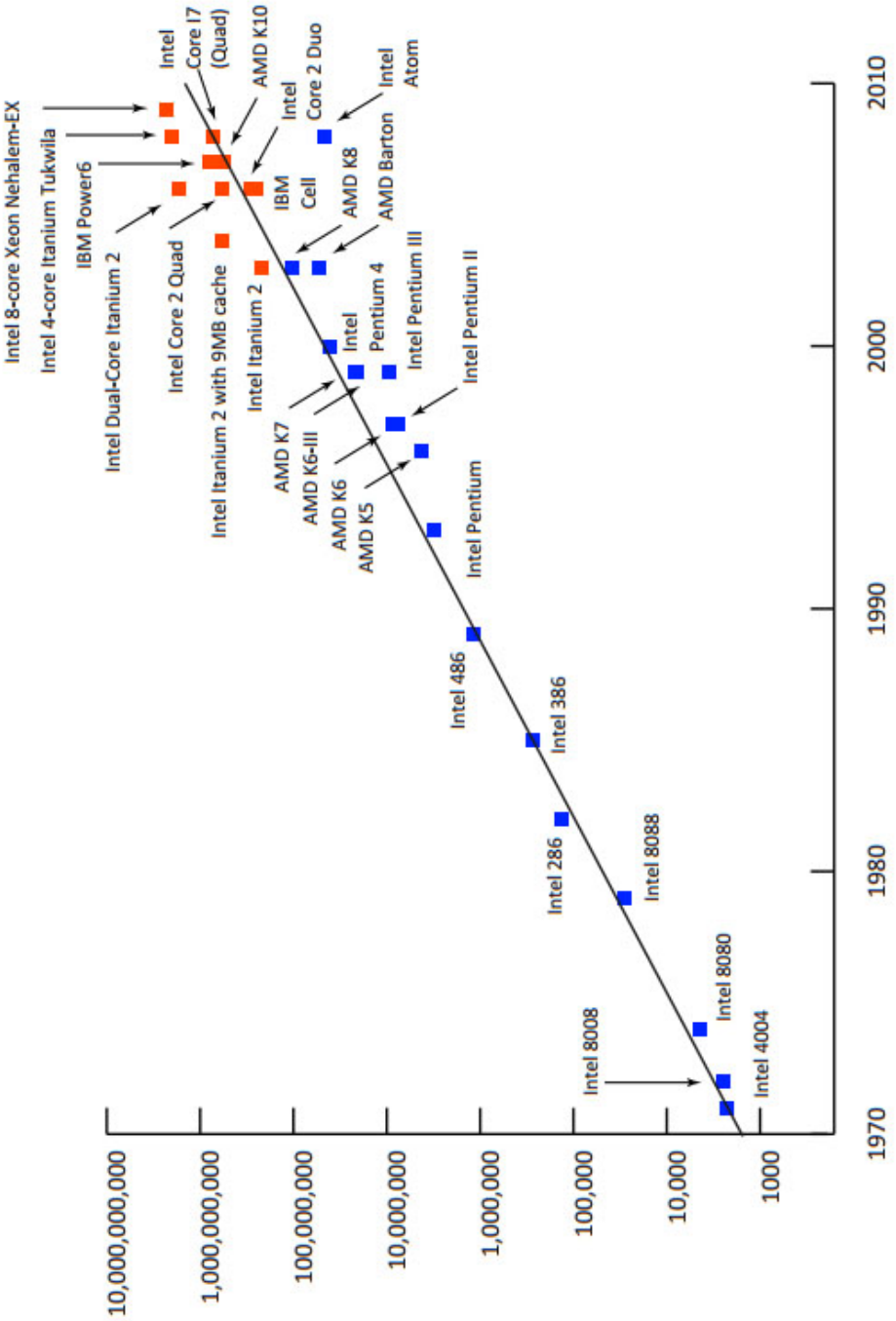
1 Inleiding

In deze masterproef proberen we een antwoord te geven op een aantal vragen zoals: Wat is Quantum Computing? Waarom is Quantum Computing nodig? Wat zijn de voordelen? Zijn er nadelen?
Wat is Quantum Computing? Quantum Computing is de studie van informatica waarin men gebruik maakt van Quantummechanische systemen, dit in tegenstelling tot de klassieke informatica waar men gebruik maakt van computers zoals we ze nu kennen. Dit is echter een zeer simpel antwoord. Als Quantum Computing zo simpel voor te stellen is, waarom heeft het dan zolang geduurd voordat we er aan gedacht hebben? Waarom is het zo moeilijk om een Quantumcomputer te bouwen? Ook op deze vragen gaan we proberen een antwoord te geven. Zoals het woord doet vermoeden is Quantum Computing een onderwerp waar twee werelden elkaar ontmoeten. Enerzijds hebben we de Quantumfysica of Quantummechanica, anderzijds de wereld van de informatica. We zullen proberen kort en bondig een inleiding te geven in deze twee werelden, voordat we aan het eigenlijke onderwerp kunnen beginnen.

1.1 Waarom

Sinds het uitvinden van de transistor (1947) en daarmee ook het begin van de computerchip zoals we ze nu kennen is onze vraag naar rekenkracht steeds groter geworden. Dit in tegenstelling tot de voorspellingen van de ontwerpers van de eerste computers zoals Howard Aiken, die voorspelde dat de Verenigde Staten niet meer dan 6 computers zouden nodig hebben. Of Thomas Watson die in 1943 schatte dat de wereldmarkt voor computers niet meer dan 5 zou zijn. Deze man was op dat moment voorzitter bij IBM. Uitspraken zoals deze klinken ons zeer raar in de oren, in een tijd waar computers en internet overal gebruikt worden om enorme grote hoeveelheden data te verwerken. Na de eerste chips met transistoren volgt een enorme en snelle evolutie van computerchips. Deze groei werd door Gordon Moore vastgelegd in wat wij nu Moore's Law noemen (1965). Zie figuur 1. In deze wet wordt gezegd dat het aantal transistors op een chip elke 2 jaar verdubbelt, wat neerkomt op het feit dat de rekenkracht elke twee jaar toeneemt.

Voorlopig lijkt Moore's Law nog te kloppen, nadat ze een aantal keren is aangepast, maar het aantal transistoren op een chip begint zo groot te worden dat het einde van Moore's Law in zicht komt. We beginnen een fysieke grens te bereiken van het aantal transitoren dat we op een chip krijgen. We moeten transistoren zo klein maken dat ze bijna op atomair niveau



Figuur 1: Moore's Law (Bron: `http://rapidconsultingusa.com/wp-content/uploads/2013/04/Atego-Java-Figure-1.jpg`)

zijn. Op dit niveau beginnen Quantumeffecten in te werken op de transistors waardoor ze niet meer betrouwbaar zijn. Maar we hebben nog steeds meer rekenkracht nodig om bepaalde problemen op te lossen.
Het bereiken van deze fysieke grens zorgt er dus voor dat we niet verder kunnen op de klassieke manier waar we gebruik maken van de klassieke fysica. Maar wat als we het optreden van Quantumeffecten niet zien als een probleem maar als de oplossing voor onze steeds toenemende vraag voor rekenkracht? Wat als we gebruik maken van deze Quantummechanica om berekeningen uit te voeren in een computer? Dit is het idee achter de theorie van Quantum Computing.
Waarvoor is een Quantumcomputer dan nuttig? Het blijkt dat de meeste interessante problemen die we tegenkomen in de informatica praktisch niet op te lossen zijn op een gewone computer. Niet omdat deze problemen zo moeilijk zijn dat we er geen oplossing voor kunnen vinden, maar eerder omdat de resources die nodig zijn om de oplossing te laten werken zo enorm hoog zijn dat ze onrealistisch zijn voor mensen. Vaak gaat het hier ook over enorme hoeveelheden tijd. Met het ontwikkelen van Quantumcomputers hebben we uitzicht op nieuwe mogelijke algoritmes die ons toelaten problemen op te lossen zoals we dat niet kunnen op een gewone computer.
We moeten echter behoedzaam zijn met uitspraken te doen over de snelheidswinst die we kunnen bereiken met een Quantumcomputer, omdat we nog maar in het begin staan van de ontwikkeling en alles nog zeer theoretisch is. Op dit moment is men er nog maar in geslaagd een Quantumcomputer te bouwen met enkele qubits (Quantum bits). Met deze enkele qubits kan men nog niet echt problemen oplossen maar toont men eerder aan dat Quantum Computing niet meer alleen een pure theorie is.
Quantumcomputers worden echter in de media vaak voorgesteld als een supercomputer die alle mogelijke oplossingen voor een probleem tegelijk kan berekenen en dan de juiste oplossing uit kan kiezen. Dit is gebaseerd op het belangrijkste principe Quantum Parallellisme maar gaat wat kort door de bocht zoals we later zullen zien [12].

Het idee om de kracht van Quantummechanica te gebruiken voor computatie is nog maar een relatief jong idee, zeker in vergelijking met de klassieke computers waarvan het begin al vroeg in de vorige eeuw ontstaan is. Maar net zoals in de meeste andere takken van de wetenschap gebeuren ontdekkingen en uitvindingen steeds sneller en volgen zich steeds sneller op. Toch zal het nog een tijdje duren voordat we een eerste echt praktische Quantumcomputer zullen kunnen gebruiken om echte problemen op te lossen. Toch lijkt het ons zeer interessant om nu al de volgende stap in de evolutie van de computer te bestuderen.

2 Informatica en complexiteit

Zoals eerder aangehaald bestaat Quantum Computing uit twee deelgebieden, die elkaar ontmoeten: de informatica en de Quantummechanica. In dit deel gaan we even kort overlopen wat we hier verstaan onder informatica en verklaren we enkele begrippen, die ons zullen helpen om Quantum Computing beter te verstaan.

2.1 Informatica

Wat is informatica? Op het internet gaat een quote rond die onterecht wordt toegekend aan Dijkstra. Deze uitspraak geeft ons een beter beeld van wat wij onder informatica verstaan.

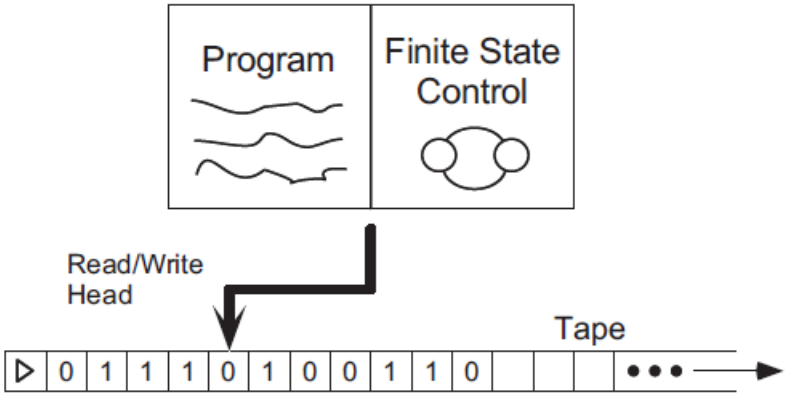
> "Computer Science is no more about computers than astronomy is about telescopes."

Hiermee willen we aanduiden dat informatica niet de wetenschap is die computers bestudeert, maar eerder de wetenschap waarvan één van de belangrijkste gereedschappen de computer is. Waarover gaat informatica dan wel? Vooral over algoritmes.
Een algoritme is een lijst van instructies, die eindig is. Deze lijst van instructies kan gebruikt worden om van een begintoestand tot een beoogde eindtoestand te komen en zo problemen op te lossen. Omdat complexe problemen vaak ingewikkelde of lange algoritmes vragen maken we vaak gebruik van computers om deze algoritmes uit te voeren. Binnen de informatica wordt vaak gebruik gemaakt van een theoretische en geïdealiseerde machine, de Turingmachine, hier komen we later nog op terug.
Er zijn verschillende redenen waarom we geïnteresseerd zijn in het bestuderen van algoritmes. Als eerste willen we voor elk specifiek probleem een goed uitgeschreven oplossing vinden. Bovendien willen we hiervan weten hoeveel resources (tijd, opslagruimte) deze gebruikt. Daarnaast willen we ook graag weten of onze gevonden oplossing de best mogelijke is en zo ook een limiet definiëren van wat mogelijk is met een algoritme. De ideale situatie zou zijn, wanneer de limiet die we vinden voor een probleem dezelfde is als de resources die onze oplossing gebruikt. Hier knelt echter het schoentje. Er zijn problemen bekend waarvoor we tot nog toe geen efficiënte oplossing hebben gevonden in de klassieke informatica, zoals het factoring probleem. Maar er is wel een efficiënt algoritme bekend voor factoring in Quantum Computing. Dit doet ons vermoeden dat er nog een aantal andere problemen kunnen zijn, die voorlopig niet op te lossen lijken maar die op een Quantumcomputer wel een efficiënt algoritme hebben.

Wat betekent het om een algoritme te vinden voor een probleem? Eén van de eerste algoritmes die iedereen leert is het algoritme om twee getallen bij elkaar te tellen. Welke deze getallen zijn en hoe groot of klein ze zijn maakt niet veel uit. Dit is wat we verstaan onder algoritme, een precieze formule die we kunnen gebruiken om een probleem op te lossen, onafhankelijk van de input. Zolang de input maar uit een zekere gedefinieerde taal komt.
Algoritmes zijn in tegenstelling tot computers en de informatica niet een recente ontdekking. Denk maar aan het algoritme van Euclides om de grootste gemene deler te vinden of de zeef van Eratosthenes om priemgetallen te vinden. Deze algoritmes zijn al in de oudheid ontdekt lang voordat er sprake was van computers.
Pas veel later, in 1930 spreken we over de informatica zoals we ze nu kennen, voornamelijk voorgesteld door Alonzo Church en Alan Turing. Deze informatica is tot stand gekomen nadat Turing en Church werkten aan een probleem, gesteld door David Hilbert. Het Entscheidungsproblem, waarin Hilbert zich afvraagt of er een algoritme bestaat waarmee het mogelijk is volgend probleem op te lossen. We geven als input een mathematisch probleem en verwachten als output een antwoord op de vraag of dit probleem een oplossing heeft. Hilbert dacht dat het antwoord op zijn vraag positief zou zijn. Maar Church en Turing bewezen later dat dit niet het geval is. Om dit te kunnen bewijzen was het nodig voor Turing en Church om een zeer precieze en wiskundige definitie te maken van wat we verstaan onder een algoritme. Door deze stap kunnen we hun beschouwen als de grondleggers van wat we nu de theoretische informatica noemen.

2.1.1 Turingmachines

Om zijn definitie van een algoritme aanschouwelijk te kunnen maken bedacht Turing een mechanisme dat eender welk algoritme zou kunnen uitvoeren. We noemen dit nu Turingmachines. Een Turingmachine kan men vergelijken met een soort computer die uit 4 onderdelen bestaat. Als eerste hebben we een programma, een lijst met instructies die ons van het begin tot aan de oplossing leidt. Vervolgens is er een eindige toestandautomaat. Men kan dit vergelijken met een microprocessor. Deze houdt bij in welke toestand de Turingmachine op dit moment is en wat de volgende actie is. Als derde is er een band die aan één kant onbegrensd kan groeien (meestal naar rechts). Deze band is onderverdeeld in vakjes. In elk vakje kan een symbool komen uit de taal waarvoor het programma werkt. De band van een Turingmachine is onbegrensd, wat betekent dat wanneer we meer plaats nodig hebben we altijd meer vakjes kunnen toevoegen aan het einde van de band. Als vierde is er de lees- en schrijfkop, deze bevindt zich boven een vakje op de band en kan het symbool wat hier staat lezen of er zelf een schrijven. De lees- en schrijfkop kan blijven stilstaan of naar links of rechts bewegen, telkens één



Figuur 2: Turingmachine (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

vakje per toestand. Op figuur 2 geven we een schematische voorstelling van een Turingmachine.

De eindige toestandautomaat is een eindige verzameling van toestanden waarin de Turingmachine zich kan bevinden. Deze wordt meestal weergegeven door $q_1, q_2, q_3, ..., q_m$. De grootte van m heeft geen directe verhouding met de kracht van de Turingmachine. Naast deze toestanden zijn er nog twee speciale toestanden q_s en q_h die de begin en eindtoestand weergeven. Een Turingmachine zal altijd starten in de begintoestand en zal stoppen wanneer de eindtoestand bereikt is. Een Turingmachine zal niet altijd de eindtoestand bereiken. Dit gebeurt alleen wanneer de berekening voltooid is.
De tape van een Turingmachine is een band die in één richting onbeperkt is en onderverdeeld is in oneindig veel vakjes. We nummeren deze vakje $1, 2, 3, ...$. Elk vakje bevat een symbool uit een eindig alfabet, dat een eindig aantal verschillende symbolen bevat. Een voorbeeld van een alfabet dat vaak gebruikt wordt om de werking van een Turingmachine mee uit te leggen is de taal die bestaat uit $1, b, 0$. De b staat hier voor blank staat, een leeg vakje. Vaak voegt men aan dit alfabet een speciaal symbool toe waarmee men het linkereinde van de tape markeert, zodat de lees- en schrijfkop niet van de tape af kan.
De lees- en schrijfkop staat steeds boven één vakje en kan het symbool dat hierin staat lezen of kan er zelf een symbool in schrijven. Als er reeds een symbool aanwezig was wordt dit vervangen. De lees- en schrijfkop zal in de begintoestand boven het meest linkse vakje hangen. Na de start zal de Turingmachine stap voor stap door het programma lopen totdat de halttoe-

stand is bereikt, waarna de output van het programma te lezen zal zijn op de tape. Tijdens het uitvoeren van het programma verandert de positie van schrijfkop, afhankelijk van het symbool dat wordt gelezen en de toestand waarin de Turingmachine zich bevindt.
Een programma voor een Turingmachine is een eindige lijst van instructies, die meestal de vorm hebben van (q, x, q', x', s). Waarbij q en q' een toestand van de eindige toestandautomaatverzameling zijn en x en x' symbolen uit de taal die kan voorkomen op de tape; s geeft de richting aan waarin de lees- en schrijfkop moet bewegen. We doen dit met behulp van -1 , 0 of $+1$ om links, stilstaan of rechts aan te duiden. De Turingmachine zal steeds in het programma een regel proberen te vinden waar q zijn huidige toestand is en x gelijk is aan het symbool dat nu wordt uitgelezen. Indien er zo geen regel bestaat in het programma zal de Turingmachine in de halting toestand gaan en zal de berekening stoppen. Als er wel zo een regel wordt gevonden zal deze uitgevoerd worden, hier bedoelen we mee dat de toestand verandert van q naar q', het symbool x wordt overschreven door x' , en de lees/schrijfkop wordt verplaatst afhankelijk van de waarde van s. Hieronder volgt een voorbeeld van een simpel programma voor een Turingmachine.

1. $\langle q_s, \triangleright, q_1, \triangleright, +1\rangle$
2. $\langle q_1, 0, q_1, b, +1\rangle$
3. $\langle q_1, 1, q_1, b, +1\rangle$
4. $\langle q_1, b, q_2, b, -1\rangle$
5. $\langle q_2, b, q_2, b, -1\rangle$
6. $\langle q_2, \triangleright, q_3, \triangleright, +1\rangle$
7. $\langle q_3, b, q_h, 1, 0\rangle$

Dit programma verwacht als input op de tape een binair getal, voorafgegaan door een pijltje om het begin van de tape aan te duiden en gevolgd door het lege symbool. Het programma zal dan de tape lezen en elke 0 of 1 overschrijven met een blanco (b), wanneer er geen 0 of 1 meer gevonden wordt keert de lees/schrijfkop terug naar de beginpositie en schrijft in het eerste vakje een 1. Dit stelt dus de functie $f(x) = 1$ voor. De input heeft geen invloed op de output die altijd 1 zal zijn. Dit simpele voorbeeld geeft weer hoe een Turingmachine werkt. Een Turingmachine kan natuurlijk veel meer berekenen dan zo een eenvoudige functie zoals $f(x) = 1$. Wat kunnen we dan wel allemaal berekenen met een Turingmachine? Het blijkt dat een Turingmachine even krachtig is in berekenbaarheid als een moderne computer. Men kan dus geen algoritme vinden dat zou werken op een PC en niet op een Turingmachine.

De Turingmachine definieert dus zeer goed wat we verstaan onder een algoritme, dit wordt vastgelegd in de Church - Turing thesis [27]. Deze kan worden samengevat als: “Elke mogelijke berekening kan door een algoritme op een Turingmachine uitgevoerd worden, mits er genoeg geheugen en tijd beschikbaar is.” We nemen aan dat het begrip algoritme voldoet aan een aantal voorwaarden. Het algoritme is eindig en duidelijk geformuleerd in verschillende stappen. Het algoritme geeft een antwoord in een eindig aantal stappen. We zouden het algoritme ook met pen en papier kunnen uitvoeren in theorie. Er is geen verdere kennis nodig om de stappen uit te voeren. Maar wat bedoelen we met elke mogelijke berekening, hoe definiëren we of een algoritme duidelijk gedefinieerd is. Door al deze onduidelijkheden is het moeilijk deze stelling te bewijzen of te ontkrachten. Daarom is deze thesis meer een hypothese, aangezien ze nog niet bewezen is en waarschijnlijk ook nooit echt bewezen gaat kunnen worden. Er is echter ook geen bewijs om het tegendeel aan te duiden. Moest dit ooit wel het geval zijn, dat er een proces bestaat dat een berekening kan maken die een Turingmachine niet kan maken, dan zouden we iets gevonden hebben dat ons kan helpen om meer en snellere berekeningen te doen.
Deze thesis stelt dus dat elk algoritme dat uitgevoerd kan worden op een klassieke computer ook uitgevoerd kan worden op een Turingmachine. Dit zelfde geldt voor Quantumcomputers. Met Quantumcomputers kunnen we tot zover we weten niet meer problemen oplossen dan mogelijk is met Turingmachines. Waarom zijn we dan toch zo geïnteresseerd om een Quantumcomputer te maken ook al kunnen we er niet meer mee oplossen? Het verschil zit hem in de efficiëntie waarmee een Quantumcomputer deze oplossingen bereikt. Het laatste deel van de Church-Turing thesis gaat als volgt “...mits er genoeg geheugen en tijd is”. In de realiteit zijn tijd en geheugen echter beperkt en beschouwen we sommige problemen als niet oplosbaar met behulp van Turingmachines of klassieke computers. Net voor een aantal van deze problemen vinden we met Quantumcomputers wel een efficiënte oplossing, die bovendien in een aanvaardbare tijd en hoeveelheid geheugen berekend kan worden.

Zoals hierboven beschreven kan worden kunnen programma’s om zeer simpele functies te berekenen op een Turingmachine al snel omslachtig worden. Daarom maakt men vaak gebruik van een variant van de Turingmachine. Dit kan omdat men met behulp van de Chruch - Turing thesis kan bewijzen dat deze machines even krachtig zijn. Enkele voorbeelden hiervan zijn Turingmachines waarvan de tape aan beide zijdes onbeperkt is. Turingmachines met meerdere tapes, Turingmachines die het veranderen van toestand laten afhangen van een kansberekening. Het is zeer simpel om te bewijzen dat deze machines dezelfde kracht hebben als een gewone Turingmachine. We bewijzen hier nu op een eenvoudige manier dat een Turingmachine met meerdere tapes makkelijk te simuleren is door een gewone Turingmachine.

Dit is makkelijk in te zien omdat we de meerdere tapes nabootsen door ze allemaal op één tape achter elkaar te plaatsen en tussen de inhoud van elke tape een speciaal symbool toevoegen dat aangeeft dat hier een nieuwe tape begint. We moeten dan enkel het programma aanpassen zodat het hier rekening mee houdt. Een zeer belangrijke aanpassing die we moeten doen, zijn de stappen die we moeten toevoegen om een symbool op een tape bij te plaatsen, die niet de laatste tape is. We moeten dan een vakje bijnemen op de tape maar kunnen niet zomaar de tape splitsen en terug plakken. We moeten dus vanaf de plaats waar we willen invoegen telkens een vakje opschuiven. Dit gebeurt door elk vakje naar rechts op te schuiven. Dit doen we door de inhoud te kopiëren naar het vakje ernaast. Het toevoegen van een symbool ergens op de tape kan dus veel stappen vragen. Toch is het duidelijk dat men met deze twee machines nu exact dezelfde berekeningen kan doen, ook al lijkt de Turingmachine met meerdere tapes krachtiger. Men kan voor alle varianten van de Turingmachine een gelijkaardig bewijs vinden waarbij men steeds de variant nabootst op de gewone Turingmachine.
De reden voor het bedenken van de Turingmachine was het oplossen van het Entscheidungsprobleem, zoals we in het begin van dit deel hebben besproken. Het antwoord op dit probleem is volgens Church en Turing negatief. Om dit te bewijzen hebben ze gebruik gemaakt van het halting probleem.

Halting probleem

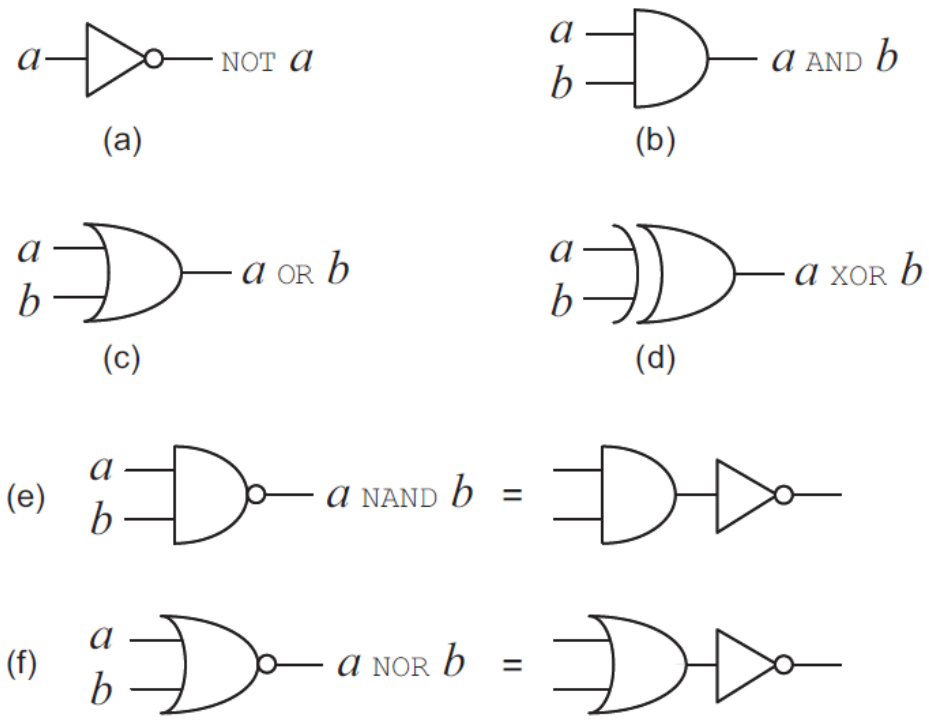
Het halting probleem kan heel simpel worden voorgesteld. Kunnen we een algoritme vinden dat beslist of een Turingmachine zal stoppen op input x? Turing bewees dat het onmogelijk is zo een algoritme te vinden. Hiervoor maken we gebruik van de halting functie, deze definiëren we als volgt. $halt(A, x)$ zal als resultaat 1 geven wanneer het algoritme A met input x kan eindigen, hier bedoelen we mee dat de Turingmachine die algoritme A kan berekenen ooit de toestand q_h zal bereiken als ze x als input krijgt. De functie $halt(A, x)$ zal als resultaat 0 geven wanneer dit niet het geval is. We nemen aan dat deze functie bestaat en definiëren een tweede functie die we paradox noemen. De functie paradox werkt als volgt, $paradox(A) = 1$ als $halt(A, A) = 0$ en $paradox(A)$ zal oneindig blijven lopen als $halt(A, A) = 1$. We geven nu de functie paradox aan zichzelf mee als input, de vraag is nu of paradox ooit zal eindigen als de input paradox is. Dus geeft $paradox(paradox)$ als oplossing 1? Indien dit mogelijk zou zijn moet $halt(paradox(paradox))$ als antwoord 0 geven maar dit betekent dat $paradox(paradox)$ niet zou eindigen wat een tegenstelling is. Als dit niet mogelijk is zou $halt(paradox(paradox))$ als oplossing 1 moeten geven maar dit betekent dat $paradox(paradox)$ wel kan eindigen wat ook een tegenstelling is. Hieruit leiden we af dat de originele halt functie niet echt kan bestaan en dat het halting probleem onbeslisbaar is, als gevolg kunnen we

ook besluiten dat het Entscheidungsprobleem ook geen oplossing heeft.
Het halting probleem is één van de eerste problemen waarvan we weten dat het onbeslisbaar is. Dit heeft grote gevolgen gehad voor de manier waarop we naar problemen kijken. We weten ondertussen al van tal van problemen dat ze onbeslisbaar zijn. Dit zorgt er tevens voor dat we de in informatica vanaf dat moment een onderscheid maken tussen problemen die makkelijk op te lossen zijn en problemen die moeilijk op te lossen zijn. Waarvan de extremen de problemen zijn die zo moeilijk op te lossen zijn dat ze onbeslisbaar worden.

2.1.2 Circuits

We hebben hierboven met behulp van de Turingmachine een model gegeven om na te kunnen denken over berekeningen en de limieten die we tegen komen wanneer we problemen proberen op te lossen aan de hand van algoritmen. Maar de Turingmachines zijn onbeperkt in grootte en tijd (Church Turing thesis). Echte computers zijn echter niet onbeperkt, daarom voeren we nog een tweede model in wat iets realistischer is, het circuit model. Zoals al eerder gezegd, voegt dit andere model geen rekenkracht toe en is het equivalent aan een Turingmachine. We kunnen er dus niet meer problemen mee oplossen. Dit model komt later wel beter van pas bij het onderzoeken hoe Quantumcomputers werken wanneer we gaan werken met Quantum circuits. In het circuit model maken we gebruik van draden en poorten die ieder hun eigen functies hebben. De poorten voeren hun acties uit op bits, de bits kunnen 2 waardes hebben ofwel 0 voor uit of 1 voor aan. De draden hebben als functie de verschillende poorten te verbinden met elkaar en geven weer in welke volgorde we de poorten gaan gebruiken. De draden vervoeren als het ware de bits van en naar de poorten. De poorten krijgen een of meerdere bits binnen en voeren hier dan een actie op uit om zo één of meerdere bits als output te geven. We kunnen een poort ook als een functie schrijven die weergeeft wat er met de bits gebeurt. Deze functies hebben dan de vorm van $f : \{0,1\}^k \to \{0,1\}^l$ waarbij k het aantal input bits weergeeft en l het aantal output bits. Verder spreken we ook nog af dat we geen loops toelaten in een circuit.

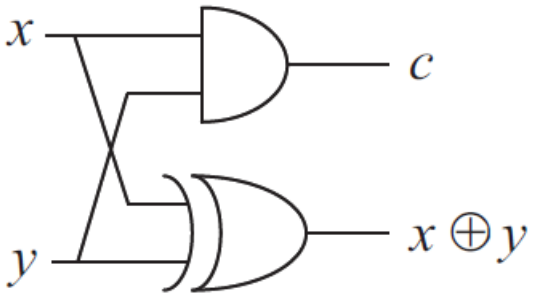
In de afbeelding hieronder geven we een aantal poorten die vaak gebruikt worden in het circuit model. Als eerste en eenvoudigste is er de NOT-poort. Deze poort krijgt 1 bit als input en geeft 1 bit als output. De output van een NOT-poort is de negatie van de input bit, dus 0 wordt 1 en 1 wordt 0. Als tweede hebben we de AND-poort. Deze krijgt 2 input bits en geeft als output 1 bit, die 1 is als de twee input bits 1 zijn en in elk ander geval 0 als output. De OR-poort geeft als output 1 als minstens een van de twee input bits 1 is. De XOR-poort geeft als output 1 als de twee input bits verschillend



Figuur 3: Logical Gates (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

zijn en anders is de output 0. De NOR- en NAND-poort werken hetzelfde als de OR- en AND-poort gevolgd door een NOT-poort. In figuur 3 geven we een schematische weergave van de meest gebruikte poorten.

Met deze verschillende poorten is het mogelijk om een circuit te bouwen door verschillende poorten aan elkaar te schakelen met draden. We kunnen met circuits eender welke functie berekenen. Er is echter één voorwaarde. We moeten weten hoeveel input en output poorten we nodig hebben. Met andere woorden wanneer we de te berekenen functie voorstellen als $f : \{0,1\}^k \to \{0,1\}^l$, dan willen we de waarde van k en l weten. We hebben voor elke waarde van k en l een ander circuit nodig. Het is echter mogelijk om een Turingmachine te maken, die als input k en l meekrijgt en dan het circuit kan bouwen voor deze grootte. Als we voor alle groottes zo een circuit kunnen laten bouwen, spreken we van een uniform circuit family. Indien we deze eis van uniformiteit niet stellen, kunnen we met behulp van circuits bijvoorbeeld het halting probleem oplossen voor een bepaalde grootte. Dit doen we door een circuit te maken dat voor een bepaalde input de output geeft die we willen. We hardcoden dus gewoon het resultaat dat



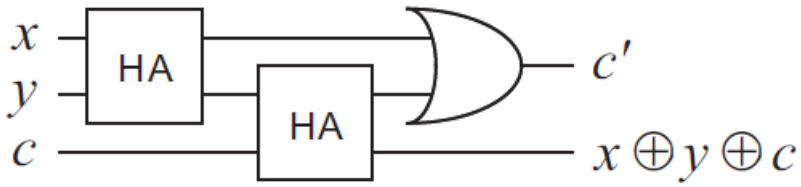
Figuur 4: Half Adder (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

we willen. Bekende voorbeelden van zo een uniform circuit family zijn de half adder en de full adder. Deze circuits worden gebruikt om twee getallen bij elkaar op te tellen en kunnen dus gebruikt worden om het algoritme van de optelling te berekenen.
De half adder (figuur 4) bestaat uit een AND-poort en een XOR-poort en krijgt als input twee bits en geeft als output twee bits. Een output bit is de modulo 2 som van de input en de tweede output die we de carry noemen is 1 wanneer de input bits beide 1 zijn. Anders is de carry bit 0.

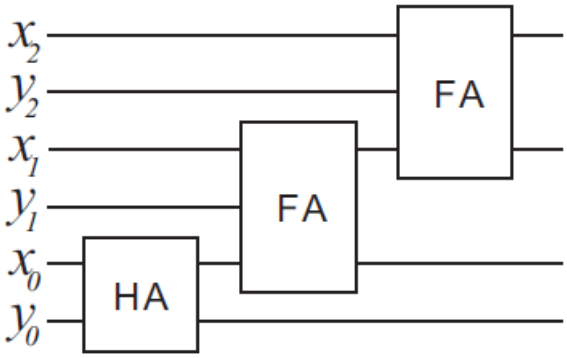
De full adder (figuur 5) is opgebouwd uit twee half adders die we achter elkaar plaatsen. De full adder krijgt als input drie bits, twee van de input bits zijn de data die we willen optellen. De derde bit die we als input krijgen is een carry bit van een eerdere berekening. De output van een full adder is de modulo twee som van de drie input bits en als tweede output bit geven we weer een carry bit die 1 zal zijn als twee of meer van de input bits 1 zijn. Deze full adders kunnen we nu achter elkaar schakelen om twee getallen bij elkaar op te tellen. Het aantal full adders dat we achter elkaar moeten plaatsen hangt af van het aantal bits waaruit het getal bestaat. In figuur 6 geven we het voorbeeld voor twee getallen van 3 bits.

2.2 Complexiteit

In het vorige deel hebben we twee modellen beschreven, waardoor we over computers en berekenbaarheid kunnen nadenken. Deze twee modellen lijken zeer verschillend, maar zijn toch equivalent in rekenkracht. Toch zijn ze handig om op verschillende manieren over algoritmes en computers na te denken. Het circuit model kunnen we gebruiken om meer realistische computers voor te stellen die dichter aansluiten bij deze die wij kennen. De



Figuur 5: Full Adder (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)



Figuur 6: Optelling (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

Turingmachine met zijn onbeperkt geheugen en onbeperkt veel tijd om in te rekenen, gebruiken we zoals eerder vermeld om na te denken over algoritmes en de limieten die we tegen komen bij het oplossen van problemen. In dit deel gaan we verder in op deze limieten, wanneer we problemen proberen op te lossen met behulp van algoritmes. We zijn vooral geïnteresseerd in de twee resources die onbeperkt aanwezig zijn bij de Turingmachine: tijd en opslagruimte. Juist omdat deze twee resources in echte computers eindig zijn en dus ook zeer kostbaar.

We willen dus weten hoeveel opslagruimte en hoeveel tijd er nodig is om een algoritme uit te voeren. Dit lijkt zeer eenvoudig, maar we stoten toch onmiddellijk al op enkele problemen. Hoe drukken we deze waardes uit? Gebruiken we gewoon het aantal secondes dat een computer nodig heeft om het algoritme uit te voeren? Welke computer kiezen we dan? We kunnen heel gemakkelijk zien dat wanneer men een ander model van computer kiest, de vereisten die nodig zijn van tijd en opslagruimte zeer snel kunnen veranderen. Neem nu een gewone optelling of vermenigvuldiging op een Turingmachine, zelfs de kleinste verandering, zoals overstappen op een Turingmachine met twee tapes, kan al veel tijdswinst met zich meebrengen. Daarom gaan we op zoek naar een manier om algoritmes te vergelijken en beoordelen onafhankelijk van het model van computer waarop ze worden uitgevoerd. Ook van grote invloed is de input. Een algoritme dat vermenigvuldigd, zal bij grote getallen langere tijd nodig hebben en meer geheugenruimte gebruiken. Het is dus niet alleen nodig om een manier te vinden om onafhankelijk te zijn van het model, maar we moeten ook rekening kunnen houden met een variabele input.
Om aan deze vereisten te voldoen maken we gebruik van een systeem waarbij we een bovengrens zoeken voor het aantal stappen dat het algoritme moet doorlopen in functie van de lengte van de input. De lengte van de input drukken we uit in n. Stel dat we een willekeurige functie hebben die om tot een oplossing te komen $40n + 6$ stappen nodig heeft. We kunnen hier besluiten dat de term die het meeste doorslag zal geven op de duur van de uitvoering $40n$ is. Wanneer n groot genoeg is kunnen we zelfs stellen dat de rest zeer weinig invloed heeft op het aantal stappen dat moet worden uitgevoerd. Hetzelfde voor een algoritme dat $5n^2 + 7n + 6$ stappen nodig heeft. Hier zal de meest bepalende factor bij een grote input n de factor $5n^2$ zijn. We gaan in onze schatting echter nog een stapje verder en houden ook geen rekening met de constante factor die bij n hoort, omdat deze wanneer n groter wordt minder belangrijk wordt. We zeggen voor deze voorbeelden dat deze algoritmes toenemen met factor n of n^2 afhankelijk van de input.
Wanneer we schatten hoeveel stappen een algoritme nodig gaat hebben moeten we niet alleen rekening houden met de lengte van de input maar ook met de aard van de input. Een algoritme kan veel sneller werken op een bepaalde input en veel langer nodig hebben voor een andere ook al zijn ze beiden even

lang. Denk maar aan het ordenen van een lijst. Wanneer we als input een reeds geordende lijst meekrijgen is het duidelijk dat het algoritme veel eerder hiermee klaar gaat zijn dan wanneer we een totaal ongeordende lijst meegeven. Om hiermee rekening te houden maken we gebruik van een speciale notatie the big O notatie. We gebruiken deze als volgt. We zeggen dat een functie $f(n)$ binnen $O(g(n))$ hoort als er een n_0 en c bestaan, zodat er voor elke waarde n groter dan n_0 geldt dat $f(n) \leq cg(n)$. Hiermee bedoelen we dat voor een grote n de functie $g(n)$ een bovengrens is voor $f(n)$. In ons vorige voorbeeld van $5n^2 + 7n + 6$ kunnen we zeggen dat deze $O(n^2)$ is omdat we hier een c kunnen vinden zodat $5n^2 + 7n + 6 \leq cn^2$ wanneer n groot genoeg is. Met de big O notatie duiden we dus een bovengrens aan. In het geval van algoritmes spreken we dan ook van the worst case waarmee we een bovenlimiet kunnen plaatsen op het aantal stappen dat een algoritme moet nemen in het slechtste geval.

We kunnen naast een bovengrens ook een ondergrens aanduiden. Hiervoor maken we gebruik van de volgende notatie. We kunnen een c vinden en een n_0 zodat voor elke $n > n_0$ geldt dat $cg(n) \leq f(n)$. Hiermee bedoelen we dat voor een n die groot genoeg is, de functie $g(n)$ een ondergrens zal zijn voor de functie $f(n)$. We duiden dit aan door te zeggen dat $f(n)$ in $\Omega(g(n))$ zit. Als derde is er ook nog de notatie $\Theta(g(n))$ deze gebruiken we in het geval dat $f(n)$ behoort tot $O(g(n))$ en tot $\Omega(g(n))$ voor een voldoende grote n. Deze laatste twee notaties komen in vergelijking met de grote O notatie niet zo vaak voor omdat we het meest geïnteresseerd zijn in de worst case resource benodigdheden van onze algoritmes, omdat we hier beter aan kunnen zien wat de beperkingen en limieten zijn van een algoritme. Eerder verwezen we naar sorteer algoritmes om aan te duiden dat de soort input en de lengte hiervan een grote factor kunnen zijn op de uiteindelijke uitvoeringstijd van een algoritme. Om dit duidelijk te maken geven we nu een voorbeeld waarmee we ook de grote O en grote omega notatie proberen te verduidelijken.
Stel dat we als input een lijst van namen krijgen die we in alfabetische volgorde moeten outputten. Een niet zo goede oplossing die we kunnen bedenken is Bubblesort. Dit werkt door gewoon doorheen de lijst van namen te lopen en te controleren of de volgende naam alfabetisch voor of na de huidige naam komt. Als de volgende naam alfabetisch na de huidige naam komt, dan staan deze twee namen goed ten opzichte van elkaar. Wanneer de volgende naam alfabetisch voor de huidige naam komt, moeten we deze twee van plaats verwisselen en controleren vervolgens de volgende namen. Zo schuiven we altijd een plaats op totdat we op het einde zijn. Hierna beginnen we terug bij het begin van de lijst en lopen we er weer door tot aan het voorlaatste element omdat we zeker weten dat het laatste op zijn plaats staat. Hierna herhalen we dit weer maar stoppen weer een element vroeger en zo verder. We houden ook bij hoeveel keer we een element van plaats hebben moeten wisselen wanneer we door de lijst lopen zodat het algoritme

vroeger kan stoppen als de lijst al gesorteerd is. Wanneer we in het beste geval een volledig gesorteerde lijst als input krijgen, zal het algoritme geen enkele naam van plaats moeten verwisselen en het aantal stappen dat nodig is om 1 maal door de lijst te lopen, zijn dan ook het aantal stappen dat het algoritme nodig zal hebben om de lijst te sorteren. Dit geval beschouwen we als de ondergrens van het algoritme. In het slechtste geval krijgen we een lijst waarin geen enkele naam op de juiste plek staat, het algoritme zal dan iedere keer de namen moeten verwisselen en zal dus het maximum aantal keren door de lijst moeten stappen, telkens met een element minder want het laatste staat na een iteratie altijd juist. Dit geeft ons in totaal $(n-1)+(n-2)+(n-3)+...+1$ keren dat we door de lijst moeten lopen wat neerkomt op $n(n-1)/2$. We zeggen dan dat dit algoritme behoort tot $O(n^2)$. Hiermee bedoelen we dan dat het maximum aantal stappen dat dit algoritme zal moeten doen nooit groter zal zijn als n^2 waarbij n het aantal namen op de lijst voorstelt.

Waarom doen we zoveel moeite om te kunnen bepalen wat de limieten van een algoritme zijn? Waarom is het zo belangrijk om te weten hoeveel tijd en opslagruimte een algoritme nodig zal hebben om tot een oplossing te komen? In de informatica is er een tak die zich bezig houdt met alleen maar zoeken naar de tijd en opslagruimte die algoritmes nodig hebben om een oplossing te vinden, namelijk de complexiteitstheorie.
Voor veel problemen kennen we een efficiënte oplossing, hiermee bedoelen we dat ze weinig tijd nodig hebben om een oplossing te vinden en tijdens het berekenen weinig computergeheugen gebruiken. Er zijn echter ook problemen waar we voorlopig nog geen efficiënte oplossing voor hebben gevonden of voor kunnen vinden. Er zijn zelfs problemen waar de bekende oplossingen zo inefficiënt zijn dat ze zoveel tijd en/of opslagruimte nodig hebben dat we ze als waardeloos beschouwen. We beschouwen het probleem dan niet op te lossen met behulp van computers. Ook hier komt weer de reden voor het ontwikkelen van Quantumcomputers naar voor. Kunnen we voor deze problemen algoritmes vinden die op een Quantumcomputer wel binnen een bruikbare tijd en ruimte tot een oplossing komen?
De complexiteitstheorie houdt zich dus bezig met uitzoeken hoeveel tijd en opslagruimte een algoritme zal nodig hebben om tot een resultaat te komen. Dit gebeurt door te bewijzen dat de uitvoering van een algoritme binnen bepaalde grenzen ligt. Hier onderscheiden we twee gevallen. Voor bekende algoritmes bewijst men dat de uitvoering onder een bepaalde bovengrens zal liggen, hiermee kan men dan inschatten hoe effectief een algoritme is. Maar men probeert ook voor een probleem te bewijzen wat de ondergrens zal zijn voor de uitvoering van het best mogelijke algoritme, ook al is dit algoritme nog niet bekend. Wanneer we dan een algoritme vinden om een probleem op te lossen kunnen we de effectiviteit hiervan toetsen aan de theoretische ondergrens die bewezen is voor een probleem.

Naast het bewijzen hoe effectief een oplossing is of kan zijn gaan we in de complexiteit algoritmes en problemen groeperen in verzamelingen van vergelijkbare effectiviteit. Zoals we eerder zeiden gaan we snelheid en het gebruik van ruimte niet uitdrukken in getallen specifiek voor een model van computer of voor een specifieke machine, we maken gebruik van een variabele in functie van de inputlengte. Deze functie gebruiken we dan om een boven -en/of ondergrens te geven voor het algoritme. Een vaak gebruikt onderscheid dat wordt gemaakt tussen algoritmes zijn deze waarvan de resources toenemen in functie van de inputlengte begrensd door een polynomiale functie. De tweede verzameling zijn de problemen en algoritmes waarvan de oplossingstijd niet kan worden begrensd door een polynomiale functie. Deze laatste groep wordt vaak ook de exponentiële algoritmes genoemd omdat de resources die nodig zijn om het probleem op te lossen exponentieel (of nog sneller) stijgen in functie van de inputlengte. De naam voor deze groep is eerder ongelukkig gekozen omdat ze ook nog problemen bevat die nog slechtere oplossingen hebben dan exponentieel, de resources stijgen dus nog sneller dan we kunnen uitdrukken met een exponent.
Vaak beschouwen we de eerste groep van algoritmes die problemen in polynomiale tijd oplossen als efficiënt en de tweede groep als inefficiënt of zelfs niet oplosbaar in realistische tijd. Wanneer we een probleem gaan groeperen kiezen we steeds voor de best bekende oplossing of de best bewezen mogelijke theoretische oplossing. Deze opdeling in twee grote groepen is de meest gebruikte, maar kan in sommige extreme gevallen voor een vertekening zorgen. Stel dat we een algoritme hebben dat een oplossing vindt in n^{10000} stappen, dan is dit een polynomiaal algoritme en wordt het in het algemeen als effectief beschouwd. Als we voor hetzelfde probleem een algoritme vinden dat een oplossing vindt in $2^{n/10000}$ stappen, dan hoort dit tot de verzameling van exponentiële problemen en zouden we dit als ineffectief beschouwen. Toch is het makkelijk te zien dat het tweede "inefficiënte" algoritme veel sneller een oplossing zal vinden behalve wanneer n heel groot zal zijn. Toch maken we gebruik van deze opdeling in twee grote groepen, omdat het voor de meeste algoritmes lijkt te kloppen dat een polynomiale oplossing veel beter omgaat met de tijd en opslagruimte die het nodig heeft om een oplossing te vinden. We vermelden ook nog dat het vinden van een polynomiale oplossing voor een probleem vaak veel moeilijker is dan het vinden van een eenvoudige en vaak intuïtieve exponentiële oplossing.
De opdeling tussen polynomiale en niet-polynomiale problemen is niet de enigste die we maken in de informatica maar wel de belangrijkste, omdat ze in de meeste gevallen zeer goed aangeeft welke problemen efficiënt oplosbaar zijn en welke niet.

2.2.1 P en NP

In de informatica is het mogelijk om de algoritmes die we kennen te herleiden naar een serie van ja/nee vragen. De meeste problemen zijn van de vorm "Is oplossing m een oplossing voor probleem x?" Deze problemen worden beslissingsproblemen genoemd, omdat ze opgelost worden door een beslissing te nemen. We kunnen deze problemen ook makkelijk voorstellen op een Turingmachine. Hiervoor maken we gebruik van talen en alfabetten. Een alfabet is een verzameling van symbolen die we kunnen gebruiken om een string te maken. Zo is bijvoorbeeld de verzameling $\sum = \{0, 1\}$ het alfabet dat nodig is om een binaire string voor te stellen. Een taal is een deelverzameling van de verzameling van alle mogelijke strings die we kunnen maken met een alfabet. Zo is bijvoorbeeld $L = 0, 10, 100, 110, ...$ de taal die alle even binaire getallen bevat.
Op welke wijze wordt een beslissingsprobleem voorgesteld? We kunnen dit probleem voorstellen als de functie $f : \sum^* \to \{ja, nee\}$. De taal die dan beslist wordt door deze functie stellen we voor als $L_f = \{ x \in \sum^* \mid f(x) = ja \}$. Het is ook mogelijk om de functie voor te stellen als een Turingmachine. We zeggen dat een Turingmachine een taal beslist (decides) wanneer er voor elke willekeurige input x op de tape kan beslist worden of deze wordt geaccepteerd (accept) in de taal wordt afgewezen (reject). Hiervoor passen we onze Turingmachines aan, in plaats van 1 halting toestand definiëren we nu twee halting toestanden $q_a$en q_r die aanduiden of de input x geaccepteerd of afgewezen wordt in de taal L. Een Turingmachine die dit voor elke input x kan beslissen noemen we een beslisser (decider) voor die taal. Deze Turingmachine voert dus het algoritme uit om het beslissingsprobleem op te lossen.

Hoe kunnen we nu bepalen hoe snel een probleem kan worden opgelost? of anders geformuleerd: hoe snel kan een Turingmachine een taal L beslissen, waarbij de taal L het beslissingsprobleem dat we willen oplossen beschrijft. We zeggen dat een probleem behoort tot $TIME(f(n))$ als we een Turingmachine kunnen vinden die voor input x kan beslissen of ze tot de taal behoort in een tijd die begrensd wordt door $O(f(n))$, waarbij n de lengte is van input x. We zeggen dat een probleem opgelost kan worden in polynomiale tijd als ze behoort tot $TIME(n^k)$ waar k een eindig getal is. Deze verzameling van problemen oplosbaar in polynomiale tijd wordt ook vaak aangeduid als de verzameling problemen in P. We noemen P een complexiteitsklasse. We zijn niet alleen geïnteresseerd in welke problemen deel uitmaken van een complexiteitsklasse, maar ook de verhoudingen en relaties tussen verschillende klassen. Als we een klasse definiëren om problemen in te groeperen die oplosbaar zijn in polynomiale tijd geeft dit ook aan dat er problemen zijn die niet oplosbaar zijn in polynomiale tijd. Het is echter moeilijk om voor sommige problemen te beslissen of ze al dan niet tot de klasse P behoren, voor andere problemen is dit dan weer makkelijk.

Zo dacht men eerst dat het probleem PRIME (het beslissen of een getal een priemgetal is) geen oplossing had in P maar in 2004 werd bewezen dat er wel degelijk een algoritme bestaat dat kan beslissen of een getal een priemgetal is in polynomiale tijd [2].
Een probleem dat samenhangt met PRIME is het factoring probleem. Het factoring probleem bestaat uit het volgende: gegeven een getal m en een getal l waarvoor geldt dat $l < m$. Kunnen we een factor vinden van m zodat de gevonden factor kleiner is dan l. Voor zover we weten is het niet mogelijk om hiervoor een algoritme te vinden dat tot P behoort. Later zullen we zien dat factoring op een Quantumcomputer wel efficiënt kan.
We kunnen voor het factoring probleem dus niet in polynomiale tijd tot een oplossing komen, maar als we een oplossing gevonden hebben, kunnen we wel zeer snel nagaan of dit een correcte oplossing is, namelijk door twee simpele tests. Is het gevonden getal kleiner dan l, en is m deelbaar door het gevonden getal? Deze twee stappen kunnen we zeer snel en efficiënt uitvoeren. De mogelijke oplossing noemen we dan de witness van het probleem. We beweren dat deze witness een mogelijke oplossing is voor het probleem en dat het antwoord op het beslissingsprobleem dus ja is. We kunnen makkelijk nagaan of deze witness inderdaad een juiste oplossing is en het antwoord dus ook echt ja is. Dit soort problemen waarvoor we geen efficiënte oplossing kennen maar wel op een zeer efficiënte wijze kunnen controleren of een gevonden oplossing correct is groeperen we in de verzameling NP. NP staat niet voor niet-polynomiaal maar voor niet-deterministisch polynomiaal, omdat ze de problemen bevat die door een niet deterministische Turingmachine kunnen worden opgelost in polynomiale tijd. Er wordt ook wel soms gezegd dat NP de klasse is van beslissingsproblemen waar we geen efficiënte oplossing voor kunnen vinden maar wel de ja-oplossing in polynomiale tijd kunnen controleren met een Turingmachine. Het nagaan van de ja-oplossing zit dus in P, maar het nagaan van de nee-oplossing is vaak veel moeilijker en niet uit te voeren in polynomiale tijd. Zo kan men voor het factoring probleem niet eenvoudig nagaan of nee een juist antwoord is. Men zou dit pas kunnen beslissen nadat men van alle mogelijkheden heeft getest of ze geen ja-oplossing zijn. We creëren hiervoor een nieuwe klasse die alle problemen bevat waarvoor ook efficiënt een nee-oplossing kan worden gecontroleerd. Deze klasse noemen we CoNP. De problemen die in CoNP zitten zijn een complement van een probleem in NP. Er zijn ook problemen die zowel in NP als in CoNP zitten, zoals bijvoorbeeld alle problemen uit P.

Een meer belangrijkere verhouding is deze tussen P en NP. Het is duidelijk te zien dat $P \subseteq NP$ want als we voor een probleem snel een oplossing kunnen vinden dan kunnen we ook snel beslissen of deze oplossing een juiste oplossing is. De andere richting is veel interessanter want dan zouden we kunnen besluiten dat P gelijk is aan NP. Deze vraag of P gelijk is aan NP is een van de meest bekende problemen in informatica en er is tot op van-

daag nog steeds geen oplossing voor gevonden. Het $P = NP$ probleem is zelfs één van de millenniumprijsproblemen van het Clay Mathematics Institute in Massachusetts. De millenniumprijsproblemen zijn 7 problemen die in 2000 werden uitgekozen door het Clay Mathematics Institute. Degene die voor één van deze problemen een oplossing vindt, krijgt een geldprijs van 1 miljoen dollar. Naast waarschijnlijk eeuwige roem in de wetenschappelijke wereld. Deze problemen zijn al jarenlang het onderwerp van verschillende onderzoeken. Tot op vandaag is er nog maar een van deze zeven problemen opgelost, namelijk het Poincaré probleem. De ontdekker van de oplossing heeft de geldprijs wel geweigerd.
Het $P = NP$ probleem is volgens velen een van de belangrijkste problemen in de huidige wetenschap. Het vinden van een oplossing zou immers zeer grote gevolgen hebben en dit niet enkel binnen de informatica en wiskunde. Om aan te tonen hoe groot deze gevolgen zouden zijn, volgt hier een misschien ietwat dramatische quote van Scott Aaronson.

> "If P = NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps,' no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss..." — Scott Aaronson, MIT

Hoewel we voor dit probleem nog geen oplossing hebben gevonden zijn de meeste wetenschappers het toch eens dat $P \neq NP$, maar zoals reeds gezegd, is er nog steeds geen sluitend bewijs hiervoor gevonden en bestaat nog steeds de mogelijkheid dat $P = NP$.
We definiëren in de klasse NP nog een zeer belangrijke subset van problemen, namelijk de problemen die NP-compleet zijn. NP-complete problemen worden beschouwd als de moeilijkste problemen in NP. Om in de klasse te komen moet een probleem A voldoen aan twee voorwaarden. Als eerste moet A behoren tot NP, als tweede moet elk ander probleem in NP reduceerbaar zijn tot A en dit in polynomiale tijd. Wat bedoelen we hiermee? Een probleem is reduceerbaar naar een ander probleem als we een Turingmachine kunnen vinden die voor een input x voor probleem B een resultaat $R(x)$ geeft. Deze $R(x)$ kan dan gebruikt worden voor probleem A en er geldt $x \in B \iff R(x) \in A$. We zeggen dan dat B reduceerbaar is tot A. Als de uitvoeringstijd van deze Turingmachine polynomiaal is, dan zeggen we dat de taal B polynomiaal reduceerbaar is naar A. Dus als we een oplossing kunnen vinden voor één van deze problemen, kunnen we dus ook een oplossing vinden voor het andere probleem in dezelfde tijd mits een beetje overhead van het reduceren. We zien nu dat wanneer we een polynomiale oplossing zouden vinden voor een probleem dat NP-compleet is we een polynomiale oplossing hebben gevonden voor alle problemen in NP want alle problemen

zijn reduceerbaar tot een NP-compleet probleem. En dat we dan hebben bewezen dat $P = NP$.
Bestaan er ook problemen die in NP zitten maar niet NP-compleet zijn? Als we aannemen dat $P \neq NP$ dan kunnen we bewijzen dat er een klasse moet bestaan NPI (waar de I voor intermediate staat) die niet leeg is en die de problemen bevat die in NP zitten maar niet NP-compleet zijn en ook niet in P zitten. (Ladner's theorem [10].)
Het is uit deze definitie duidelijk dat we nog geen problemen kennen waarvan we weten dat ze in NPI zitten anders zouden we ook al weten dat $P \neq NP$. Er zijn echter wel problemen waarvan men een sterk vermoeden heeft dat ze in NPI zitten zoals isomorfisme tussen grafen en factoring. Het zijn net deze problemen waar we in geïnteresseerd zijn bij Quantum Computing, omdat problemen in P al efficiënte oplossingen hebben en er verwacht wordt dat we voor NP-complete problemen geen efficiënte oplossing kunnen vinden. Omdat we een Quantumalgoritme kennen dat factoring efficiënt oplost gaan we er van uit dat er nog problemen zijn, waarvan we verwachten dat ze in NPI zitten, waarvoor we een efficiënt Quantumalgoritme kunnen vinden.

Wanneer we willen bewijzen dat een probleem A NP-compleet is moeten we dus twee dingen bewijzen. Als eerste, dat A in NP zit, dit is meestal de makkelijkste stap om te bewijzen. We bewijzen dit door te laten zien dat het moeilijk is om een oplossing te vinden voor A maar wel mogelijk is een gevonden oplossing te verifiëren in polynomiale tijd. In het tweede deel van het bewijs moeten we bewijzen dat elke taal in NP reduceerbaar is naar A. Omdat dit enorm veel werk zou zijn om dit voor elk probleem in NP te doen, beperken we dit bewijs tot het bewijzen dat er een NP-compleet probleem bestaat dat reduceerbaar is tot A, dit kan om de volgende reden. Stel we hebben naast A nog twee problemen B en C en van deze twee laatste weten we dat ze NP-compleet zijn dan geldt $B \leq_P C$ (we gebruiken $\leq_p$ om een polynomiale reductie aan te duiden). Als we nu bewijzen dat $C \leq_p A$ dan geldt ook vanzelfsprekend $B \leq_p A$ want $B \leq_p C \leq_p A$. Om hiervan gebruik te kunnen maken, moeten we echter eerst een probleem hebben waarvan we al bewezen hebben dat het NP-compleet is. Hiervoor maken we meestal gebruik van het SAT-probleem [27].
De klasse NP is voor ons zeer interessant omdat ze veel problemen bevat die wij beschouwen als niet oplosbaar in realistische tijd. Vaak duurt de uitvoering van de bekende algoritmes te lang. De oplossing voor deze problemen komt ook vaak neer op het nagaan van alle mogelijke oplossingen en deze dan controleren. Net omdat deze problemen onoplosbaar lijken te zijn, gaan we hiervoor nieuwe manieren zoeken om over berekeningen na te denken. Dit brengt ons weer terug bij ons onderwerp. Is het mogelijk om met behulp van een Quantumcomputer problemen efficiënter op te lossen? En vooral is het mogelijk om problemen efficiënter op te lossen die niet in P zijn, want voor deze hebben we al goede oplossingen. Velen denken dat het

echter niet mogelijk is om voor alle problemen in NP efficiënte oplossingen te vinden waarmee we dus het bestaan van de klasse NPI bewijzen, voorlopig bestaat er een Quantumalgoritme dat factoring op een efficiënte wijze oplost. Factoring is een probleem waarvan men vermoedt dat het in NPI zit. Later komen we nog terug op dit algoritme.

We kunnen nu ook complexiteitsklassen definieren in termen van de geheugenruimte die de algoritmes nodig hebben. We houden nu dus geen rekening meer met de tijd die het algoritme nodig heeft. We gebruiken weer Turingmachines als computermodel en lossen weer beslissingsproblemen op. We kunnen de benodigde geheugenruimte ook zien als de hoeveelheid vakjes die we zullen gebruiken op de tape van de Turingmachine. Opnieuw delen we de klasse op in twee grote groepen. Als eerste definiëren we PSPACE, de klasse van problemen die opgelost kunnen worden op een Turingmachine door gebruik te maken van een polynomiaal aantal vakjes op de tape ten opzichte van de grootte van de input. De tijd die nodig is om te komen tot een oplossing is niet belangrijk en dus ongelimiteerd. Als tweede definiëren we de klasse NPSPACE waarbij de gebruikte Turingmachine non deterministisch is. Nu geldt echter dat PSPACE en NPSPACE equivalent zijn (bewezen door Walter Savitch in 1970 [27]). Dit kunnen we zien doordat het mogelijk is om een non-deterministische Turingmachine te simuleren op een deterministische Turingmachine zonder dat hiervoor veel meer ruimte nodig is. Er is wel veel meer tijd nodig in deze simulatie maar hier houden we in deze klasse geen rekening mee.
Het is makkelijk in te zien dat P een deelverzameling is van PSPACE, het is namelijk onmogelijk om in polynomiaal aantal stappen meer dan een polynomiaal aantal vakjes aan te spreken op de band van de Turingmachine. Maar ook NP is een deelverzameling van PSPACE want $NPSPACE = PSPACE$, dit kunnen we nagaan door te overlopen hoe een probleem wordt opgelost op een non-deterministische Turingmachine. Vaak zijn de oplossingen van NP-problemen niets meer dan het nagaan van de correctheid van elke mogelijke oplossing, deze oplossing zal echter steeds polynomiaal lang zijn. Als we alle mogelijke oplossingen achter elkaar uitproberen en tussen twee verschillende mogelijke oplossingen het geheugen terug wissen zullen we nooit meer dan een polynomiale hoeveelheid ruimte gebruiken. We kunnen later ook nog concluderen dat ook de problemen die efficiënt oplosbaar zijn op een Quantumcomputer een deelverzameling zijn van PSPACE. Wat we echter nog niet weten is of $PSPACE = P$. Moest dit wel zo zijn dan zou zeker gelden dat $P = NP$. En dan zou ook gelden dat P = de klasse van efficiënt oplosbare problemen op een Quantumcomputer. Voor het verdere verloop van deze masterproef hopen we dat dit niet het geval is net zoals door veel computerwetenschappers wordt verwacht.

Voor de volledigheid introduceren we nog twee extra complexiteitsklassen

LOGSPACE en EXP . EXP is de klasse van problemen die oplosbaar zijn op een Turingmachine in exponentiële tijd de problemen die als bovengrens $O(2^{n^k})$ hebben, k is hier een constant getal. De klasse LOGSPACE bevat de problemen die door een Turingmachine kunnen beslist worden met behulp van logaritmisch geheugengebruik. Hier moeten we echter nog opmerken dat er gebruik gemaakt wordt van een Turingmachine met twee tapes, een tape die alleen gelezen kan worden waar de input op staat. En een tweede werktape, waar we op kunnen schrijven en lezen. De eis voor het logaritmisch geheugengebruik is enkel voor de tweede tape.
De complexiteitsklassen zijn dus gebaseerd op de hoeveelheid tijd en geheugen die we gebruiken om het algoritme uit te voeren. Nu kunnen we ons afvragen of de toevoeging van meer tijd en ruimte ervoor zorgt dat we meer rekenkracht krijgen en dus moeilijkere problemen kunnen oplossen. Het antwoord hierop is tweemaal 'Ja' maar zoals al regelmatig aangehaald zijn er problemen die zoveel tijd of opslagruimte nodig hebben dat we de oplossing ervan niet als realistisch haalbaar beschouwen. Zo kan de tijd die nodig is om een probleem uit NP met een redelijke inputgrootte al snel toenemen tot enkele honderden jaren, wat voor ons niet handig is. Verder hebben we ook gezien dat de verschillende complexiteitsklassen onderling ook met elkaar gerelateerd zijn, we vatten deze relaties hieronder nog even samen.

$$LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXP$$

We zien dat de relatie deelverzameling steeds in dezelfde richting wordt gebruikt. Hierbij stellen we ons de vraag of de andere richting ook klopt. Met andere woorden, zijn er nog klassen die gelijk zijn aan elkaar of zijn deze relaties strikt. Het antwoord op deze vragen weten we nog niet zeker maar we gaan er van uit dat deze relaties strikt zijn. Men is er wel zeker van dat P een strikte deelverzameling is van EXP en dat LOGSPACE een strikte deelverzameling is van PSPACE. Dus minstens een van de relaties hierboven moet een strikte deelverzameling zijn.
We komen terug op het feit dat problemen uit NP zoveel resources nodig hebben dat ze als onoplosbaar worden beschouwd. Wat moeten we dan doen wanneer we zo een probleem tegenkomen. Er zijn verschillende manieren om de schijnbare onoplosbaarheid van problemen te omzeilen. Een eerste manier is om een speciale vorm van het probleem te zoeken waarvan we wel een efficiënte oplossing kunnen vinden, zoals 2SAT dat wel oplosbaar is maar SAT en 3SAT niet. Een voor ons interessantere aanpak is het zoeken van een oplossing die problemen uit NP oplost, door niet de exacte oplossing te zoeken maar een benadering die dicht in de buurt komt. Zo dicht dat ze voor ons even goed als correct is. Als we een NP-compleet probleem kunnen oplossen via een benadering dan lijkt het dat het mogelijk is om alle problemen uit NP op te lossen want we kunnen met behulp van reductie de oplossing voor het ene probleem gebruiken om de anderen op te lossen.

Hier wringt het schoentje echter want de benadering die eerst goed genoeg is, kan na de reductie helemaal niet meer goed genoeg zijn en zo tot foute oplossingen leiden.

Als we verder gaan op deze manier om het onoplosbare karakter van problemen te omzeilen en een oplossing accepteren kunnen we nog een zeer speciale complexiteitsklasse introduceren. Deze keer passen we de gebruikte Turingmachine aan. We gebruiken een Turingmachine die niet meer alleen stappen neemt afhankelijk van wat zijn huidige toestand en input zijn, maar ook acties kiest onder invloed van willekeurige kansen. We eisen echter dat de kans op een foute accept of reject klein genoeg is, zodat het resultaat toch betrouwbaar is. We definiëren voor deze Turingmachine dan de klasse BPP (bounded-error probabilistic time). Deze klasse bevat alle talen L waarvoor we een Turingmachine M kunnen vinden die met kansen werkt zodat deze Turingmachine x accepteert als een element van L met een kans van $3/4$ wanneer x echt een element is van L. En dat M input x reject met een kans van $3/4$ wanneer x geen element is van L. De grootte van de kans op een juist antwoord maakt op zich niet zoveel uit aangezien we het algoritme verschillende keren achter elkaar kunnen laten lopen en zo de meerderheid van de outputs kunnen laten beslissen. Wanneer we het algoritme maar vaak genoeg herhalen kunnen we de kans op een correcte output opdrijven totdat deze bijna 1 is.
We noemen een algoritme bounded error als er een waarde $1/2 + \epsilon$ bestaat dat het resultaat juist is en een kans $1/2 - \epsilon$ dat het resultaat fout is. In het geval van BPP kiezen we $\epsilon = 1/4$. Wanneer we een bounded error algoritme meerdere keren achter elkaar uitvoeren zal de juiste oplossing meer voorkomen dan de foutieve. We kunnen dus de zekerheid van een correcte oplossing vergroten door het algoritme meer uit te voeren. Maar hoe betrouwbaar is dit resultaat en hoeveel keer moeten we een algoritme uitvoeren voordat we het resultaat kunnen vertrouwen? Dit lossen we op met behulp van de Chernoff Bound uit de kansberekening.

Chernoff Bound
Stel $X_1, X_2, ..., X_n$ zijn willekeurige en onafhankelijke getallen die ofwel 1 zijn met een kans $1/2 + \epsilon$ en 0 zijn met een kans $1/2 - \epsilon$. De kans dat er meer getallen 0 zijn dan 1 kunnen we ook voorstellen als de kans dat de som van deze getallen kleiner is dan de helft van het aantal getallen.

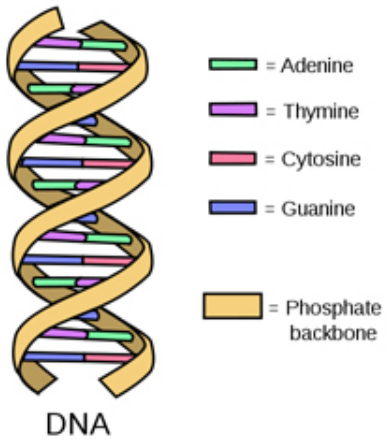
$$p(\sum_{i=1}^{n} X_i \leq n/2) \leq e^{-2\epsilon^2 n}$$

Deze kans kan men ook voorstellen als de kans dat we toch meer foutieve antwoorden krijgen als juiste, wanneer we ons algoritme n keer uitvoeren.

We zien dus dat de correctheid exponentieel toeneemt als we het aantal keer dat we het algoritme uitvoeren verhogen voor eenzelfde ϵ. In ons geval bij BPP is de waarde van $\epsilon = 1/4$. We kunnen hierdoor zien dat we al snel een foutmarge kunnen bereiken die zo klein is, dat er meer kans is dat de fout niet uit het algoritme zelf komt maar door een fout van een hardware component.
De klasse BPP is dus efficiënt in het oplossen van problemen op een betrouwbare wijze, daarom beschouwen we de klasse BPP vaak als de klasse van problemen die oplosbaar zijn op een computer zoals we die nu kennen. We definiëren ook al een variant op deze klasse BQP die de analoge klasse is, maar voor Quantumcomputers. Hier komen we later op terug want dit is de klasse waarin we geïnteresseerd zijn.

2.3 Rekenkracht

We hebben het nu gehad over de resources die nodig zijn om problemen te kunnen oplossen op een computer. We hebben steeds de Turingmachine gebruikt als het model om een computer mee voor te stellen. We willen in dit deel opnieuw kijken naar de rekenkracht die we kunnen krijgen om problemen op te lossen maar laten de Turingmachine achter ons en overlopen verschillende computermodellen waarvan we verwachten dat ze grote rekenkracht hebben zoals parallel computers en DNA-computers. We zijn geïnteresseerd hoe krachtig deze modellen zijn in vergelijking met een Turingmachine en hoe ze zich gedragen ten opzichte van de tijd en opslagruimte die ze nodig hebben om te werken.
Als eerste gaan we het hebben over parallel computing. Bij parallel computing proberen we een snelheidswinst te maken bij het oplossen van problemen door een groot probleem op te splitsen in kleinere problemen en deze tegelijk uit te voeren. Tegenwoordig beschikken de meeste PC's over verschillende cpu-cores. Parallel computing is dus niet zo uitzonderlijk meer. Er bestaan echter ook meer gespecialiseerde versies van parallel computing waarbij men verschillende computers laat samen werken in plaats van een computer met verschillende cores. Deze vorm van computing kan een grote tijdswinst opleveren bij problemen die goed op te splitsen zijn in kleinere problemen. Echter moeten we tijdens het splitsen ook rekening houden met een overhead die gecreëerd wordt doordat de verschillende deelproblemen moeten worden gecoördineerd en terug verwerkt worden. Ook moeten de verschillende computers of cores communiceren met elkaar. Het is ook niet mogelijk om alle problemen op te splitsen, dus kunnen we parallel computing niet zomaar gebruiken.
Als we kijken naar de verhouding van totale resources (tijd + opslagruimte) die nodig zijn bij parallel computing in vergelijking met de Turingmachine besluiten we dat parallel computing niet meer rekenkracht geeft, omdat we



Figuur 7: DNA - molecule (Bron: `http://www.hartnell.edu/tutorials/biology/dnareplication.html`)

de winst die we in de tijd maken terug inleveren op vlak van geheugen en aantal computers dat nodig is. Het is echter voor mensen interessanter om minder tijd nodig te hebben dan minder ruimte om een oplossing te vinden voor een probleem.

Een speciale vorm van parallel computing is DNA computing. Hier maakt men gebruik van DNA-moleculen om berekeningen uit voeren in plaats van elektrische signalen. De DNA-molecule wordt bewaard en wordt met behulp van chemische processen gemanipuleerd om zo berekeningen uit te voeren. Omdat DNA zo enorm klein is passen er in een klein reageerbuisje al snel een enorme grote hoeveelheid DNA-moleculen. Hierin ligt ook de grote kracht van DNA computing. Bij DNA computing maakt men gebruik van de vier basen die kunnen voorkomen op een DNA-molecule om data te coderen in plaats van 0 en 1. Deze vier basen zijn Adenine, Thymine, Guanine en Cytosine. Zie figuur 7.

Omdat het mogelijk is enorm veel DNA-moleculen op een kleine plaats te bewaren is het mogelijk om algoritmes te gebruiken die zeer veel mogelijke oplossingen gelijktijdig testen. De DNA-computer werkt echter alleen maar goed zolang we op een eenvoudige wijze mogelijke oplossingen kunnen genereren om te testen, ook is het niet echt een verbetering op gebied van rekenkracht ten opzichte van een Turingmachine. Zo zullen de vereisten voor de resources nog steeds dezelfde blijven. Het voordeel zit dus echt in het feit dat de DNA-computer zeer weinig fysieke plaats nodig heeft om een enorm groot werkgeheugen aan te bieden.

We merken hier echter op dat zelfs wanneer we zoveel geheugenruimte kunnen aanbieden we al zeer snel tegen fysieke grenzen aanlopen. In het artikel "On the weight of computations" van Juris Hartmanis [8] wordt een simpel algoritme voorgesteld om met behulp van een DNA-computer het Hamiltonpad te zoeken van een graaf. Het algoritme komt er op neer om alle mogelijke paden te zoeken en in verschillende stappen telkens paden te elimineren totdat we een Hamiltonpad hebben gevonden of geen pad meer overhouden. Maar zelfs als we de verschillende paden voorstellen met DNA-molecules die enorm klein zijn zal het exponentiële karakter van dit algoritme zorgen dat het onbruikbaar wordt. Stel dat de graaf waarvoor we een Hamiltonpad zoeken 200 knopen bevat. In het artikel stellen ze dat een DNA-molecule die een pad in deze graaf voorstelt minstens uit $\log_4 200$ basen bestaat en een base weegt ongeveer $10^{-25}kg$. Het totale gewicht van alle mogelijke paden komt dan overeen met:

$$2^{200} \cdot \log_4 200 \cdot 10^{-25}kg \geq 3 \cdot 10^{25}kg$$

Dit gewicht is groter dan het totale gewicht van de aarde. Het is dus duidelijk dat dit niet erg realistisch is om uit te voeren. Hiermee probeert Hartmanis aan te tonen dat DNA computing zeker voordelen kan hebben, maar dat exponentiële groei een zo sterke grens is dat we er moeilijk omheen kunnen.

Een derde model dat gebruikt kan worden zijn de gedistribueerde systemen. Deze zijn te vergelijken met parallel computing omdat men ook hier het probleem opsplitst in kleinere stukken. Alleen maakt men bij gedistribueerde systemen gebruik van computers die fysiek gescheiden zijn en dus niet een computer met verschillende CPU's. Het is duidelijk dat ook deze vorm niet meer rekenkracht heeft dan een Turingmachine, net als bij parallel computing. De totale rekenkracht van alle computers is misschien heel groot maar hoe kunnen we deze zo efficiënt mogelijk aanspreken. We krijgen hier zelfs te maken met een nieuw probleem met resources. Hoe kunnen we zo efficiënt mogelijk de verschillende computers met elkaar laten communiceren. Gedistribueerde systemen kan gebruikt worden op computers die hier speciaal voor bestemd zijn. Maar met de komst van het internet wordt het mogelijk om over heel de wereld computers aan te spreken, om te helpen berekeningen te maken, zelfs de PC's van een gewone thuisgebruiker. Een voorbeeld hiervan is het Playstation 3 cluster. Hier kunnen gebruikers van de Playstation 3 zich voor aanmelden bij het folding@home project (`http://folding.stanford.edu/`). Wanneer hun apparaat dan idle was kreeg het instructies om een berekening uit te voeren en het resultaat terug te sturen. Er bestaan voor PC's gelijkaardige projecten waarbij gewone gebruikers de rekenkracht van hun PC kunnen aanbieden wanneer ze deze zelf niet gebruiken. Op deze manier kan men al snel een zeer groot aantal

computers bekomen om mee te rekenen.
Zoals hierboven beschreven halen we vaak aan dat de verschillende soorten computers geen rekenkracht toevoegen ten opzichte van de Turingmachine. Hiermee bedoelen we dat we geen winst maken op vlak van efficiëntie wanneer we deze meten met behulp van de bovengrens, waarbij we het aantal stappen tellen dat nodig is om tot de oplossing te komen. Maar deze computers helpen wel problemen oplossen in een snellere tijd door veel meer ruimte mogelijk te maken (DNA) of door verschillende stappen tegelijk uit te voeren. Omdat voor mensen de tijd die nodig is om een resultaat te krijgen veel belangrijker is, hebben deze computers toch voordelen en zijn dus belangrijk om te onderzoeken.

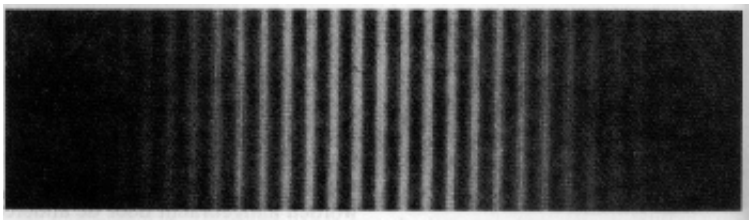
3 Quantummechanica

Nadat we in het kort informatica hebben besproken, willen we het nu hebben over het tweede vakgebied dat komt kijken bij Quantum Computing, namelijk de Quantummechanica.
Quantummechanica is een theorie uit de fysica die rond de jaren 1920 ontdekt is. De Quantummechanica beschrijft het gedrag van materie en energie op een atomair of subatomair niveau. De Quantummechanica is echter geen theorie zoals we gewoon zijn in de fysica, zoals het elektromagnetisme of de relativiteitstheorie. Quantummechanica is eerder een wiskundige basis of een set regels, waaraan de fysica zich moet houden en op verder kan bouwen. Daarom hoort het misschien eerder bij de wiskunde dan bij de fysica. De Quantummechanica wordt vaak beschreven als het operating system waar alle andere theorieën uit de fysica op werken. Allemaal behalve de relativiteitstheorie, hoewel men vermoedt dat er ook hier een verband te vinden is. In dit deel doen we ons best om op een korte en bondige manier uit te leggen wat Quantummechanica inhoudt en hoe we hier gebruik van kunnen maken. Mochten er na het lezen van dit deel nog een aantal elementen onduidelijk blijven, onthoud dan volgende quote van één van de grondleggers van de Quantummechanica.

> I think I can safely say that nobody understands quantum mechanics. - Richard Feynman

Daar we Quantummechanica beschouwen als de basis van alles, is het wel eigenaardig dat deze theorie pas zo laat ontdekt werd. Doordat de resultaten van een experiment niet te verklaren waren, werd het noodzakelijk de Quantummechanica te ontdekken. Dit experiment "double slit experiment" leggen we hieronder kort en eenvoudig uit. Het experiment komt voort uit het experiment dat Young gebruikte om de dualiteit van licht aan te tonen, hoewel dit oorspronkelijk niet zijn bedoeling was. De intentie van Young was om te bewijzen dat licht zich als een golf gedraagt, om zo anderen tegen te spreken die beweerden dat licht bestond uit deeltjes. Maar het experiment toonde aan dat licht zich gedroeg volgens de wetten van de toen nog onbekende Quantummechanica, de ene keer zich als een deeltje gedraagt, de andere keer zich als een golf.

Double Slit Experiment
Materie wordt in de wetenschap vaak voorgesteld als bolletjes. Stel nu dat we een muur of scherm plaatsen op een bepaalde afstand. We vuren in de richting van deze muur willekeurig materie af. Dan zal deze de muur raken op willekeurige plaatsen. Wanneer we echter voor de muur een plaat houden, waarin we een spleet open laten en dan opnieuw materie afvuren op de muur, dan zal de muur alleen geraakt worden op de plaatsen die zich



Figuur 8: Interferentiepatroon (Bron: `http://www.staff.science.uu.nl/~Hoekz103/wwwpmn/05-06/ppp23.htm`)

direct achter de spleet bevinden. Hetzelfde gebeurt wanneer we een plaat gebruiken met twee spleten. We zien dan dat de materie de muur alleen kan raken onmiddelijk achter een van de twee spleten.
We gaan nu in plaats van deze bolletjes materie gebruik maken van elektronen die we ook voorstellen als bolletjes, maar veel kleiner. Wanneer we de elektronen afvuren op de muur komen ze ook op een willekeurige plaats terecht. We plaatsen nu het scherm met een spleet voor de muur en vuren opnieuw elektronen af. We zien dat de plaatsen waar de muur geraakt wordt door elektronen, een lijn vormen op de plaats onmiddelijk achter de spleet. We verwachten dus dat elektronen zich gedragen zoals we al gewoon zijn van materie. Maar er is echter iets zeer eigenaardig aan de hand wanneer we elektronen in de richting van de muur schieten en ze eerst door het scherm met twee spleten laten passeren. We verwachten dat we dan mooi twee lijnen met elektronen krijgen op de plaats onmiddelijk achter de twee open gelaten spleten. Dit is echter niet het geval. Wanneer we op de muur gaan kijken waar de elektronen terecht zijn gekomen, krijgen we het patroon uit figuur 8 te zien.

Aan de hand van dit resultaat lijkt het alsof de elektronen door de plaat met twee spleten zijn gegaan en de muur hebben geraakt op plaatsen die volgens ons niet mogelijk zijn. Dit patroon kennen we echter binnen de fysica al; het is een interferentiepatroon. Dit patroon is zeer typisch voor golven en zou voorkomen wanneer we in plaats van bolletjes, golven naar de muur zouden sturen door de twee spleten. Uit dit resultaat lijkt het dus dat wanneer elektronen worden afgevuurd en dus contact verliezen met de rest van het universum, ze zich niet meer als een deeltje gedragen maar als een golf. Maar waarom merken we dit gedrag dan niet op wanneer we elektronen door het scherm met maar één spleet sturen? Om dit te verklaren moeten we nog een zeer eigenaardig feit bekijken dat zich voordoet bij dit experiment en bij de Quantummechanica. Als we de elektronen door het scherm met maar één spleet laten gaan, gedragen ze zich wel als een deeltje en niet als een golf. Maar wanneer we het elektron volgen wanneer we het afschieten, door te meten langs welke van de twee spleten het passeert, merken we op

dat het elektron zich ook niet als een golf zal gedragen. Dit zouden we als volgt kunnen verklaren. Wanneer we een elektron afschieten, verliest het het contact met de wereld rondom zich en bestaat dan niet meer als een deeltje maar alleen nog als een golf van kansen om zich op die plaats te bevinden. Het is pas wanneer we het elektron "verplichten" om zijn locatie te kiezen, dat we zien dat de golf terug verandert in een deeltje. Dit verplichten om een locatie te kiezen gebeurt wanneer we een scherm plaatsen met een spleet, of meten of het elektron door een van de twee spleten passeert. Vreemd genoeg verplichten we het elektron ook te verschijnen wanneer het passeert door de spleet die we niet aan het meten zijn op dat moment.
We besluiten dus dat deeltjes zich op atomair en subatomair niveau gedragen als een golf of kansverdeling. Het zoeken wat deze kansverdeling juist is, dat is de taak van de Quantummechanica.

Schrödingers kat

Een ander gedachte-experiment dat vaak gebruikt wordt om duidelijk te maken op welke wijze we denken dat Quantummechanica werkt, is Schrödingers kat. Dit experiment wordt niet echt uitgevoerd maar we gebruiken het om de wetten van Quantummechanica eenvoudiger duidelijk te maken. We nemen tijdens het gedachte-experiment aan dat de wetten van de Quantummechanica op alle objecten werken en niet alleen op atomaire en subatomaire deeltjes.
Het idee is als volgt. We nemen een doos waarin zich een flesje gif bevindt dat zal open gaan, wanneer een bepaald radioactief atoom in de doos vervalt. De kans dat dit atoom binnen de tijd van het experiment vervalt is 50%. In de doos zit ook een kat. Wanneer we de doos dan afsluiten van de buitenwereld staat de inhoud hiervan niet meer in contact met de buitenwereld en gedraagt deze zich volgens de Quantummechanica als een kansverdeling. De kans dat de kat nog leeft is even groot als de kans dat de kat dood is en hangt af van de kans dat het atoom al dan niet vervallen is en zo het gif heeft losgelaten. We stellen dus dat de kat op hetzelfde moment zowel levend als dood is en dat we dit pas te weten kunnen komen, wanneer we de doos openen en de Quantummechanica de kat verplicht om een waarde aan te nemen uit de kansverdeling.

3.1 Quantummechanica als kanstheorie

Zoals hierboven beschreven zien we dat we Quantummechanica kunnen beschouwen als een kanstheorie, die beschrijft hoe atomaire en subatomaire deeltjes zich gedragen. Het is net deze kanstheorie die we willen benutten om problemen op te lossen met behulp van Quantumcomputers. We leggen hier daarom uit, hoe we deze kanstheorie bekijken vanuit het oogpunt

van Quantumcomputers. De kanstheorie in de Quantummechanica lijkt zeer sterk op degene die we al kennen maar we voegen hier aan toe dat de kans niet enkel een reëel positief getal moet zijn. In de Quantummechanica kan ze ook negatief of zelfs imaginair zijn.
In de klassieke kanstheorie stellen we de kans dat een bepaalde gebeurtenis zich voordoet met een getal tussen 0 en 1. Een geval waar N verschillende gebeurtenissen kunnen gebeuren, stellen we dan voor met behulp van een vector $(p_1, p_2, p_3, ..., p_n)$, waarbij de som van alle elementen van deze vector 1 is. Bijvoorbeeld als we het gooien met een dobbelsteen zouden voorstellen krijgen we de vector $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$, indien we er van uitgaan dat elke zijde van de dobbelsteen evenveel kans heeft om gegooid te worden. Deze verwachting dat de som van alle kansen 1 is, kunnen we ook uitdrukken door te zeggen dat de 1-norm gelijk moet zijn aan 1. Er zijn echter meerdere normen die we kunnen gebruiken. We kunnen ook de 2-norm of Euclidische norm hanteren. Wanneer we gebruik maken van deze norm, verwachten we niet dat de som 1 is, maar wel de vierkantswortel uit de som van de kwadraten. Als we deze norm nu gebruiken om de rest van onze kanstheorie mee te bepalen, dan krijgen we net dat wat Quantummechanica is.
Hierboven gebruiken we in de klassieke kanstheorie twee kansen die de waardes p en $p-1$ hebben en de som ervan is steeds 1. Stel dat we nog steeds twee kansen willen gebruiken die voorstellen of een bit 0 of 1 is maar we gebruiken nu de 2-norm. We krijgen dan twee getallen (α, β) die we de amplitudes noemen, waarvoor geldt dat $\alpha^2 + \beta^2 = 1$. We zijn nu niet meer beperkt tot alleen maar getallen tussen 0 en 1; we kunnen nu als amplitude elk reëel getal gebruiken. We hebben dus een bit die bepaald wordt door deze twee getallen. We moeten nu beschrijven wat er gebeurt wanneer we naar deze bit kijken. Omdat het een bit is zijn er maar twee mogelijke resultaten 0 of 1. Waarbij α^2 de kans is dat de bit 0 is en β^2 de kans dat de bit 1 is. De som van deze twee kansen is 1. Deze situatie lijkt nu sterk op deze uit de klassieke methode waar de som van de kansen ook 1 is. Maar er zit nu een verschil in in operaties die we kunnen uitvoeren, omdat we gebruik maken van de 2-norm. Wanneer we een operatie uitvoeren op een bit met klassieke kanstheorie moeten de kansen na de operatie terug voldoen aan dezelfde eis dat de som 1 is. We zijn dus beperkt tot operaties die kunnen uitgedrukt worden in een stochastische matrix. dit is een matrix waarvan de elementen in de kolommen als som 1 hebben. Wanneer we gebruik maken van de 2-norm kunnen we echter gebruik maken van unitaire matrices en nog steeds voldoen aan de eis dat $\alpha^2 + \beta^2 = 1$. We kunnen dus veel meer berekeningen mogelijk maken.
Op deze wijze hebben we dus een kanstheorie gecreëerd, waarbij we gebruik maken van reële getallen om bits voor te stellen. Maar de Quantummechanica gaat nog een stap verder en laat ook complexe getallen toe om de amplitude voor te stellen. Dit zorgt ervoor dat het niet genoeg is om een kwadraat te nemen van de amplitudes om de kansen op 0 en 1 te bekomen.

We moeten nu het kwadraat van de absolute waarde nemen. Waarom maakt de Quantummechanica gebruik van complexe getallen en niet van gewoon reële getallen? Dit komt omdat de complexe getallen algebraïsch gesloten zijn. Een veld van getallen is algebraïsch gesloten als elke bewerking op getallen uit dat veld als resultaat een getal geeft dat ook in dat veld zit. Dit is bij reële getallen niet het geval denk maar aan $\sqrt{-1}$ dat geen oplossing heeft maar wel in de complexe getallen daar geldt $\sqrt{-1} = i$. Waarom is het nodig dat we gebruik maken van een gesloten veld? Blijkbaar verwacht de Quantummechanica dat als er een operatie U bestaat er ook een operatie V bestaat zodat $V^2 = U$. En dit is alleen mogelijk met een gesloten veld.

4 Quantum Bits

Wanneer we het hebben over klassieke computers, dan slaan we informatie op in bits, die ofwel 0 ofwel 1 zijn. Analoog met deze klassieke bit definiëren we ook een Quantumbit oftewel qubit. Hoewel qubits ook fysiek kunnen worden gerealiseerd wordt er vaak over nagedacht als een puur abstract en theoretisch iets. Dit doen we omdat we dan hierop allerlei verschillende wiskundige eigenschappen kunnen gebruiken. Ook kunnen we hierdoor over Quantum Computing spreken, zonder dat we vastzitten aan een specifieke manier van werken. Hoe stellen we Quantumbits dan fysiek voor? Dit kan op verschillende manieren. Door de verschillende toestanden voor te stellen als polarisaties van een photon of als de spin van een elektron; of de verschillende toestanden waarin een elektron zich kan bevinden, wanneer het rond een atoom draait. We kunnen deze toestanden dan beïnvloeden door energie toe te voegen aan de elektronen of weg te nemen van de elektronen. We kunnen op deze manier een begintoestand van qubits gereed maken om daarna met behulp van Quantum circuits hier berekeningen op uit te voeren.

Een qubit kan net als een gewone bit zich in een toestand bevinden. De klassieke bit kan zich in 2 toestanden bevinden: 0 of 1. De qubit kan zich in vergelijkbare toestanden bevinden die we aanduiden met $|0\rangle$ en $|1\rangle$ die te vergelijken zijn met de toestanden 0 en 1 bij de klassieke bit. Deze twee noemen we de basistoestanden van een qubit. De qubit kan zich echter nog in oneindig veel meer toestanden bevinden, de qubit kan zich namelijk in een lineaire combinatie van deze basistoestanden bevinden. Dit noemen we een superpositie.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

De getallen α en β zijn complexe getallen en noemen we de amplitudes die verschillende toestanden hebben in deze superpositie. We kunnen ook nadenken over een qubit als een vector in de complexe ruimte. De toestanden $|0\rangle$ en $|1\rangle$ zijn dan speciale voorvallen die de basis zijn voor deze ruimte.
Wanneer we berekeningen uitvoeren op een klassieke computer kunnen we de bits die we gebruiken meten, wanneer we willen weten of ze 0 of 1 zijn op dat moment. Bij qubits is dit echter niet mogelijk. Wanneer we een qubit gaan meten om te weten wat de waardes zijn van de amplitudes zal dit niet lukken. We kunnen van een qubit niet weten wat de juiste waarde is van de superpositie. We zullen dus wanneer we een qubit gaan meten als resultaat alleen maar $|0\rangle$of $|1\rangle$ krijgen. We krijgen het resultaat $|0\rangle$ dan met kans $|\alpha|^2$ en resultaat$|1\rangle$ met kans $|\beta|^2$, bijkomend geldt ook nog dat $|\alpha|^2 + |\beta|^2 = 1$. Nadat we een qubit gemeten hebben, zal hij daarna ook in deze toestand blijven die we juist gemeten hebben. Waarom dit zo is, weten we niet zeker. Het is gewoon één van de wetten van de Quantummechanica. Dit is ook de reden waarom het zo moeilijk is om de resultaten van berekeningen of

simulaties te gebruiken, zoals we later zullen zien.

Als we dan toch niet kunnen weten wat de superpositie is van een qubit, is de Quantumcomputer dan nog steeds krachtiger dan een klassieke computer waar we ook alleen maar een 0 en 1 kunnen uitlezen? Dit is net de moeilijkheid, maar ook de kracht van Quantumcomputers. Het is zeer moeilijk te voorspellen wat de verschillende berekeningen als resultaten zullen geven op verschillenden superposities. Deze superposities gaan in tegen hoe wij op de klassieke manier omgaan met informatie, die we steeds kunnen opvragen. We stellen een klassieke bit voor als een muntstuk, dat ofwel kop ofwel munt kan zijn. Terwijl een qubit zich in een hele hoop combinaties van die twee kan bevinden, totdat we de qubit meten en hij ofwel $|0\rangle$ ofwel $|1\rangle$ zal zijn. Een toestand die vaak voorkomt voor een qubit is de volgende.

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Deze toestand geeft wanneer gemeten, als resultaat met 50 procent kans, de toestand $|0\rangle$ en met 50 procent kans de toestand $|1\rangle$. Omdat deze toestand zo vaak gebruikt wordt, duiden we hem soms aan met het symbool $|+\rangle$.

Hoeveel informatie kunnen we in een qubit opslaan? Een qubit kan zich in oneindig veel verschillende superposities bevinden, dus we zouden verwachten dat we met een qubit dan ook oneindig veel informatie zouden kunnen voorstellen. Dit blijkt echter helemaal niet zo te zijn. Een qubit zal namelijk alleen maar de toestanden $|0\rangle$ en $|1\rangle$ geven, wanneer we gaan meten. Meer nog wanneer we een qubit meten die zich in een superpositie bevindt zal deze moeten "kiezen" om $|0\rangle$ of $|1\rangle$ te zijn bij de meting. Wanneer we dan stoppen met observeren zal de qubit niet terug in een superpositie geraken, maar blijven in de toestand die hij aannam toen hij gemeten werd.
Het lijkt dus dat we uit een qubit niet meer informatie kunnen halen dan we uit een klassieke bit kunnen halen. Dit lijkt te kloppen maar enkel wanneer we de qubit gaan meten. Wanneer we de qubit niet gaan observeren zal hij in een superpositie blijven bestaan. Wanneer we operaties uitvoeren zonder een meting uit te voeren, blijft alle informatie van de amplitudes (α en β) behouden. In deze superposities zit dus een enorme hoeveelheid van verborgen informatie. De grote kracht van Quantumcomputers komt uit het feit dat deze verborgen informatie exponentieel toeneemt wanneer we extra qubits toevoegen. Hoe we moeten omgaan en rekenen met deze verborgen informatie zonder dat we de superpositie van de qubits verstoren, is een van de grote moeilijkheden van Quantum Computing.

4.1 Meerdere qubits

We weten nu wat qubits zijn en hoe we ze voorstellen, we hebben ook beweerd dat de kracht van qubits exponentieel groeit wanneer we meer qubits toevoegen. Hier gaan we nu dieper op in.
Stel dat we een systeem hebben met twee qubits en een systeem met twee gewone bits. Met de gewone bits kunnen we vier verschillende toestanden maken, namelijk 00 , 01 , 10 en 11. Met de qubits kunnen we ook dezelfde combinaties maken, maar we kunnen nu ook verschillende superposities maken met deze vier basistoestanden. We zien dus dat we al snel meer toestanden krijgen wanneer we zelfs maar één qubit toevoegen. De mogelijk superposities waarin twee qubits zich kunnen bevinden zijn de volgende.

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

Analoog met wanneer we maar één qubit hebben, zullen we bij het meten als resultaat een van de vier basistoestanden krijgen met kans $|\alpha_x|^2$ (waarbij x één van de verschillende basistoestanden is). De som van alle kansen zal dan 1 zijn. En net zoals bij een qubit zal na het meten de superpositie niet meer bestaan en zullen de qubits in hun gemeten toestand blijven verder bestaan.
Stel dat we een systeem hebben met n qubits dan zullen de basistoestanden van dit systeem van de vorm $|x_1, x_2, x_3, ..., x_n\rangle$ zijn. Met $x_n \epsilon \{0, 1\}$. Om de superposities van dit systeem te beschrijven zullen we dan 2^n verschillende amplitudes nodig hebben. We zien dus dat de informatie die we kunnen opslaan in de superpositie van een qubit exponentieel toeneemt, wanneer we qubits toevoegen. Het wordt nu ook zeer duidelijk dat wanneer we Quantumtoestanden willen simuleren op een klassieke computer, dit enorm veel geheugen zal eisen om al die amplitudes op te slaan, zelfs wanneer we over zeer kleine toestanden spreken. Het is net deze exponentiële kracht waarvan we graag gebruik willen maken met Quantum Computing.

5 Quantumpoorten

Klassieke computers hebben we voorgesteld met behulp van het circuitmodel. Een circuit bestaat uit een aantal logische poorten, die we verbinden met behulp van draden. Deze zorgen ervoor dat de bits tot bij de poorten geraken, die er dan bewerkingen op doen. We kunnen een Quantumcomputer ook voorstellen met behulp van zo een circuitmodel. We maken hier dan geen gebruik van logische poorten, maar van Quantumpoorten (gates) die de qubits kunnen manipuleren zonder te meten, zodat de superpositie intact blijft. We overlopen nu enkele simpele en vaak gebruikte Quantumpoorten en gaan hier dan later meer ingewikkelde circuits mee bouwen.

5.1 Enkele qubit poorten

In de klassieke computer is de simpelste poort die er bestaat de NOT-poort, die als input een bit krijgt en als output het tegenovergestelde van die bit geeft; dus 0 wordt 1 en 1 wordt 0. We vragen ons af of het mogelijk is om van deze eenvoudige poort een Quantumversie te maken met als input een qubit en als output de tegenovergestelde qubit. We krijgen dan $|0\rangle$ wordt $|1\rangle$ en $|1\rangle$ wordt $|0\rangle$. Dit lijkt niet zo een moeilijke opgave, maar wat te doen met de superpositie waar de qubit zich in een toestand tussen $|0\rangle$en $|1\rangle$ bevindt? Wanneer een qubit zich in een superpositie bevindt, willen we dat we door het toepassen van een NOT het volgende resultaat krijgen:

$$\alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \alpha\,|1\rangle + \beta\,|0\rangle$$

We kunnen een Quantumpoort ook eenvoudig als een matrix noteren en de qubit als een vector. Dit laat toe om er makkelijker mee te rekenen. Voor een Quantum NOT-poort die we aanduiden met X en een qubit met toestand $\alpha\,|0\rangle + \beta\,|1\rangle$ zouden we dan als resultaat krijgen:

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

Wanneer we de NOT-poort dan toepassen op de qubit krijgen we het volgende resultaat:

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

We kunnen Quantumpoorten die op één qubit werken dus schrijven als 2×2 matrices. We vragen ons nu af, of er beperkingen zijn op de matrices die we kunnen gebruiken als Quantumpoorten. Of mogen we zo maar elke 2×2 matrix gebruiken?
Het blijkt dat er maar één voorwaarde is waaraan de matrix van de Quantumpoort moet voldoen. Voor elke qubit geldt $|\alpha|^2 + |\beta|^2 = 1$. We willen

dat deze voorwaarde nog steeds in orde is, wanneer we de Quantumpoort hebben toegepast op de qubit. Met andere woorden, we willen dat na het toepassen van matrix U geldt dat $|\alpha'|^2 + |\beta'|^2 = 1$, waarbij α' en β' de resultaten zijn van het toepassen van de matrix U op de qubit (α, β). Nu blijkt dat deze voorwaarde voldaan is, wanneer matrix U unitair is. Dit wil zeggen dat voor matrix U geldt dat $U^\dagger U = I$. Dit wil zeggen dat het product van de adjointe matrix en de matrix zelf de eenheidsmatrix als resultaat geven. Een adjointe matrix is een getransponeerde matrix, waarin elk getal wordt vervangen door zijn complex toegevoegde. Een vector of matrix behoudt zijn norm als we hem vermenigvuldigen met een unitaire matrix. Dit komt omdat een unitaire matrix een orthogonale matrix is met complexe getallen en een orthogonale matrix behoudt het inwendige product. Daarom geldt ook dat de norm behouden blijft [5]. We verifiëren nu voor de NOT-matrix dat dit een unitaire matrix is.

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

$$U^\dagger = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$U^\dagger U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Het feit dat de enigste vereiste voor een Quantumpoort deze unitaire vereiste is, zorgt ervoor dat in tegenstelling tot de klassieke computer, waar we alleen een NOT poort hebben die op 1 bit werkt, er voor de Quantumcomputer veel meer poorten zijn die op enkel 1 qubit werken. Twee van de vele mogelijke Quantumpoorten die veel gebruikt worden zijn de Z-poort en de Hadamard-poort. De Z-poort wordt gebruikt om de toestand $\alpha|0\rangle + \beta|1\rangle$ om te vormen tot de toestand $\alpha|0\rangle - \beta|1\rangle$. De matrix-notatie van deze poort is de volgende:

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

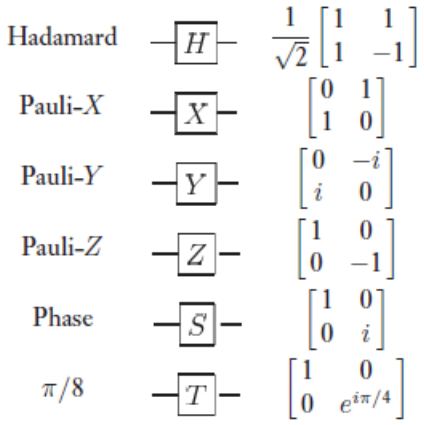
Opnieuw kunnen we makkelijk verifiëren dat dit een unitaire matrix is.

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

De Hadamard-poort heeft als matrix-notatie de volgende:

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

De Hadamard-poort verandert $|0\rangle$ in $(|0\rangle + |1\rangle)/\sqrt{2}$ wat we kunnen bekijken als halverwege tussen $|0\rangle$ en $|1\rangle$. En $|1\rangle$ wordt dan $(|0\rangle - |1\rangle)/\sqrt{2}$ wat ook



Figuur 9: Single Qubit Gates (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

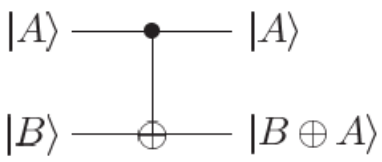
halverwege is tussen $|0\rangle$ en $|1\rangle$.

We geven in figuur 9 kort even de belangrijkste en meest gebruikte poorten weer, met daarbij ook het symbool dat we gebruiken in de circuit-notatie. Deze circuit-notatie laat ons toe om Quantumalgoritmes uit te leggen op een eenvoudige manier, zonder dat we hiervoor moeten gebruik maken van een heel aantal matrices.

Er zijn echter nog veel meer 2×2 unitaire matrices. Dus er zijn ook nog enorm veel meer Quantumpoorten die kunnen werken op een enkele qubit. Het is echter niet nodig om alle mogelijke poorten te zoeken. We kunnen door een aantal poorten te combineren, alle andere poorten simuleren. Om zo een universal set te creëren, hebben we wel niet genoeg aan alleen maar poorten die werken op één qubit. We moeten ook poorten toevoegen die meerdere qubits nodig hebben als input.

5.2 Meerdere qubit poorten

We stappen nu over van een enkele qubit als input, naar poorten die meerdere qubits als input gebruiken. Bij klassieke computers hadden we 5 belangrijke logische poorten die vaak voorkwamen de AND, OR, XOR, NAND en NOR. We weten dat we met een NAND-poort de andere logische poorten kunnen simuleren en hebben daarom alleen deze NAND-poort nodig om elke operatie te kunnen uitvoeren. We noemen de NAND-poort daarom een universele poort.



Figuur 10: CNOT (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

Een vaak gebruikte Quantumpoort is de controlled NOT of de CNOT. Deze poort krijgt als input twee bits die we de control - qubit en de target - qubit noemen. We kunnen de CNOT als de volgende matrix schrijven.

$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We beschrijven nu de functie van de poort voor de basistoestanden. We kunnen vervolgens afleiden hoe de andere toestanden zich zullen gedragen, doordat dit lineaire combinaties zijn van de basistoestanden. De CNOT-poort verandert niks aan de target-qubit als de control-qubit op $|0\rangle$ staat. En zal een NOT uitvoeren op de target-qubit als de control-qubit op $|1\rangle$ staat. We kunnen de CNOT ook bekijken als een variant van de XOR uit de klassieke poorten. De target- en control-bit worden ge-XORed en het resultaat wordt opgeslagen in de target-qubit. We kunnen deze poort ook schematisch weergeven, zie figuur 10, waarbij de bovenste lijn de control-qubit is en de onderste lijn de target-qubit. A en B stellen hierbij een van de basistoestanden voor.

We merken op dat de CNOT een soort van XOR is voor qubits en we hebben ook al een NOT-poort kunnen creëren voor qubits. Is het nu mogelijk om nog poorten om te zetten naar Quantumpoorten zoals de NAND-poort of de gewone XOR? Het blijkt dat dit niet mogelijk is, doordat de NAND- en XOR-poorten, net zoals de meeste andere logische poorten, onomkeerbaar zijn. Hiermee bedoelen we dat wanneer we de output kennen van een poort, het niet mogelijk is om te weten welke de exacte input was van de poort. Dit terwijl unitaire Quantumpoorten altijd omkeerbaar zijn door gebruik te maken van een andere Quantumpoort. Deze inverteerbaarheid van Quantumpoorten is één van de grote krachten die we kunnen gebruiken bij Quantum Computing.

Er bestaan nog heel wat andere poorten dan de CNOT-poort, maar we

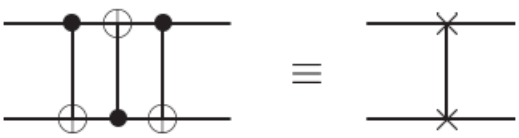
kunnen nu al stellen dat elke poort met meerdere qubits als input, kan worden opgebouwd uit een CNOT-poort, met toevoeging van enkele poorten die één qubit als input vragen. Dit is dus de universal set van de Quantumpoorten. Meer zelfs, we kunnen stellen dat we met behulp van Hadamard, phase, $\pi/8$, en CNOT-poorten alle andere unitaire operaties kunnen simuleren. Er wordt echter niets gezegd over hoe efficiënt zo een simulatie zal zijn en hoeveel van deze poorten we achter elkaar zullen moeten gebruiken. Het is echter het doel van Quantum Computing om zo efficiënt mogelijke oplossingen te vinden. Daarom dat we toch van meer poorten gebruik gaan maken, dan enkel van deze vier.

6 Quantum Circuits

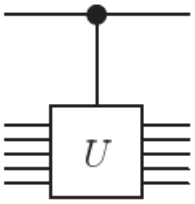
We hebben reeds enkele simpele en vaak gebruikte Quantumpoorten beschreven en uitgelegd hoe we van Quantumpoorten gebruik kunnen maken. We gaan nu verder door deze poorten te combineren in Quantum circuits die we dan voor allerhande toepassingen kunnen gebruiken. Als we verschillende poorten willen combineren, moeten we alleen uitwerken hoe deze combinaties moeten werken op de basistoestanden. Net zoals bij Quantumpoorten kunnen we dan hieruit afleiden hoe deze combinatie van poorten zal werken op een toestand die een lineaire combinatie is van de basistoestanden. We verwachten ook steeds dat de circuits die we hier definiëren als inputs basistoestanden krijgen. Wanneer dit niet zo is geven we dit expliciet aan. We kunnen Quantum circuits bekijken als een model dat analoog is met het circuitmodel van de klassieke computer. We hebben al Quantumpoorten gedefinieerd die dezelfde functie hebben als de logische poorten in het klassieke circuitmodel. We definiëren nu ook wires zoals in het klassieke model, die als functie hebben de qubits door te geven tussen de poorten. In tegenstelling tot het klassieke model, waar de wires ook fysieke draden voorstelden, die elektrische stroom doorgaven en waarmee we dan de bits representeren, kunnen de wires bij een Quantummodel verschillende betekenissen hebben. Een wire kan een echte draad zijn, maar evengoed een verloop van tijd of een fysieke verplaatsing van de qubit doorheen de ruimte bijvoorbeeld als een photon.

We geven nu een eenvoudig voorbeeld van een Quantum circuit, om vertrouwd te raken met de voorstelling van zo een circuit. Het circuit dat we hiervoor gebruiken, krijgt als input twee qubits en zal deze met behulp van drie Quantumpoorten van plaats wisselen. De linkse voorstelling van dit circuit is degene waarin we zien wat er exact gebeurt. Maar omdat dit een circuit is dat zo vaak voorkomt, heeft men er ook een ingekorte versie van gemaakt, die hetzelfde voorstelt. We geven beide voorstellingen weer in figuur 11.

We overlopen nu kort wat er precies gebeurt in dit circuit. Stel dat we aan dit circuit een basistoestand over twee qubits meegeven, die we voor-



Figuur 11: Wissel circuit (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)



Figuur 12: Control U poort (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

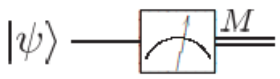
stellen als $|a, b\rangle$. Waarbij $a, b \in \{0, 1\}$.

$$\begin{aligned} |a, b\rangle &\rightarrow |a, a \oplus b\rangle \\ &\rightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle \\ &\rightarrow |b, (a \oplus b) \oplus b\rangle = |b, a\rangle \end{aligned}$$

We zien dat we als resultaat dezelfde qubits krijgen, ze zijn echter van plaats verandert. De qubit a komt dus binnen op de bovenste wire en verlaat het circuit op de onderste wire en voor qubit b gebeurt het omgekeerde.

Omdat we onmogelijk alle Quantum circuits kunnen overlopen, gaan we ze pas introduceren wanneer we ze nodig hebben. Toch willen we nog twee belangrijke elementen introduceren die vaak voorkomen in Quantum circuits. De eerste daarvan is de controlled-poort, waarvan we al een voorbeeld hebben gezien met de CNOT-poort. Een controlled-poort werkt net hetzelfde als een gewone poort, maar we voegen een extra qubit toe als input. Deze extra input-qubit noemen we dan de control-qubit. Als de control als waarde $|0\rangle$ heeft dan zal de poort geen effect hebben op de andere input-qubits. Wanneer de control-qubit echter als waarde $|1\rangle$ heeft, zal de poort de verwachte operatie uitvoeren op de input-qubits. Een control-poort stellen we schematisch voor als de gewone poort, waarboven we een extra wire plaatsen die de control-qubit verbindt met de poort. Voor een willekeurige poort U krijgen we dan het resultaat uit figuur 12.

Een tweede speciale poort die we vaak gebruiken is de meter-poort. Zie figuur 13. Deze poort verandert de input-qubit in een klassieke bit. Voor input $\alpha|0\rangle + \beta|1\rangle$ betekent dit dan dat we als output een bit krijgen met waarde 0, met een kans van $|\alpha|^2$ en een output bit 1 met een kans van $|\beta|^2$, waarvoor geldt dat de som van deze kansen als resultaat 1 geeft. Om verwarring te vermijden stellen we de klassieke output bit voor met behulp van een dubbele wire. We moeten echter oppassen wanneer we gebruik maken van een meting in een Quantum circuit. Want zodra we een qubit gaan meten zal hij niet meer in een superpositie zijn en vernietigen we alle verborgen



Figuur 13: Meter (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

Quantuminformatie die de qubit bezat. Deze meting is een onomkeerbare operatie. Wanneer we over meerdere qubits beschikken is het mogelijk om slechts een subset van deze qubits te meten. Wat gebeurt er dan met de amplitudes van de overgebleven qubits, wanneer we slechts een deel van de qubits meten? We proberen dit duidelijk te maken aan de hand van een voorbeeld met twee qubits. De toestand waarin twee qubits zich kunnen bevinden kunnen we voorstellen als volgend:

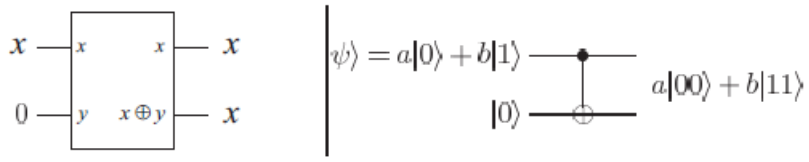
$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

Wanneer we nu de eerste qubit gaan meten zullen we als resultaat $|0\rangle$ krijgen met kans $|\alpha_{00}|^2 + |\alpha_{01}|^2$. De overgebleven qubit bevindt zich dan in de volgende toestand:

$$|\psi'\rangle = \frac{\alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

We merken hier op dat de toestand na het meten gedeeld wordt door de factor $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$. Dit doen we om ervoor te zorgen dat de norm van deze nieuwe toestand opnieuw 1 zal zijn, zoals we verwachten van een Quantumtoestand.

Buiten de wires zijn er in Quantum circuits nog een aantal elementen die we niet direct kunnen overnemen en zelfs elementen die onmogelijk zijn om over te nemen.
Het eerste wat niet toegelaten is in Quantum circuits, is het maken van een loop; waarbij we het resultaat van een Quantumpoort terugsturen naar een vorige positie in het circuit. Dit mocht wel in een klassiek circuit. Twee andere operaties die vaak voorkomen in klassieke circuits en die niet bestaan in Quantum circuits, zijn Fan In en Fan Out. Bij Fan In laten we verschillende wires samen komen in een wire. De bit op deze nieuwe wire krijgt dan de waarde van een OR-operatie op alle wires die samenkomen. Bij Fan Out doen we het omgekeerde en laten we uit een wire meerdere wires vertrekken en kopiëren de bit op deze wire naar alle nieuwe wires. We zullen zien dat dit voor qubits onmogelijk is omdat de wetten van de Quantummechanica het kopiëren van informatie niet toelaat. We proberen dit nu duidelijk te maken aan de hand van een voorbeeld, waarin we gaan proberen een qubit te kopiëren. We maken hiervoor gebruik van het circuit uit figuur 14 wat



Figuur 14: Copy circuit. Links met klassieke bits rechts met qubits (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

bestaat uit een CNOT-poort. Wanneer we zo een CNOT-poort klassieke bits als input geven, kunnen we deze poort gebruiken om bits te kopiëren. We doen dit door de bit die we willen kopiëren modulo 2 op te tellen bij een bit die we op 0 laten starten. Ook wanneer we dit circuit uitvoeren met de basistoestanden $|0\rangle$ en $|1\rangle$, merken we dat we de qubits kunnen kopiëren. We verwachten dat dit nu ook geldt voor de lineaire combinaties van de basistoestanden, maar dit lijkt niet zo te zijn.

Wanneer we een willekeurige qubit $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ willen kopiëren geven we deze samen met een qubit $|0\rangle$ als input aan de CNOT-poort. De begintoestand kunnen we dan ook schrijven als:

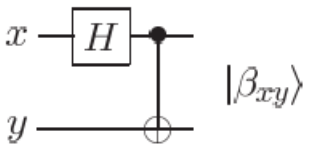
$$[a\,|0\rangle + b\,|1\rangle]\,|0\rangle = a\,|00\rangle + b\,|10\rangle$$

Als we deze bits als input gebruiken voor de CNOT poort krijgen we als resultaat de toestand $|\psi\rangle = a\,|00\rangle + |11\rangle$. Is het kopiëren nu succesvol geweest? Met andere woorden is deze toestand de toestand $|\psi\rangle\,|\psi\rangle$. In het geval dat $|\psi\rangle = |0\rangle$ of $|\psi\rangle = |1\rangle$ lijkt dit te kloppen. Maar voor een algemene toestand $|\psi\rangle$ weten we dat de toestand $|\psi\rangle\,|\psi\rangle$, waarmee we een tensorproduct tussen deze twee toestanden uitdrukken, er als volgt uitziet:

$$|\psi\rangle\,|\psi\rangle = a^2\,|00\rangle + ab\,|01\rangle + ab\,|10\rangle + b^2\,|11\rangle$$

We zien dus dat het kopiëren van de qubit gelukt is, als $ab = 0$. Maar als dit zo is, moet a of b gelijk zijn aan 0 wat ons terug in de basistoestand brengt. Het is dus niet mogelijk een qubit in een willekeurige toestand te kopiëren alleen de basistoestanden.
We kunnen dit ook begrijpen, door na te denken over de geheime informatie die verborgen zit in de amplitudes van de superpositie waarin een qubit zich kan bevinden. Zoals eerder gezegd gaat deze verborgen informatie verloren wanneer een qubit gemeten wordt. Stel dat we twee qubits hebben die zich in de volgende superpositie bevinden $a\,|00\rangle + b\,|11\rangle$. De tweede qubit is hier een kopie van de eerste. Wanneer we één van deze qubits gaan meten, krijgen we als resultaat 0 of 1 met de kansen $|a|^2$ en $|b|^2$. Dit is echter nog niet alles. Wanneer we één van de qubits meten, weten we ook meteen de

In	Out
$\|00\rangle$	$(\|00\rangle + \|11\rangle)/\sqrt{2} \equiv \|\beta_{00}\rangle$
$\|01\rangle$	$(\|01\rangle + \|10\rangle)/\sqrt{2} \equiv \|\beta_{01}\rangle$
$\|10\rangle$	$(\|00\rangle - \|11\rangle)/\sqrt{2} \equiv \|\beta_{10}\rangle$
$\|11\rangle$	$(\|01\rangle - \|10\rangle)/\sqrt{2} \equiv \|\beta_{11}\rangle$



$|\beta_{xy}\rangle$

Figuur 15: Bell toestand circuit (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

waarde van de tweede qubit en gaat de informatie die verborgen zat in de amplitudes a en b verloren. Want de overgebleven qubit krijgt als toestand $|\psi'\rangle = \frac{a|00\rangle}{\sqrt{|a|^2}} = |00\rangle$ als de eerste gemeten qubit $|0\rangle$was. Stel nu dat het mogelijk was om qubits te kopiëren. Dan zouden we nog de verborgen informatie uit a en b kunnen aanspreken in de tweede qubit. Daarom besluiten we dat het niet mogelijk is om een willekeurige qubit te kopiëren.

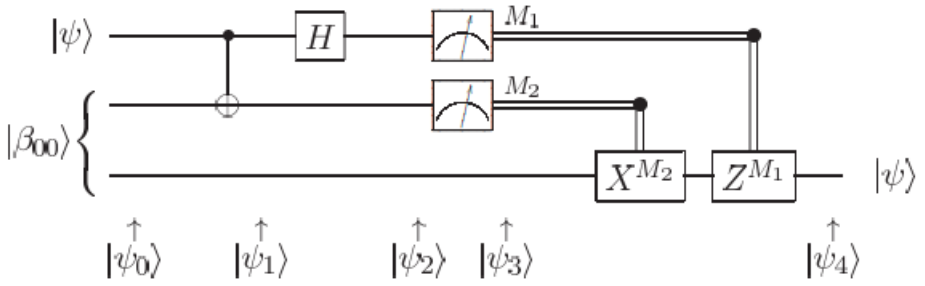
6.1 Quantum teleportation

We hebben enkele simpele poorten en circuits bekeken en gaan deze nu gebruiken om een eenvoudig circuit te bouwen. Dit circuit gaan we gebruiken om een zeer eigenaardige en speciale eigenschap van de Quantummechanica mee te laten zien, namelijk Quantumteleportatie. Voor dit circuit moeten we over qubits beschikken, die zich in een Bell-toestand bevinden. De Bell-toestand is een bijzondere toestand waarin twee qubits zich kunnen bevinden. Hiervoor geldt dat als we het eerste qubit meten, we ook onmiddellijk weten wat het tweede qubit is, dit noemen we entanglement. We kunnen een circuit maken dat twee bits kan bewerken, zodat ze na het circuit zich in een Bell-toestand bevinden. Hiervoor maken we gebruik van een Hadamard-poort gevolgd door een CNOT. In figuur 15 geven we een schema van dit circuit weer met daarbij een waarheidstabel die aangeeft wat de resultaten zijn voor de verschillende mogelijke inputs. We kunnen het resultaat ook als een algemene formule noteren.

$$|\beta_{xy}\rangle \equiv \frac{|0, y\rangle + (-1)^x |1, \bar{y}\rangle}{\sqrt{2}}$$

Waarbij x en y de verschillende combinaties weergeven met waardes 0 of 1.

Nu we de mogelijkheid hebben om qubits in een Bell-toestand te plaatsen, kunnen we een circuit maken waarmee we Quantumteleportatie kunnen



Figuur 16: Quantum teleportation (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

doen. Quantumteleportatie is geen systeem dat een probleem oplost, maar kan gebruikt worden om Quantumtoestanden door te geven aan andere circuits, zonder dat hier een verbinding voor nodig is tussen deze circuits.
Net als bij gewone communicatie tussen klassieke computers gebruiken we hiervoor een voorbeeld met twee mensen Alice en Bob. Alice en Bob hebben samen twee qubits die zich in een Bell-toestand bevinden. Ze nemen elk één van de qubits en gaan naar plaatsen die ver uit elkaar liggen. De enige manier die Alice en Bob nog hebben om met elkaar te communiceren is via een klassieke manier. Er kunnen dus geen qubits doorgestuurd worden. We gaan nu een circuit bouwen dat Alice zal helpen om toch een willekeurige qubit $|\psi\rangle$ door te sturen naar Bob.
De toestand van de qubit die Alice moet doorsturen is voor haar onbekend. Ze kan deze ook niet meten, want dan bestaat de qubit niet meer zoals eerder gezegd. We kunnen echter de entanglement eigenschap van de qubits die zich in een Bell-toestand bevinden, gebruiken om toch de qubit door te geven met behulp van wat klassieke bits communicatie.
We gaan als volgt te werk. Alice heeft een circuit dat als input een qubit $|\psi\rangle$ krijgt, waarvan we weten dat $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ en alpha en beta zijn onbekende amplitudes. De tweede input waarover Alice beschikt, is de qubit die zij heeft uit het qubitpaar dat zich in een Bell-toestand bevindt. Op deze twee inputs voert Alice achtereenvolgens een CNOT- en een Hadamardpoort uit. Hierna meet ze haar qubits en stuurt deze klassieke informatie door naar Bob. Wanneer Bob deze bits krijgt, past hij op zijn deel van de Bell-toestand een CNOT- en een controlled Z-poort toe, waarbij de bits die hij van Alice heeft gekregen gebruikt worden als controle-bits. Als resultaat krijgt Bob exact de qubit $|\psi\rangle$ die Alice wou doorsturen.

In het schema van figuur 16 zijn de bovenste twee wires het systeem dat Alice gebruikt. De onderste is het systeem dat Bob gebruikt. De dubbele lijnen geven de communicatie van de klassieke bits weer tussen Alice en Bob.

We gaan nu laten zien dat dit circuit ook inderdaad echt de qubit $|\psi\rangle$ doorgeeft. We volgen het circuit in de veronderstelling dat de Bell-toestand die gebruikt wordt, is verkregen door $|00\rangle$ als input te gebruiken in het Bell-toestand circuit. In het begin van het teleportatie-circuit zal de input $|\psi_0\rangle$ te beschrijven zijn als volgt:

$$\begin{aligned} |\psi_0\rangle &= |\psi\rangle\,|\beta_{00}\rangle \\ &= \frac{1}{\sqrt{2}}\left[\alpha\,|0\rangle\,(|00\rangle + |11\rangle) + \beta\,|1\rangle\,(|00\rangle + |11\rangle)\right] \end{aligned}$$

Waarbij we afspreken dat de door te sturen qubit en de eerste helft van de Bell-toestand in het bezit zijn van Alice en Bob de tweede helft van de Bell-toestand in zijn bezit heeft. Alice stuurt haar twee qubits nu door een CNOT- en een Hadamard-poort. De toestand waarin de qubits zich dan bevinden kunnen we dan uitdrukken in de toestanden $|\psi_1\rangle$ en $|\psi_2\rangle$.

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\left[\alpha\,|0\rangle\,(|00\rangle + |11\rangle) + \beta\,|1\rangle\,(|10\rangle + |01\rangle)\right]$$

$$|\psi_2\rangle = \frac{1}{2}\left[\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)\right]$$

Als we de termen groeperen krijgen we

$$\begin{aligned} |\psi_2\rangle = \frac{1}{2}[&|00\rangle\,(\alpha\,|0\rangle + \beta\,|1\rangle) + |01\rangle\,(\alpha\,|1\rangle + \beta\,|0\rangle) \\ &+ |10\rangle\,(\alpha\,|0\rangle - \beta\,|1\rangle) + |11\rangle\,(\alpha\,|1\rangle - \beta\,|0\rangle)] \end{aligned}$$

Deze som bevat vier termen. Eén term voor elke mogelijke combinatie van bits die Alice kan krijgen op het einde van haar circuit, de rest van de term is dan de toestand waarin de qubit van Bob zich bevindt. We kunnen dit als volgt schrijven zodat het duidelijk wordt in welke toestand de qubit van Bob zich bevindt, afhankelijk van de meetresultaten die Alice doorstuurt.

$$\begin{aligned} 00 \longmapsto |\psi_3(00)\rangle &\equiv [\alpha\,|0\rangle + \beta\,|1\rangle] \\ 01 \longmapsto |\psi_3(01)\rangle &\equiv [\alpha\,|1\rangle + \beta\,|0\rangle] \\ 10 \longmapsto |\psi_3(10)\rangle &\equiv [\alpha\,|0\rangle - \beta\,|1\rangle] \\ 11 \longmapsto |\psi_3(11)\rangle &\equiv [\alpha\,|1\rangle - \beta\,|0\rangle] \end{aligned}$$

Afhankelijk van de meting die Alice doet, zal de qubit van Bob zich in een van deze toestanden bevinden, zelfs nog voordat Bob de informatie van Alice krijgt. Het lijkt dus alsof we informatie kunnen versturen sneller dan een

elektrisch signaal of de snelheid van het licht. Maar omdat Bob nooit kan weten in welke van deze vier toestanden de qubit zich bevindt, zonder eerst de resultaten van Alice te krijgen, is deze illusie dat we sneller als het licht kunnen communiceren snel doorbroken. Wanneer Bob de resultaten van Alice krijgt, kan hij met behulp van zijn circuit zijn qubit aanpassen zodat deze in exact dezelfde toestand komt als de originele input $|\psi\rangle$. Bob gebruikt dus de bits die hij van Alice als controlebits voor zijn X- en Z- poort krijgt, om zo te bepalen welke nodig zijn om de toestand van zijn qubit om te vormen naar de toestand van de originele qubit.
Er zijn vier mogelijke berichten die Bob kan krijgen van Alice, we overlopen deze hier en duiden aan welke actie Bob onderneemt om zijn qubit te verbeteren.

1. Bob krijgt 00. Zijn qubit is dan in toestand $\alpha|0\rangle + \beta|1\rangle$. Dit is de toestand die we willen dus er moet niets ondernomen worden.

2. Bob krijgt 01. Zijn qubit is dan in toestand $\alpha|1\rangle + \beta|0\rangle$. We maken gebruik van een X-poort (NOT-poort) om de qubit in de gewenste toestand te krijgen.

3. Bob krijgt 10. Zijn qubit is dan in toestand $\alpha|0\rangle - \beta|1\rangle$. We maken gebruik van een Z-poort om de qubit in de gewenste toestand te krijgen.

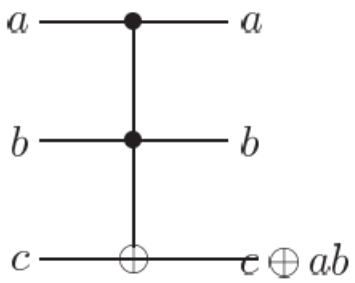
4. Bob krijgt 11. Zijn qubit is dan in toestand $\alpha|1\rangle - \beta|0\rangle$. We maken gebruik van een X-poort gevolgd door een Z-poort om de qubit in de gewenste toestand te krijgen.

Belangrijk om op te merken is dat er geen kopie wordt verstuurd van de qubit wanneer we deze naar Bob willen sturen. De originele qubit gaat verloren wanneer we deze meten om door te sturen naar Bob. We recreëren dus als het ware de superpositie waarin deze qubit zich bevond. Is het dan wel handig om zo veel werk uit te voeren om een qubit door te sturen? Het antwoord hierop is 'Ja'. We hebben al uitgelegd, dat de mogelijke informatie die opgeslagen is in de verborgen amplitudes van een qubit, enorm kan zijn. We hebben nu een manier om deze grote hoeveelheden informatie door te sturen over grote afstanden, met behulp van maar twee bits die effectief moeten worden doorgestuurd.

6.2 Klassieke algoritmes op Quantum circuits

We weten niet zeker of Quantumcomputers over meer rekenkracht beschikken dan klassieke computers, hoewel we een zeer sterk vermoeden hebben dat dit wel zo is. Toch blijft het moeilijk om dit te bewijzen. We kunnen

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



Figuur 17: Toffoli-poort (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

wel bewijzen dat Quantumcomputers minstens even krachtig zijn als klassieke computers door met behulp van Quantum circuits klassieke circuits te simuleren. Dit is mogelijk omdat de klassieke computers gebaseerd zijn op elementen uit de fysica. En de fysica is nu eenmaal gebaseerd op Quantummechanica. Dus het zou het zeer tegenstrijdig zijn, moest een Quantumcomputer geen klassieke circuits kunnen simuleren. Toch hebben we eerder uitgelegd dat er een aantal elementen uit de klassieke circuits zijn, die niet mogelijk zijn in Quantum circuits. Zoals poorten die niet omkeerbaar zijn, fan in en fan out, ...
Dit lossen we op door de poorten die niet omkeerbaar zijn, te vervangen door poorten die dat wel zijn, maar hetzelfde resultaat geven. We maken hier gebruik van een poort die we de Toffoli-poort noemen. De matrix-notatie van de Toffoli-poort lijkt op deze van een 8×8 identiteitsmatrix, waarvan de laatste twee kolommen van plaats gewisseld zijn. De Toffoli-poort krijgt als input 3 qubits, waarvan de eerste twee controle bits zijn, die niet veranderen door de actie van de poort. De derde qubit, de target-qubit wordt geflipt, wanneer de twee controle bits 1 zijn. We geven de werking van de Toffoli-poort weer in de waarheidstabel in figuur 17.

Het is niet nodig om te bewijzen dat we met deze Toffoli-poort alle logische poorten van een klassiek circuit kunnen simuleren. Het is genoeg wanneer we bewezen hebben dat we een NAND-poort kunnen simuleren. Dit komt, omdat zoals eerder gezegd, de NAND-poort de universele poort is voor de klassieke circuitmodellen. En we genoeg hebben aan een NAND-poort alleen, om voor alle algoritmes een circuit te construeren. Ook is het mogelijk om een FANOUT te realiseren met behulp van een Toffoli-poort. We kunnen dus een klassieke deterministische computer simuleren met een Quantumcomputer. We kunnen zelfs een stap verder gaan en niet determi-

nistische computers simuleren op een eenvoudige manier. Het enige wat we hiervoor moeten kunnen, is het simuleren van het random creëren van bits. Dit is zeer makkelijk voor een Quantumcomputer. We nemen gewoon een qubit dat zich in toestand $|0\rangle$ bevindt en sturen dit door een Hadamard-poort. Als resultaat krijgen we dan een qubit in toestand $(|0\rangle + |1\rangle)/\sqrt{2}$. Wanneer we deze qubit gaan meten, zullen we dan ofwel 0 ofwel 1 krijgen met beiden een kans van $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. We hebben dus succesvol een random coin toss gesimuleerd, wat genoeg blijkt te zijn om niet deterministische computers te simuleren met een Quantumcomputer.

6.3 Quantum parallellisme

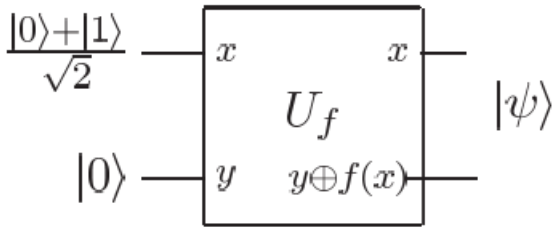
Een volgende eigenaardige eigenschap van Quantumcomputer die we vaak gebruiken in Quantumalgoritmes en die vaak wordt gezien als de grote kracht van Quantumcomputers is parallellisme. Vaak wordt parallellisme te eenvoudig voorgesteld, waardoor mensen een verkeerd beeld krijgen van de kracht van Quantumcomputers. Parallellisme wordt dan voorgesteld als de mogelijkheid van een Quantumcomputer om een functie $f(x)$ voor veel (alle) verschillende waardes van x te evalueren en dit op hetzelfde moment.

In dit deel willen we daarom uitleggen hoe parallellisme wel werkt en wat de beperkingen ervan zijn.
Stel dat we een functie $f(x) : \{0,1\}^n \to \{0,1\}^n$ hebben, we gaan deze functie nu evalueren met behulp van een Quantumcomputer. We maken hiervoor gebruik van een Quantumcomputer die als input twee qubits krijgt, die zich in een basistoestand $|x, y\rangle$ bevinden. De Quantumcomputer voert nu enkele operaties uit en geeft als input twee qubits, die zich nu in de toestand $|x, y \oplus f(x)\rangle$ bevinden, waarbij de tweede qubit een modulo 2 som is van y en $f(x)$. Wanneer $y = 0$ is dan zal de tweede qubit gelijk zijn aan $f(x)$. Stel dat we beschikken over een Quantumcomputer die deze bewerking kan maken die we voorstellen als een black box U_f.

Wanneer we zoals op afbeelding 18 als input de eerste qubit in de toestand $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ gebruiken. Die we bekomen door een Hadamard-poort toe te passen op $|0\rangle$. En als tweede de qubit $|0\rangle$ gebruiken, dan krijgen we als resultaat het volgende:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

Het lijkt nu dat we in deze toestand de functie $f(x)$ tegelijk hebben geëvalueerd voor de waardes 0 en 1. Dit noemen we Quantumparallellisme. We moeten maar één circuit gebruiken om de functie $f(x)$ te evalueren voor verschillende waardes van x. We kunnen deze techniek ook makkelijk uitbreiden



Figuur 18: Black Box U_f (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

naar functie met meer mogelijke waardes voor x en ook meerdere waardes tegelijk evalueren. We maken hiervoor gebruik van een Hadamard transform, dat neerkomt op het achter elkaar gebruiken van een aantal Hadamard-poorten op n bits. We noteren deze opvolging van Hadamard-poorten voor het geval $n = 2$ als volgt $H^{\otimes 2}$. Als output van een Hadamard transform op n bits krijgen we

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle$$

Wanneer we deze qubits geproduceerd door de Hadamard transform dan als input geven voor onze blackbox U_f met nog een extra $|0\rangle$ als laatste qubit, krijgen we als resultaat alle mogelijke evaluaties van $f(x)$ voor het aantal qubits dat we meegeven.

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle\,|f(x)\rangle$$

We zien dus dat we met behulp van Quantumparallellisme een functie $f(x)$ simultaan kunnen evalueren voor verschillende waardes van x. Toch zeiden we in het begin van dit deel dat dit niet klopt en een te simpele voorstelling is. Dit komt doordat we de evaluatie voor verschillende waardes van x niet direct kunnen gebruiken. Wanneer we de qubits gaan meten om de waarde van $f(x)$ te weten te komen, gaat deze qubit telkens maar een van de verschillende evaluaties van $f(x)$ teruggeven volgens de amplitude die deze had in de superpositie. De andere evaluaties van $f(x)$ gaan verloren, omdat de qubit na het meten de toestand aanhoudt die ze als resultaat van de meting gaf. In ons voorbeeld komen we als resultaat de toestand

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

Wanneer we de toestand gaan meten om de waardes van $f(0)$ en $f(1)$ te weten te komen, krijgen we ofwel $|0, f(0)\rangle$ met kans $\frac{1}{2}$ ofwel $|1, f(1)\rangle$ met

kans $\frac{1}{2}$. We komen uiteindelijk maar één resultaat van de functie $f(x)$ te weten.
De moeilijkheid om gebruik te maken van de kracht van parallellisme is dus een manier te vinden om de evaluaties die verborgen zitten in de superpositie te gebruiken, zonder deze superpositie te laten instorten door een meting. Dit is een van de grote uitdagingen wanneer men een Quantumalgoritme ontwikkelt.

7 Quantumalgoritmes

Quantummechanica is een zeer jonge theorie in de ogen van de wetenschap. En de informatica, zoals wij ze kennen met Turing als één van de grondleggers is zelfs nog jonger. In vergelijking met andere, reeds gevestigde waarden in de wetenschappelijke wereld verschijnt Quantum Computing nog maar net op het toneel.

Het begin van Quantumcomputers wordt vaak toegeschreven aan David Deutsch, die zich in 1985 afvroeg of het mogelijk was een computer te maken die men kon gebruiken om eender welk fysisch systeem te simuleren. Omdat elk fysisch systeem uiteindelijk onderhevig is aan de wetten van de Quantummechanica kwam Deutsch uiteindelijk met het idee om een computer te maken gebaseerd op de Quantummechanica. Deutsch definieerde zijn Quantumcomputer model analoog aan dat van een Turingmachine. Of dit een goed model is en of er misschien nog betere modellen zijn weten we nog niet. Wel heeft Deutsch al laten zien dat we met behulp van Quantumcomputers waarschijnlijk problemen efficiënt kunnen gaan oplossen die op een klassieke Turingmachine niet efficiënt oplosbaar zijn. Hij toonde zo aan dat de rekenkracht van een Quantumcomputer waarschijnlijk groter is dan deze van een klassieke computer.
Een tweede grote doorbraak op het vlak van Quantumcomputers kwam er in 1994 toen Peter Shor aantoonde dat twee belangrijke problemen waarvan we denken dat ze niet efficiënt oplosbaar zijn op een klassieke computer, wel oplosbaar zijn op een Quantumcomputer, namelijk factoring en discrete logarithm. Dit bracht veel aandacht met zich mee, omdat men hierdoor een sterk vermoeden krijgt dat Quantumcomputers over meer rekenkracht beschikken dan Turingmachines en zelfs probabilistische Turingmachines.
Niet veel later in 1995 kwam er opnieuw een bewijs voor de rekenkracht van Quantumcomputers, toen Lov Grover zijn algoritme ontdekte waarmee men sneller kan zoeken in een ongeordende zoekruimte. Hoewel de snelheidswinst bij Grover niet zo spectaculair was als bij Shor, was hier toch ook veel aandacht voor, vooral omdat de toepassingen van sneller kunnen zoeken enorm zijn.
Niet alleen het aantonen van de mogelijke snelheidswinsten die we kunnen maken bij het oplossen van problemen zorgden voor een vergrote interesse naar Quantumcomputers, ook de wetenschap was geïnteresseerd om een Quantumcomputer te maken om zo Quantumsystemen efficiënt te kunnen simuleren. Dit vraagt momenteel veel tijd en geheugenruimte van klassieke computers, zelfs voor heel simpele en kleine simulaties.

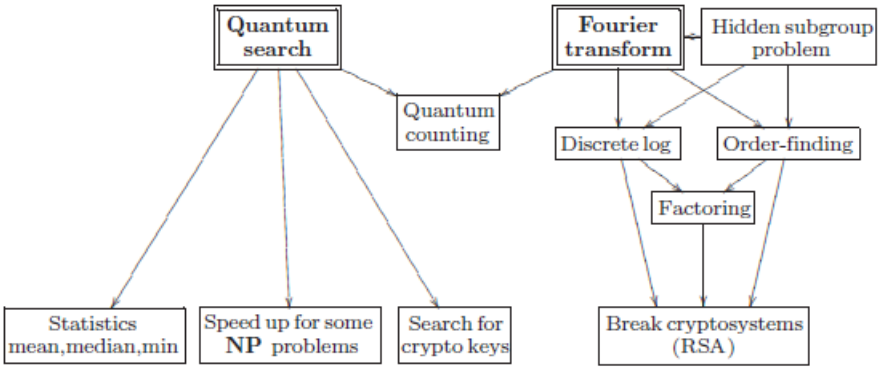
We veronderstellen dat er nog veel andere problemen bestaan die we met behulp van de Quantumcomputer efficiënt kunnen oplossen, maar we weten nog niet precies hoeveel en dewelke. Dit komt doordat Quantum Computing

nog zo nieuw is en er nog zo weinig over bekend is. Alsook omdat het zeer moeilijk is om een goed algoritme te bedenken, dat op een Quantumcomputer werkt. En dit nog in meerder mate dan dat het al moeilijk is om een goed en efficiënt algoritme te vinden op een klassieke computer. Dit om verschillende redenen. Ten eerste is het zeer moeilijk om na te denken in de wereld van Quantummechanica, zeker voor ons, omdat we gewoon zijn na te denken over problemen in functie van de klassieke manier van problemen op lossen. Ten tweede is het niet genoeg om alleen maar een Quantumalgoritme te vinden. Het algoritme moet ook sneller werken dan een algoritme op een klassieke computer, dat hetzelfde probleem oplost. Anders is het niet interessant om zoveel moeite te doen om Quantum Computing te gebruiken en uiteindelijk geen snelheidswinst te maken.

We weten dus niet hoeveel problemen we efficiënter met een Quantumcomputer zullen kunnen oplossen en hoeveel algoritmes we zullen vinden. Wel kunnen we aan de hand van de Quantumalgoritmes die we al kennen zoals Shor en Grover, al een beetje nadenken over de groepen van problemen waar we mogelijk een oplossing voor kunnen vinden, doordat we al een oplossing hebben voor gelijkaardige problemen. We kunnen deze mogelijke algoritmes voorlopig opdelen in drie grote groepen. Twee van deze groepen zijn voor ons als informatici zeer interessant, omdat ze helpen problemen efficiënter op te lossen. De derde groep gaat over simulaties en is voor informatici niet zo handig, maar des te meer voor degene die bezig zijn met Quantummechanica of afgeleiden daarvan, zoals Quantumchemie. In figuur 19 plaatsen we de tot nu toe bekende en te verwachten algoritmes in een schema, waarin te zien is tot welke groep van technieken deze algoritmes horen.

7.1 Quantumalgoritmes gebaseerd op de Fouriertransformatie

De eerste groep van algoritmes die we zouden kunnen vinden voor een Quantumcomputer baseren hun oplossing op de Fouriertransformatie. De Fouriertransformatie is een transformatie die veel gebruikt wordt in verschillende vakgebieden, ook in de informatica komen we deze transformatie vaak tegen. De Fouriertransformatie wordt vooral gebruikt omdat de getransformeerde versie van een probleem makkelijker op te lossen is dan het origineel.
De belangrijkste eigenschappen van de Fouriertransformatie voor ons zijn dat ze omkeerbaar is en dat wanneer we ze als een matrix-operatie schrijven, kunnen zien dat deze een unitaire matrix is. Dit betekent dus dat het mogelijk is om er een Quantum circuit voor te maken. Het belangrijkste reeds bekende Quantumalgoritme dat gebruik maakt van de Fouriertransformatie is het algoritme van Shor.



Figuur 19: Quantumalgoritmes (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

De snelheidswinst die we kunnen maken bij algoritmes met een Fouriertransformatie is enorm. Zo zal een klassieke computer de Fouriertransformatie met behulp van FFT (Fast Fourier Trasnform) $N \log(N) = n2^n$ stappen gebruiken om $N = 2^n$ getallen te transformeren. Wanneer we hetzelfde aantal getallen op een Quantumcomputer zouden transformeren hebben we maar $\log^2(N) = n^2$ stappen nodig [18]. Dit is een exponentiële winst, dit is een enorme tijdswinst. Hoe deze Fouriertransformatie precies gebeurt leggen we later uit.
We besluiten dus dat een Quantumcomputer exponentieel sneller een Fouriertransformatie kan uitvoeren, dus verwachten we dat we een Quantumcomputer zouden kunnen gebruiken om een hele hoop problemen die hier op gebaseerd zijn nu exponentieel sneller kunnen oplossen. Maar hier zit net het probleem van Quantumcomputers. Hoewel de computer zeer snel een Fouriertransformatie kan uitvoeren is het zeer ingewikkeld om deze transformatie verder te gebruiken, doordat het Quantumgedrag van de computer ervoor zorgt dat we niet direct naar het resultaat kunnen kijken. denk aan het elektron dat gemeten wordt bij het dubbele spleet experiment. We moeten dus manieren en algoritmes vinden om deze Fouriertransformaties te gebruiken, zonder dat we de Quantumtoestand van de computer schaden. Dit is net de moeilijkheid van dit soort algoritmes. Een algoritme wat hier in geslaagd is, is het algoritme van Shor om factoring op te lossen. Niet alleen factoring kunnen we oplossen met behulp van een Quantumcomputer, ook het discreet logaritme probleem kunnen we nu efficiënt oplossen. Doordat deze problemen oplosbaar zijn voor een Quantumcomputer is een Quantumcomputer ook in staat om verschillende systemen van cryptografie te kraken zoals het vaak gebruikte RSA.

7.2 Quantum zoekalgoritmes

Een tweede groep algoritmes die zeer verschillend is van de vorige groep zijn de Quantum zoekalgoritmes, waaronder ook het zoekalgoritme van Grover valt. De zoekalgoritmes zijn makkelijk uit te leggen als de oplossing voor het volgende probleem. Stel dat we een aantal elementen N hebben waaruit we er een willen zoeken met een bepaalde eigenschap. We nemen aan dat we niets weten over de verzameling van elementen, zoals het feit of ze al dan niet geordend zijn. Op een klassieke computer duurt zo een operatie over het algemeen N stappen, op een Quantumcomputer kunnen we dit versnellen tot $\sqrt{N}$ [18]. Het huidige bekende algoritme van Grover is zo een zoekalgoritme ook hier komen we later nog op terug.
Dit is niet zo een indrukwekkende snelheidswinst als de exponentiële winst van hierboven, maar zoekalgoritmes hebben een bredere toepassingsbasis. Daarbij maakt een zoekalgoritme vaak deel uit van een ander algoritme, dat ook profiteert van deze snelheidswinst. Daarom vinden we deze snelheidswinst toch zeer interessant, omdat we er gebruik van kunnen maken bij het zoeken naar efficiëntere oplossingen voor problemen uit NP. Vooral de problemen waarbij de huidige oplossing bestaat uit het doorzoeken van een set van mogelijke oplossingen. We komen later nog terug op toepassingen van Quantum zoekalgoritmes wanneer we het over het algoritme van Grover hebben.

7.3 Quantumsimulatie

Een derde grote toepassing waarvoor we Quantumcomputers zouden kunnen gebruiken is het simuleren van fysische systemen, die voorkomen in de natuur. Zoals we ondertussen al weten zijn fysische systemen ook Quantummechanische systemen omdat de fysica nu eenmaal gebaseerd is op de Quantummechanica. Het is ook mogelijk om zulke systemen te simuleren op een klassieke computer, maar dit is zeer moeilijk en vraagt enorm veel rekenkracht en tijd. Moest het wel eenvoudig zijn om zo een Quantummechanisch systeem te simuleren op een klassieke computer, dan zouden we ook eenvoudig een Quantumcomputer kunnen simuleren op een klassieke computer en zou een Quantumcomputer dus geen winst zijn op rekenkracht. Waarom is het zo moeilijk voor een klassieke computer om een Quantumsysteem te simuleren? Dit komt doordat het aantal complexe getallen dat een computer moet bijhouden om een Quantumsysteem te simuleren exponentieel toeneemt wanneer we de grootte van het systeem laten toenemen. Terwijl een Quantumcomputer hiervoor enkel een lineaire groei zal nodig hebben.

Er is echter een probleem met het gebruiken van Quantumcomputers om simulaties mee te doen. Net zoals bij de algoritmes gebaseerd op Fourier-

transformaties is het niet mogelijk om direct de oplossing op te vragen, door de Quantummechanische eigenschap dat de kansfunctie zal gereduceerd worden tot één bepaalde waarde wanneer we deze proberen te meten. Terwijl we bij een simulatie net geïnteresseerd zijn in de waardes die de amplitudes aannemen als het Quantumsysteem verandert door verschillende operaties er op uit te voeren. Ook hier is het dus weer de moeilijkheid om een manier te vinden om deze informatie uit de Quantumtoestand te halen zonder dat deze instort. Onafhankelijk van dit feit zal een van de grootste toepassingen van Quantumcomputers het simuleren van Quantumsystemen worden. Vooral omdat we nu eenmaal in een wereld leven die onderhevig is aan de wetten van de Quantummechanica zal deze toepassing dus in alle vakgebieden gebruikt kunnen worden.

7.4 Quantumcomputers en complexiteit

Hoe krachtig zijn Quantumcomputers en wat kunnen we er allemaal mee berekenen? Zijn Quantumcomputers krachtiger dan klassieke computers? Het antwoord op deze vragen is nog steeds onbekend. We vermoeden heel sterk dat Quantumcomputers krachtiger zijn dan klassieke computers doordat we algoritmes kennen zoals die van Shor en Grover. We vermoeden dus dat we nu problemen efficiënt kunnen oplossen, Waarvan we denken dat ze geen efficiënte oplossing hebben op een klassieke computer. Het is echter ook nog mogelijk dat Quantumcomputers geen extra rekenkracht opleveren in vergelijking met klassieke computers. Het is mogelijk dat we voor dezelfde problemen ook efficiënte oplossingen kunnen vinden op klassieke computers. Toch hebben we een sterk vermoeden dat dit niet zo is.

We definiëren nu een complexiteitsklasse voor Quantumcomputers zoals we dat reeds voor klassieke computers hebben gedaan. BQP (Bounded-error Quantum Polynomial-time) is de klasse van problemen die efficiënt oplosbaar zijn op een Quantumcomputer wanneer we een beperkte kans op een error toelaten. In dit opzicht lijkt de klasse BQP eerder op de klasse BPP dan op de klasse P. We weten nog niet precies hoe de klasse BQP zich verhoudt ten opzichte van de andere complexiteitsklassen. Wel weten we dat alle problemen uit P ook efficiënt oplosbaar zijn op een Quantumcomputer. Ook kunnen we inzien dat $BQP \subseteq EXP$. Waarom is dit? Als we een klassieke computer exponentiële tijd geven, dan kunnen we gewoon een Quantumcomputer simuleren die het probleem oplost. Ook is er reeds bewezen door Bernstein en Vazirani dat $BQP \subseteq PSPACE$ [3]. Dus kunnen we besluiten dat de klasse BQP zich ergens tussen de klassen P en PSPACE bevindt. Als we ooit kunnen bewijzen dat Quantumcomputers strikt krachtiger zijn dan klassieke computers en er dus geldt dat $P \neq BQP$ dan zou hieruit ook volgen dat $P \neq PSPACE$.

8 Quantum Fouriertransformatie

Het ontdekken van een Quantumalgoritme voor het efficiënt oplossen van het factoring probleem is tot nu toe een van de belangrijkste ontdekkingen die er gebeurd zijn op het vlak van Quantumalgoritmen. Het factoring probleem werd lang als niet realistisch oplosbaar beschouwd op een klassieke computer. Wanneer we een factorisatie willen vinden voor een getal van n bits zal dit op een klassieke computer die gebruik maakt van number field sieve $exp(O(n^{1/3}\log^{1/3} n))$ stappen duren [25]. Dit zorgt ervoor dat zelfs bij een getal dat nog niet enorm groot is, we al snel meer tijd nodig hebben om een oplossing te vinden dan voor mensen bruikbaar en realistisch is. We zijn er zelfs zo van overtuigd dat het factoring probleem niet oplosbaar is op een computer, dat we daarom encryptiealgoritmes erop baseren omdat we de factorisatie van grote getallen niet kunnen achterhalen. Op een Quantumcomputer kunnen we factoring oplossen in maar $O(n^2 \log\, n\, \log\log\, n)$ stappen [18]. Dit geeft een enorme snelheidswinst ten opzichte van de klassieke methode. Door deze ontdekking vragen we ons ook af voor welke andere problemen we een zo grote snelheidswinst kunnen maken met behulp van een Quantumcomputer. Ook beginnen de mensen die achter de encryptiealgoritmes zitten zich zorgen te maken, dat wanneer we een Quantumcomputer realiseren hun algoritmes niet meer veilig genoeg zijn, omdat we op een efficiënte wijze getallen kunnen factoren.

8.1 De Quantum Fouriertransformatie

Zoals we al eerder gezegd hebben is de Fouriertransformatie een veel gebruikte transformatie om moeilijk oplosbare problemen te transformeren naar een makkelijker oplosbaar probleem. Een belangrijke stap vooruit in het vinden van Quantumalgoritmes was het ontdekken dat zo een Fouriertransformatie effectiever kan worden uitgevoerd op een Quantumcomputer dan op een klassieke computer.
We definiëren de Fouriertransformatie als de transformatie die een vector complexe getallen $x_0, x_2, x_3, ..., x_{N-1}$ omzet naar een andere vector complexe getallen $y_0, y_2, y_3, ..., y_{N-1}$ gedefinieerd als

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i(jk/N)} x_j$$

We kunnen deze transformatie ook definiëren voor Quantum Computing waarbij we deze transformatie niet uitvoeren op een vector complexe getallen, maar op een vector van basistoestanden $|x_0\rangle, ..., |x_{N-1}\rangle$. De transformatie wordt dan gedefinieerd als:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i(jk/N)} |k\rangle$$

We kunnen dit ook veralgemenen naar

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle$$

Hierbij zijn de amplitudes y_k de Fouriertransformatie van de amplitudes x_j. Deze transformatie is een unitaire bewerking. En dus kunnen we van deze transformatie een bewerking en poort maken voor een Quantumcomputer. We construeren nu stap voor stap een poort die een Quantum Fouriertransformatie kan uitvoeren en tonen zo ook aan dat deze operatie unitair is.In het geval dat wij gebruiken, stellen we dat $N = 2^n$. Dit zorgt ervoor dat de vector van basistoestanden, die we als input gebruiken er als volgt uitziet $|x_0\rangle, ..., |x_{2^n-1}\rangle$.
Met behulp van wat algebra kunnen we de Quantum Fouriertransformatie nu schrijven als: (we nemen ook de notatie $0.j_l j_{l+1} \dots j_m$ aan om de volgende bewerking aan te duiden $j_l/2 + j_{l+1}/4 + ... + j_m/2^{m-l+1}$)

$$\begin{aligned}
|j\rangle &\rightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2n-1} e^{2\pi ijk/2n} |k\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \dots \sum_{k_n=0}^{1} e^{2\pi ij(\sum_{l=1}^{n} k_l 2^{-l})} |k_1 \dots k_n\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \dots \sum_{k_n=0}^{1} \otimes_{l=1}^{n} e^{2\pi ijk_l 2^{-l}} |k_l\rangle \\
&= \frac{1}{2^{n/2}} \otimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} e^{2\pi ijk_l 2^{-l}} |k_l\rangle \right] \\
&= \frac{1}{2^{n/2}} \otimes_{l=1}^{n} \left[|0\rangle + 2^{2\pi ij2^{-l}} |1\rangle \right] \\
&= \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}
\end{aligned}$$

We kunnen dus besluiten dat

$$|j_1, ..., j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}$$

De manier waarop we de Fouriertransformatie nu hebben geschreven beschouwen we vanaf nu als de definitie van de Quantum Fouriertransformatie.

Deze definitie gaan we nu gebruiken om het Quantum circuit te maken dat ons toelaat de Fouriertransformatie uit te voeren op een Quantumcomputer. We zien dat de poort die een qubit kan Fourier transformeren overeenkomt met de poort R_k die unitair is.

$$R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

Om na te gaan of deze poort inderdaad de Quantum Fouriertransformatie toepast, gaan we hem uitvoeren op input $|j_1 ... j_n\rangle$ maar we sturen de eerste qubit eerst door een Hadamard-poort voordat we de poort R_k toepassen. De toestand waarin de input zich dan bevindt, wordt dan:

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0.j_1} |1\rangle) \ |j_2 ... j_n\rangle$$

Want $e^{2\pi i 0.j_1} = -1$ wanneer $j_1 = 1$ en voor de andere gevallen is het gelijk aan $+1$. We passen nu een controlled R_2 poort toe op deze toestand, dit geeft als resultaat.

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0.j_1 j_2} |1\rangle) \ |j_2 ... j_n\rangle$$

We blijven nu controlled - R poorten toepassen op de eerste qubit tot dat we aan poort R_n zitten. We hebben dan ondertussen de volgende toestand .

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle) \ |j_2 ... j_n\rangle$$

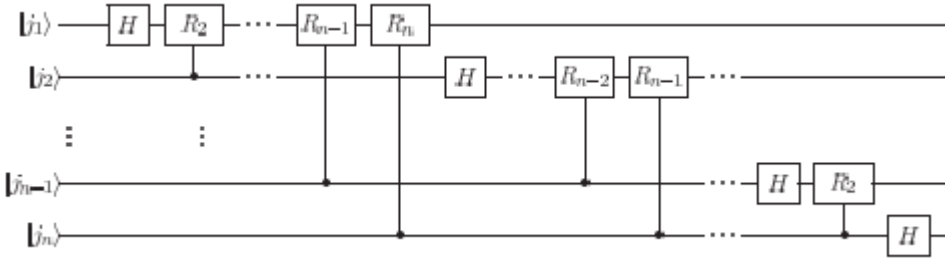
Nu we klaar zijn met de eerste qubit, gaan we dit proces herhalen. We passen achtereenvolgens de Hadamard-poort en controlled R_2 tot en met controlled -R_{n-1} toe op de tweede qubit. Dit geeft als resultaat de toestand:

$$\frac{1}{2^{2/2}}(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_2 j_3 ... j_n} |1\rangle) \ |j_3 ... j_n\rangle$$

We herhalen dit proces nu voor elke qubit totdat we als uiteindelijke toestand krijgen:

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_2 j_3 ... j_n} |1\rangle) \dots \ (|0\rangle + e^{2\pi i 0.j_n} |1\rangle)$$

We zien nu in de amplitudes de termen van de Fouriertransformatie verschijnen, maar de qubits staan nog in de verkeerde volgorde. Om de qubits te ordenen in de volgorde die we willen hebben, maken we gebruik van een speciaal swap circuit, waarmee we de eerste met de laatste wisselen de tweede



Figuur 20: Quantum Fouriertransformatie (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

met de voorlaatste enzovoort. Een swap circuit bestaat uit drie CNOT-poorten en is eerder al behandeld geweest. We krijgen nu

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle)\dots(|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n}|1\rangle)$$

Dit is exact dezelfde formule als de formule die we als definitie van de Quantum Fouriertransformatie hadden gebruikt. Het blijkt dus dat we het circuit om een Quantum Fouriertransformatie uit te voeren hebben geconstrueerd. Omdat we alleen maar gebruik maken van Hadamard-poorten en R-poorten die unitair zijn, gevolgd door swap circuits die ook unitair zijn, moet het volledige circuit ook unitair zijn. We geven in figuur 20 nog een schema van het circuit dat we zojuist beschreven hebben zonder de swap circuits op het einde.

Hoe effectief is dit circuit nu? We hebben beweerd dat we een enorme tijdswinst kunnen maken door de Fouriertransformatie op een Quantumcomputer uit te voeren. We gaan nu proberen aan te tonen hoeveel stappen we nodig hebben om een reeks van n qubits te transformeren. We nemen hierbij aan dat het aantal poorten dat we gebruiken overeenkomt met het aantal stappen dat het Quantum circuit nodig heeft om de Quantum Fouriertransformatie uit te voeren.
We maken voor de eerste qubit gebruik van een Hadamard-poort en $n-1$ R-poorten, wat een totaal geeft van n poorten voor de eerste qubit. De volgende qubit transformeren we door één R-poort minder te gebruiken. De derde qubit gebruikt nog één R-poort minder. Dit gaat zo verder tot we bij de laatste qubit alleen nog maar een Hadamard-poort nodig hebben. Dit geeft als resultaat dat we voor n qubits dus $n + (n-1) + (n-2) + ... + 1 = n(n+1)/2$ poorten nodig hebben. Buiten deze poorten die de transformatie berekenen gebruiken we ook nog een aantal swap-poorten die de termen in de juiste volgorde plaatsen. We hebben maximum $n/2$ van die swap-poorten nodig, Elke swap-poort bestaat uit 3 CNOT-poorten. We stellen dus dat dit

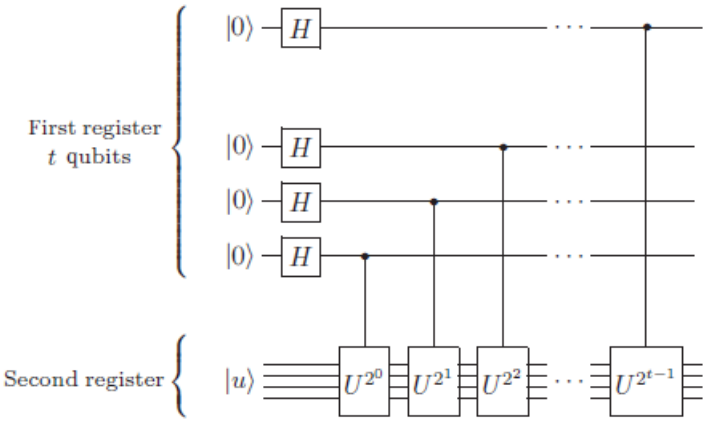
Quantum circuit een Fouriertransformatie kan berekenen in $O(n^2)$. Ter vergelijking het snelst bekende klassieke algoritme om een Fouriertransformatie uit te voeren. De fast Fouriertransformatie (FFT) heeft $O(n2^n)$ poorten nodig om dezelfde transformatie te doen [18]. Dit betekent dat we met behulp van een Quantumcomputer een exponentiële snelheidswinst hebben.

We kunnen dus enorm veel tijd winnen bij het berekenen van de Fouriertransformatie. Betekent dit dan ook dat we bekende algoritmes die gebruik maken van de Fouriertransformatie enorm kunnen versnellen? Het antwoord hierop is voorlopig nog 'Neen'. De Quantum Fouriertransformatie slaat zijn oplossing op in de amplitudes van de qubits. Zoals we al eerder hebben aangehaald is het niet mogelijk om de informatie uit de amplitudes van een qubit rechtstreeks te meten. We moeten dus een manier vinden om deze amplitudes te gebruiken zonder dat we de superpositie van de qubit verbreken. Een tweede probleem van de Quantum Fouriertransformatie is dat het voorlopig nog niet mogelijk is om de eerste toestand van de qubit voor te bereiden op een efficiëntie manier.

8.2 Faseschatting

We hebben nu een circuit ontwikkeld waarmee we snel een Fouriertransformatie kunnen uitvoeren, maar kunnen deze berekende waardes niet direct gebruiken. We moeten dus een manier vinden waarop we deze snelheidswinst toch kunnen gebruiken in onze algoritmes. Een belangrijke stap hierin is de faseschatting. Deze schatting zorgt ervoor dat we de Quantum Fouriertransformatie in vele andere algoritmes kunnen gebruiken.
Stel dat we beschikken over een unitaire operatie U, waarvan we een eigenvector schrijven als $|u\rangle$ met een eigenwaarde die gelijk is aan $e^{2\pi i\varphi}$. Het doel van de faseschatting is het zoeken van een waarde voor φ die onbekend is. Hiervoor maken we gebruik van black boxes die in staat zijn om de toestand $|u\rangle$ te produceren en de controlled U^{2^j}-poort uit te voeren. Het algoritme van faseschatting maakt gebruik van twee registers van qubits. Het eerste register bevat t qubits die allemaal in de toestand $|0\rangle$ zijn gebracht, waarbij de grootte van t afhangt van de correctheid en nauwkeurigheid die we verwachten van de schatting. Het tweede register bevat de toestand $|u\rangle$ die geproduceerd is door een black box. Deze twee registers gebruiken we nu als input voor het circuit uit figuur 21.

We passen dus eerst op elke qubit van het eerste register een Hadamardpoort toe, hierna voeren we op het tweede register opeenvolgende U-poorten tot de n-de macht toe, waarbij we deze macht steeds doen toenemen met een veelvoud van 2. Het eerste register ziet er na deze stappen als volgt uit.



Figuur 21: Faseschatting (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

$$\frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k} |k\rangle$$

We merken op dat dit overeenkomt met de Fouriertransformatie, zoals we die eerder beschreven hebben. We passen nu een inverse Fouriertransformatie toe op het eerste register, dat de controlle bits voor de U-poorten bevat. Het circuit om een inverse Quantum Fouriertransformatie uit te voeren bekomen we door het circuit voor een Quantum Fouriertransformatie in omgekeerde volgorde uit te voeren. Wanneer we aannemen dat we φ uitdrukken in t bits waarvoor we de volgende notatie gebruiken $\varphi = 0.\varphi_1 \dots \varphi_t$ kunnen we de toestand van het eerste register als volgt schrijven:

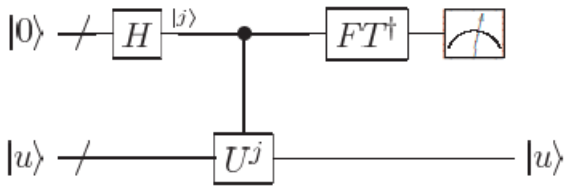
$$\frac{1}{2^{2/t}}(|0\rangle + e^{2\pi i 0.\varphi_t} |1\rangle)(|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2 \dots \varphi_t} |1\rangle)$$

Wanneer we nu een inverse Fouriertransformatie uitvoeren, krijgen we als resultaat de toestand $|\varphi_1 \dots \varphi_t\rangle$. We gaan nu het eerste register meten en krijgen dus een waarde voor φ met nauwkeurigheid t bits. We kunnen de operatie van de inverse Fouriertransformatie schrijven als:

$$\frac{1}{2^{t/2}} \sum_{j=0}^{2^{t-1}} e^{2\pi i \varphi j} |j\rangle |u\rangle \rightarrow |\tilde{\varphi}\rangle |u\rangle$$

waarbij $|\tilde{\varphi}\rangle$ de toestand voorstelt die we gaan meten om φ te schatten. Het volledige faseschattingscircuit geven we weer in figuur 22.

We gebruiken het faseschattingsalgoritme dus om de fase φ te schatten



Figuur 22: Faseschattingscircuit (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

van de eigenwaarde van een unitaire operatie U waarvoor we een eigenvector $|u\rangle$ hebben. Het echte nut van het faseschattingsalgoritme komt echter uit het feit dat we het kunnen gebruiken om andere problemen op te lossen met behulp van reductie. We zullen in de volgende delen enkele van deze problemen en oplossingen overlopen.

8.3 Order finding en Factoring

We gebruiken het algoritme voor faseschatting dus als onderdeel van andere algoritmes om zo problemen op te lossen met behulp van de Quantum Fouriertransformatie. Twee van die problemen waarvoor we zo een algoritme kennen zijn order finding en factoring. Deze twee problemen zijn equivalent aan elkaar. We kunnen dus met de oplossing voor het ene het andere probleem ook oplossen.

8.3.1 Order Finding

Het order finding probleem is als volgt gedefinieerd. We hebben twee positieve getallen x en N waarvoor geldt dat $x < N$ en ze hebben geen gemeenschappelijke factoren. We definieren nu de order van x *modulo* N als volgt. Neem het kleinst mogelijke positieve getal r zodat er geldt $x^r = 1(mod\, N)$. Dit getal r noemen we dan de order. Het order finding probleem bestaat er in zo een getal r te vinden wanneer x en N gegeven zijn. Het order finding heeft nog geen bekende efficiënte oplossing op een klassieke computer. We gaan nu gebruik maken van fase schatting om een Quantum circuit te maken, dat we kunnen gebruiken om het order finding probleem efficiënt op te lossen.
We maken dus gebruik van het algoritme van faseschatting. We definiëren hiervoor een unitaire operatie U waarop we dan het faseschattingsalgoritme toepassen:

$$U\,|y\rangle \equiv |x.y(mod\, N)\rangle$$

We stellen echter nog een aantal voorwaarden. $y \in \{0,1\}^L$ met L het aantal bits dat nodig is om N voor te stellen. En we verwachten dat wanneer $N \leq y \leq 2^L - 1$ we afspreken dat $xy(mod\, N)$ gelijk is aan y.
We berekenen nu dat de volgende toestanden

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{\left[\frac{-2\pi isk}{r}\right]} \left|x^k \, mod\, N\right\rangle$$

voor de getallen $0 \leq s \leq r-1$ eigentoestanden van U zijn want

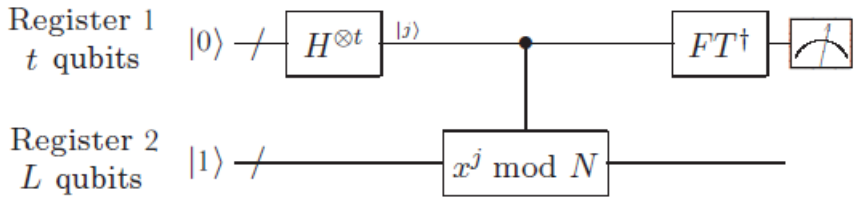
$$\begin{aligned} U\,|u_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} exp\left[\frac{-2\pi isk}{r}\right] \left|x^{k+1} \, mod\, N\right\rangle \\ &= e^{\left[\frac{2\pi is}{r}\right]} |u_s\rangle \end{aligned}$$

Met behulp van de faseschatting kunnen we nu uit deze eigentoestanden de eigenwaarden $e^{2\pi i \frac{s}{r}}$ berekenen, waar we dan op zijn beurt de waarde van r kunnen uit schatten, waarbij r de order is die we zoeken.
Voordat we aan faseschatting kunnen doen, zijn er wel twee voorwaarden waaraan voldaan moet zijn. Ten eerste moet het mogelijk zijn om de controlled-U^{2^j} poorten efficiënt uit te voeren voor elke waarde van j. Ten tweede moet het mogelijk zijn om op een efficiënte manier de eigentoestanden $|u_s\rangle$ te creëren. Aan de eerste voorwaarde is voldaan doordat er gebruik wordt gemaakt van modular exponentiation. We kunnen het achtereen uitvoeren van de controlled U^{2^j} poorten als volgt noteren.

$$\begin{aligned} |z\rangle\,|y\rangle &\rightarrow |z\rangle\, U^{z_t 2^{t-1}} \ldots U^{z_1 2^0}\,|y\rangle \\ &= |z\rangle \left|x^{z_t 2^{t-1}} \times \ldots \times x^{z_1 2^0} y(mod\, N)\right\rangle \\ &= |z\rangle\,|x^z y(mod\, N)\rangle \end{aligned}$$

We zien dus dat het uitvoeren van de U-poorten overeenkomt met vermenigvuldigen van het tweede register met het modular exponential $x^z(mod\, N)$ waarbij z de inhoud van het eerste register is.
De tweede voorwaarde is iets moeilijker om aan te voldoen, omdat we de waarde van r moeten kennen om $|u\rangle$te maken. Maar we kunnen dit probleem omzeilen door gebruik te maken van het feit dat

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$$



Figuur 23: Order finding circuit (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

Dit is een gevolg van:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{2i\pi sk/r} |u_s\rangle = \left|x^k \bmod N\right\rangle$$

Voor het geval dat $k = 0$. Dus het is voldoende om deze vergelijking te bewijzen [20]. We maken hiervoor gebruik van de definitie van $|u_s\rangle$

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{2i\pi sk/r} |u_s\rangle = \frac{1}{r} \sum_{s=0}^{r-1} \sum_{n=0}^{r-1} e^{2i\pi s(k-n)/r} |x^n \bmod N\rangle$$

We verwisselen nu de volgorde van de sommen en krijgen dan:

$$\sum_{s=0}^{r-1} e^{2i\pi s(k-n)/r} = r\delta_{k,n}$$

En weten dat $\delta_{k,n} = 0$ als $k \neq n$ en $\delta_{k,n} = 1$ als $k = n$. Dus er volgt dat:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{2i\pi sk/r} |u_s\rangle = \sum_{n=0}^{r-1} \delta_{k,n} \left|x^k \bmod N\right\rangle = \left|x^k \bmod N\right\rangle$$

Wanneer we nu de faseschatting uitvoeren met als grootte van het eerste register qubits $t = 2L + 1 + [\log(2 + \frac{1}{2\epsilon})]$ en in het tweede register L qubits in de toestand $|1\rangle$ prepareren. Dan volgt dat voor elke s tussen 0 en $r - 1$ we een schatting van de fase $\varphi \approx s/r$ kunnen vinden met een nauwkeurigheid van $2L + 1$ en dit met een waarschijnlijkheid van minstens $(1 - \epsilon)/r$. Het Quantum circuit dat we hiervoor gebruiken geven we weer in figuur 23.

Nu dat we de fase hebben gevonden, blijft er nog één stap over in ons algoritme, namelijk het vinden van de order r. Deze kunnen we berekenen met behulp van de fase waarvoor geldt dat $\varphi \approx s/r$. We kennen de fase met een nauwkeurigheid van $2L + 1$ bits en weten uit de definitie dat het een rationaal getal is, de verhouding van twee gehele getallen, waarvan het tweede niet 0 is. We kunnen dus een schatting maken van de dichtstbijzijnde breuk die hieraan voldoet. Deze kunnen we dan gebruiken om r te vinden. De

dichtsbijzijnde breuk vinden we met behulp van het kettingbreukalgoritme. Om een kettingbreuk te maken van een willekeurig getal $x \in \mathbb{R}$ schrijven we x als $x = \lfloor x \rfloor + (x - \lfloor x \rfloor) = \lfloor x \rfloor + \frac{1}{1/(x-\lfloor x \rfloor)}$. Het getal $\frac{1}{1/(x-\lfloor x \rfloor)}$ kunnen we dan opnieuw als een kettingbreuk schrijven. Dit kunnen we blijven doen totdat we het geval $(x - \lfloor x \rfloor) = 0$ tegenkomen. Dit kan alleen als x een rationeel getal is. We krijgen dus voor een input φ, dat een rationeel getal is, uiteindelijk een breuk van de vorm $\frac{s}{r}$, waardoor we dus de waarde van r te weten komen.

We bewijzen dat deze oplossing correct is met behulp van de volgende stelling.
Stel s/r is een rationaal getal zodat

$$\left| \frac{s}{r} - \varphi \right| \leq \frac{1}{2r^2}$$

Dan volgt hieruit dat s/r een resultaat is voor het kettingbreukalgoritme uitgevoerd op φ.
Omdat φ een benadering is van s/r die nauwkeurig is tot $2L + 1$ bits, moet ook gelden dat $|s/r - \varphi| \leq 2^{-2L-1} \leq 1/2r^2$, want $r < N \leq 2^L$ dus de stelling moet kloppen.
Samengevat kunnen we dus met behulp van de fase φ en het kettingbreukalgoritme efficiënt een schatting maken van r' en s', die geen factor gemeenschappelijk hebben zodat $s'/r' = s/r$. Dit getal r' is dan de kandidaat om de order te zijn die we zoeken. We kunnen eenvoudig verifiëren of deze oplossing de juiste is door $x^{r'} mod N$ uit te voeren en na te gaan of het resultaat 1 is. Als het resultaat 1 is dan is r' de order die we zochten.
Wat als de gevonden r' niet de order is. Hier zijn een aantal mogelijk oorzaken voor. Als eerste kan de schatting van de fase niet goed genoeg zijn, want het faseschattingsalgoritme is bounded error. We kunnen de kans op een foute schatting verkleinen door meer qubits te gebruiken bij het schatten. Een tweede fout kan optreden als s en r een gezamelijke factor hebben. De r' die we dan vinden is niet de order, maar een factor van de order. Dit kunnen we echter oplossen door het algoritme twee keer achter elkaar uit te voeren en zo twee waardes s'_1, r'_1 en s'_2, r'_2 te bekomen. Als s'_1, s'_2 geen gezamenlijke factor hebben, kunnen we altijd r te weten komen door het kleinste gemene veelvoud te nemen van r'_1, r'_2.

Hoe efficiënt is dit order finding algoritme nu? Opnieuw gebruiken we hiervoor als maat het aantal Quantumpoorten dat we nodig hebben om het circuit uit te voeren. Als eerste maken we gebruik van een aantal Hadamardpoorten. Hun aantal wordt begrenst door $O(L)$. Verder maken we gebruik van een inverse Quantum Fouriertransformatie. Dit is gewoon een Quantum Fouriertransformatie omgekeerd en gebruikt dus zoals al aangetoond $O(L^2)$ poorten. De grootste kost komt echter van de poorten die nodig zijn om het

tweede register te maken en de poorten nodig om het kettingbreukalgoritme uit te voeren. Deze twee hebben beide $O(L^3)$ poorten nodig. We voeren het kettingbreukalgoritme echter tweemaal achter elkaar uit, omdat de waarde van r' niet altijd overeenkomt met de r die we zoeken. Maar door het kettingbreukalgoritme twee keer uit te voeren, kunnen we de kans verhogen dat r de juiste order is die we zoeken. Dit maakt dat het algoritme een totale kost heeft van $O(L^3)$. We vatten hieronder even kort de eigenschappen van het order finding algoritme samen.

Quantum Order Finding

Input :

1. Een black box $U_{x,N}$ die de volgende transformatie kan uitvoeren $|j\rangle\,|k\rangle \rightarrow |j\rangle\,|x^j k \; mod \; N\rangle$. Waar geldt dat x copriem is aan N dat uit L-bits bestaat.
2. $t = 2L + 1 + [log(2 + \frac{1}{2\epsilon})]$ qubits die geïnitialiseerd zijn in de toestand $|0\rangle$.
3. L qubits geïnitialiseerd in de toestand $|1\rangle$.

Output:

Het kleinste gehele getal $r > 0$ zodat $x^r = 1 \; (mod \, N)$.

Runtime:

$O(L^3)$ operaties.

Algoritme:

1. $|0\rangle\,|1\rangle$ — Initiële toestand
2. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^{t-1}} |j\rangle\,|1\rangle$ — creëer superpositie op eerste register
3. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^{t-1}} |j\rangle\,|x^j \; mod \; N\rangle$ — Pas $U_{x,N}$ toe op het tweede register
   $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} \,|j\rangle\,|u_s\rangle$
4. $\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \widetilde{|s/r\rangle}\,|u_s\rangle$ — Voer inverse Quantum Fouriertransformatie uit op eerste register
5. $\rightarrow \widetilde{s/r}$ — Meet het eerste register
6. $\rightarrow r$ — Pas het kettingbreukenalgoritme toe

8.3.2 Factoring

In het begin van dit deel hebben we al aangehaald dat wanneer we een oplossing vinden voor het order finding probleem, we ook een oplossing vinden voor het factoring probleem met behulp van reductie. We definiëren het factoring probleem als volgt.
We krijgen een positief geheel getal N dat geen priemgetal is. Zoek nu de getallen, waarvan het product gelijk is aan het getal N. Het blijkt dat we gebruik kunnen maken van het order finding algoritme om een oplossing te vinden voor het factoring probleem.
We kunnen nu een algoritme definiëren dat een factor kan vinden voor N. Dit algoritme bestaat uit verschillende stappen, waaronder een stap die het order finding algoritme toepast. De meeste stappen van dit algoritme zouden we ook efficiënt kunnen uitvoeren op een klassieke computer, behalve de stap waarbij we gebruik maken van order finding, waarvoor tot nu toe nog geen efficiënt algoritme voor bekend is op een klassieke computer. Door onze oplossing voor factoring meerdere keren uit te voeren, kunnen we een complete factorisatie vinden voor een getal N.
We testen eerst of N even is. Wanneer dit zo is zijn we klaar, want dan is 2 een priemfactor van N. Als tweede testen we dat N geen macht is van een priemgetal. Dit kunnen we eenvoudig doen door de vierkantswortel, 3-de machtswortel, ..., k-de machtswortel te nemen uit N, waarbij $k \leq \log_3 N$ en door na te gaan of het resultaat geen geheel getal is.

Factoring
Input:
Een positief geheel getal N dat geen priem is.
Output:
Een niet triviale factor van het getal N.
Runtime:
$O((\log N)^3)$ operaties.
Algoritme:

1. Als N even is geef factor 2 terug.
2. Ga na of we getallen a en b kunnen vinden zodat $a \geq 1$ en $b \geq 2$ en ook $N = a^b$. Als we zulke getallen kunnen vinden, geef dan a als factor. (Hiervoor kunnen we gebruik maken van een klassiek algoritme)
3. Kies een willekeurig getal x dat ligt tussen 1 en $N-1$. Als $ggd(x, N) > 1$ geef dan het getal $ggd(x, N)$ als factor terug. (Ook hiervoor bestaat een klassiek algoritme)
4. Gebruik het Quantum order finding algoritme om een getal r te vinden dat de order is van x *modulo* N.

5. Als r oneven is of $x^{r/2} = -1 (mod\, N)$ ga dan terug naar stap 1.

6. Bereken $ggd(x^{r/2} - 1, N)$ en $ggd(x^{r/2} + 1, N)$. Test dan of één van de grootste gemene delers een factor is van N en geef dit dan terug als factor. Anders vindt het algoritme geen factor.

8.4 RSA

We hebben nu de algoritmes gezien die ons in staat stellen efficiënt een oplossing te vinden voor factoring en order finding. We hebben in het begin aangegeven dat deze algoritmes nu kunnen gebruikt worden om het RSA cryptosysteem te kraken. Dit is een zeer belangrijk feit, omdat RSA een van de meest gebruikte cryptosystemen is op dit moment. Een Quantumcomputer zou dus in staat moeten zijn om dit systeem te breken. Dit zou voor veel websites een veiligheidslek betekenen. Het kraken van RSA en een nieuw Quantumalgoritme om encryptie te doen, zijn toepassingen waar enorm veel aan gewerkt wordt, zelfs de NSA is hierin geïnteresseerd [28].

In dit deel gaan we nu kort het principe achter het RSA-systeem uitleggen, waardoor het ook duidelijk wordt dat een Quantumcomputer dit zou moeten kunnen kraken.
Hoe werkt zo een encryptiesysteem nu? Het doel van een encryptiesysteem is het verbergen van informatie de we willen doorsturen, naar elkaar of naar een site. Om dit te doen maken we meestal gebruik van public key cryptografie. Bij public key cryptografie maken we gebruik van twee keys die we genereren. Eén ervan is de public key en is toegankelijk voor iedereen die een boodschap wil sturen. De tweede key is de secret key. Deze is alleen gekend voor degene die de boodschappen mag lezen. Wanneer we dan een boodschap willen sturen, halen we de public key op en gebruiken deze om een transformatie uit te voeren op de boodschap, waardoor deze versleuteld wordt. Welke deze bewerking is, hangt af van het encryptiealgoritme dat we gebruiken. Het is wel belangrijk dat deze transformatie niet om te keren is, zelfs wanneer we de public key kennen. Zo is het dus niet mogelijk om de boodschap te decoderen als we ze onderscheppen. De ontvanger beschikt echter over de secret key. Met deze secret key kunnen we wel een transformatie uitvoeren die terug de originele boodschap oplevert.

Er zijn tot nu toe nog geen bekende systemen waarbij we zeker weten dat de public key transformatie niet om te keren is met behulp van de public key. We kennen echter wel systemen die al zo veel getest zijn, dat we geloven dat ze niet op te lossen zijn met behulp van een klassieke computer. Het meest gebruikte van deze systeem is RSA, genoemd naar de ontdekkers Rivest, Shamir en Adleman. De bewerking waarop RSA zijn veiligheid baseert, is

de moeilijkheid die klassieke computers hebben bij het vinden van priemfactoren van grote getallen.
De keys van RSA zijn dus gebaseerd op priemgetallen en worden als volgt gecreëerd:

1. Kies twee priemgetallen p en q
2. Bereken het product $n = pq$
3. Kies een willekeurig klein oneven getal e dat copriem is aan $\varphi(n) = (p-1)(q-1)$
4. Bereken d, $d = (1/e)\ mod\, \varphi(n)$
5. De RSA public key is dan het paar $P = (e, n)$
   De RSA secret key is dan het paar $S = (d, n)$

Wanneer we nu een boodschap M willen coderen, moeten we de public key P kennen. We kunnen boodschappen coderen die maximaal $log\, n$ bits bevatten. Als de boodschap langer is, moeten we ze opsplitsen in kortere stukken. We versleutelen de boodschap met de volgende bewerking:

$$E(M) = M^e (mod\, n)$$

$E(M)$ is de versleutelde versie van de boodschap. Wanneer we de originele boodschap willen, moeten we de secret key bezitten om de volgende bewerking te kunnen toepassen.

$$E(M) \rightarrow D(E(M)) = E(M)^d (mod\, n)$$

Hoe kunnen we dit encryptiesysteem nu breken? Hiervoor kunnen we gebruik maken van zoals eerder aangehaald order finding en factoring.
De eerste manier die we kunnen gebruiken, maakt gebruik van het factoringalgoritme om de priemgetallen p en q te achterhalen, die gebruikt werden bij het samenstellen van de public en secret key. Als we $n = pq$ zouden kunnen factoren, zodat we de priemgetallen p en q te weten komen, dan zouden we zelf de secret key kunnen construeren op dezelfde manier als RSA dat doet. Maar omdat er nog geen bekend algoritme bestaat op een klassieke computer, dat efficiënt grote getallen kan factoren, nemen we aan dat het RSA-systeem nog altijd veilig is.
Als tweede gaan we de methode met order finding gebruiken. Hiermee willen we laten zien dat het zelfs niet nodig is om gebruik te maken van factoring en we genoeg hebben aan order finding, wat een deel van factoring is.
Stel dat we het gecodeerde bericht $E(M)$ kunnen onderscheppen of afluisteren en we kennen de public key (e, n). Is het dan mogelijk om de originele boodschap te weten te komen? Dit kan als we de order kunnen vinden van de versleutelde boodschap. Dus als we het kleinst mogelijke getal r kunnen

vinden zodat $(M^e)^r = 1(mod n)$. We nemen aan dat er zo een order bestaat. We weten ook dat r een deler is van $\varphi(n)$ en e is copriem aan $\varphi(n)$, dus e is ook copriem aan r. Er bestaat dus ook een omgekeerde van e *modulo* r, deze noemen we d' zodat $ed' = 1 + kr$ voor een getal k. We kunnen hiermee nu de boodschap ontcijferen door d' te gebruiken als secret key want

$$\begin{aligned}(M^e)^{d'}(mod\ n) &= M^{1+kr}(mod\, n) \\ &= M \cdot M^{kr}(mod\, n) \\ &= M(mod\, n)\end{aligned}$$

We kennen nu de originele boodschap M zonder dat we de secret key hebben nodig gehad. Het is dus mogelijk om RSA te omzeilen en de secret key niet nodig te hebben. Alleen kennen we nog geen algoritme dat efficiënt het order finding probleem kan oplossen op een klassieke computer. Daarom wordt het RSA cryptosysteem alsnog als veilig beschouwd, totdat we kunnen beschikken over Quantumcomputers, Aangezien we een Quantum circuit kennen dat het order finding probleem kan oplossen in $O(L^3)$ operaties.

Deze veiligheid blijft net als bij het vorige voorbeeld enkel behouden totdat we over zo een Quantumcomputer beschikken. Men weet niet of deze problemen een efficiënte oplossing hebben of niet op een klassieke computer en of RSA echt zo veilig is als we denken. Ook is het mogelijk dat er nog andere manieren bestaan om het RSA-systeem te kraken. Er zijn al vele pogingen geweest, sinds het RSA-systeem bestaat. Tot nog toe is er geen enkele van gelukt. Er wordt dus algemeen aangenomen, dat RSA veilig is tegen aanvallen van klassieke computers.

8.5 Andere algoritmes met de Quantum Fouriertransformatie

We hebben tot nu toe al een aantal algoritmes gezien, die gebruik maken van de Quantum Fouriertransformatie om tot een oplossing te komen zoals de fase schatting, order finding en factoring. Er zijn echter nog heel wat andere problemen die gebruik maken van de Quantum Fouriertransformatie om efficiënt een oplossing te vinden voor een probleem. We kunnen deze problemen groeperen met een meer algemeen probleem, het hidden subgroup probleem. Deze groep van problemen bevat alle bekende exponentieel snelle oplossingen, die bekend zijn op een Quantumcomputer, waarbij we gebruik maken van de Quantum Fouriertransformatie. Het hidden subgroup problem definiëren we als volgt.
Neem een functie f van een eindig gegenereerde groep G van een eindige

set X, zodat de functie f constant is over de cosets van een subgroep K, en distinct over elke coset. Gegeven een Quantum black box die de unitaire operatie $U|g\rangle|h\rangle = |g\rangle|h \oplus f(g)\rangle$ kan uitvoeren voor $g \in G$, $h \in X$ en $\oplus$, een goed gekozen binaire operatie op X. Zoek nu een set die K kan genereren.

Order finding, period finding, discrete logarithm, fase schatting en factoring en andere problemen zijn allemaal voorbeelden van een hidden subgroup probleem.
Stel dat G een eindige Abelse groep is, dan kunnen we een algemene werkwijze geven die het hidden subgroup probleem oplost, vergelijkbaar met de algoritmes die we reeds gezien hebben. Als eerste plaatsen we een aantal qubits in superpositie door gebruik te maken van Hadamard-poorten. Als tweede voeren we een black box operatie uit op deze qubits. We duiden deze operatie aan met f. We krijgen dan

$$\frac{1}{\sqrt{G}} \sum_{g \in G} |g\rangle |f(g)\rangle$$

Hierna schrijven we $|f(g)\rangle$ in de vorm van een Fouriertransformatie, waarna we een inverse Fouriertransformatie gebruiken om de informatie die we nodig hebben uit de amplitudes proberen te halen. De functie f en de manier waarop we informatie uit de amplitudes halen, is afhankelijk van het probleem dat we willen oplossen.

Het groeperen van deze Quantumalgoritmes in een meer algemeen probleem geeft de indruk dat er nog meer problemen zijn die op een gelijkaardige manier kunnen worden opgelost. Zijn er buiten de eindige Abelse groepen nog andere groepen waarvoor we een gelijkaardige oplossing zouden kunnen gebruiken? Een voorbeeld van zo een probleem is om isomorfie van grafen aan te tonen, waarbij we willen nagaan of twee grafen gelijk zijn op de namen van de knopen na. De groep waarover we het dan hebben is de symetrische groep S_n, waarin we de mogelijke permutaties van de knopen kunnen beschrijven als transformaties. Er bestaan algoritmes om de Fouriertransformatie uit te voeren over deze groep, maar dit zorgt er niet voor dat we dan ook een Quantumalgoritme kennen dat isomorfie van grafen kan oplossen. We weten voorlopig dus nog niet of isomorfie van grafen in BQP zit.

9 Quantum zoekalgoritmes

De tweede grote groep van problemen, waarvoor we op een Quantumcomputer een efficiëntere oplossing kennen dan voorlopig bekend is op een klassieke computer, zijn zoekproblemen. Wanneer we op een klassieke computer willen zoeken in een verzameling van N elementen en er is niets bekend over de verzameling, dan zal dit $O(N)$ operaties vragen. We gaan gewoon elk element af en controleren of dit het gezochte element is. Nu blijkt dat het mogelijk is om op een Quantumcomputer dit probleem sneller oplossen. Zo is het namelijk mogelijk om met behulp van Grovers algoritme zo een zoek probleem op te lossen in $O(\sqrt{N})$ operaties. Deze snelheidswinst is wel niet zo spectaculair als de exponentiële winst, die we hadden bij de algoritmes die gebruik maken van de Quantum Fouriertransformatie. Maar de bekende toepassingen van een sneller zoekalgoritme zijn veel groter, zodat het toch enorm interessant is om hier aandacht aan te besteden.

9.1 Oracle

Stel dat de verzameling waarin we gaan zoeken N elementen bevat. We gaan niet direct de elementen zelf zoeken, maar maken gebruik van een index. De index van een element is een getal tussen 0 en $N-1$ waarmee we elk element een uniek ID kunnen geven.Om het eenvoudig te houden nemen we aan dat $N = 2^n$, zodat de index opgeslagen kan worden in n bits. We nemen ook aan dat een zoekopdracht een aantal oplossingen heeft uitgedrukt door M, waarvoor steeds geldt dat $1 \leq M \leq N$. We kunnen zo een zoekopdracht ook voorstellen als een functie $f(x)$ waarbij de input x een getal is tussen 0 en $N-1$. Als x een oplossing is voor het zoekprobleem dan is $f(x) = 1$, anders is $f(x) = 0$.
We creëren nu een Quantum oracle, dit is een black box die in staat is om de oplossing van een zoekprobleem te herkennen. De oracle maakt hiervoor gebruik van een oracle qubit. We kunnen deze black box voorstellen door de unitaire operatie O die we definiëren als:

$$|x\rangle\,|q\rangle \xrightarrow{O} |x\rangle\,|q \oplus f(x)\rangle$$

Hierbij is $|x\rangle$ het register dat de zoekindexen bevat. Deze operatie zorgt ervoor dat de tweede qubit geflipt wordt wanneer f(x) = 1 en zal de tweede qubit niet veranderen wanneer f(x) = 0. Het oracle kan dus nagaan of x een resultaat is voor de zoekopdracht, door het oracle de toestand $|x\rangle\,|0\rangle$ mee te geven en na de operatie na te gaan of de tweede qubit zich nu in de toestand $|1\rangle$ bevindt. Er gebeurt echter iets bijzonders wanneer we als tweede qubit niet $|0\rangle$ meegeven maar de toestand $\frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$. Wanneer we het oracle dan uitvoeren krijgen we $|x\rangle\,(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}})$. Wanneer x geen oplossing is verandert

er nog altijd niets maar wanneer x wel een oplossing is van het zoekprobleem krijgen we de volgende toestand $|x\rangle\,(\frac{|1\rangle-|0\rangle}{\sqrt{2}}) = (-1)^{f(x)}\,|x\rangle\,(\frac{|0\rangle-|1\rangle}{\sqrt{2}})$. De actie van het oracle laat dus het tweede qubit onveranderd en flipt nu de eerste qubit. We kunnen dus het tweede qubit buiten beschouwing laten en ons in het vervolg alleen bezig houden met deze eerste qubit. De operatie van het oracle wordt dus als volgt:

$$|x\rangle \xrightarrow{O} (-1)^{f(x)}\,|x\rangle$$

We merken op dat het oracle de oplossingen van de zoekopdracht aanduidt door het teken van de oplossing te flippen. Op een Quantumcomputer hebben we slechts $O(\sqrt{N/M})$ operaties van dit oracle nodig, om M oplossingen te vinden in een verzameling van N elementen. Dit bewijzen we later.
We beschrijven hier een black box die alle oplossingen van het zoekalgoritme blijkbaar al kent. Is het nu mogelijk om een zoekalgoritme te baseren op dit oracle? Het blijkt nu dat er een groot verschil is tussen het herkennen van een oplossing voor een zoekprobleem en het vinden van deze oplossing. We kunnen zelfs een oplossing herkennen zonder dat het nodig is te weten dat dit een oplossing is. We proberen dit abstracte gegeven duidelijk te maken met een voorbeeld.
We proberen het factoringprobleem op te lossen als een zoekprobleem. We krijgen dus een groot getal m en zoeken twee priemgetallen p en q die als product het getal m uitkomen. Een voor de hand liggende oplossing is het afgaan van alle getallen tussen 2 en $m^{1/2}$. Testen of het getal priem is en of het getal m deelt. Als het getal m deelt, dan nemen we dit als p en nemen als q het resultaat van de deling. Wanneer ook q priem is zal ons algoritme geslaagd zijn. Het is makkelijk in te zien dat dit eenvoudige algoritme hiervoor $O(m^{1/2})$ stappen zal nodig hebben. We gaan er van uit dat het nagaan of een getal priem is geen stappen toevoegt.
We gaan nu gebruik maken van het Quantum zoekalgoritme om dit algoritme te versnellen. De toestand $|x\rangle$ stelt nu de getallen tussen 2 en $m^{1/2}$ voor en zal de waardes x flippen waarvoor geldt dat m deelbaar is door x. We moeten nu enkel de kleinste priemdeler zien te vinden om een juiste oplossing te krijgen met dit algoritme. De functie $f(x)$ stellen we hier voor als een circuit, dat een deling uitvoert op het getal m met de input x. Wanneer $f(x) = 1$ zal deze deling gelukt zijn zonder rest, anders is $f(x) = 0$. Het is nu makkelijk om een oracle te construeren die een juiste oplossing kan herkennen voor het factoringprobleem zonder dat deze de oplossingen kent. Wanneer we dus het Quantum oracle gebruiken in plaats van op de klassieke manier door de lijst van mogelijke oplossingen te lopen, zien we dat we een snelheidswinst maken, want een Quantum zoekalgoritme heeft maar $O(\sqrt{N/M})$ operaties nodig om een oplossing te vinden. We hebben nu dus nog maar $O(\sqrt{m^{1/2}})$ stappen nodig. Er zijn $N = m^{1/2}$ mogelijke oplossingen en maar $M = 1$ oplossing.

Dit voorbeeld geeft aan hoe we met het Quantum zoekalgoritme makkelijk een snelheidswinst kunnen maken door klassieke algoritmes, die gebaseerd zijn op zoeken, te versnellen. Voor factoring echter hebben we hier niet veel voordeel van, want we kennen een veel sneller Quantumalgoritme. Maar er zijn problemen die hier wel veel voordeel uit kunnen halen. We komen hier later op terug wanneer we bekijken hoe we dit Quantum zoekalgoritme kunnen toepassen om algoritmes uit NP te versnellen.

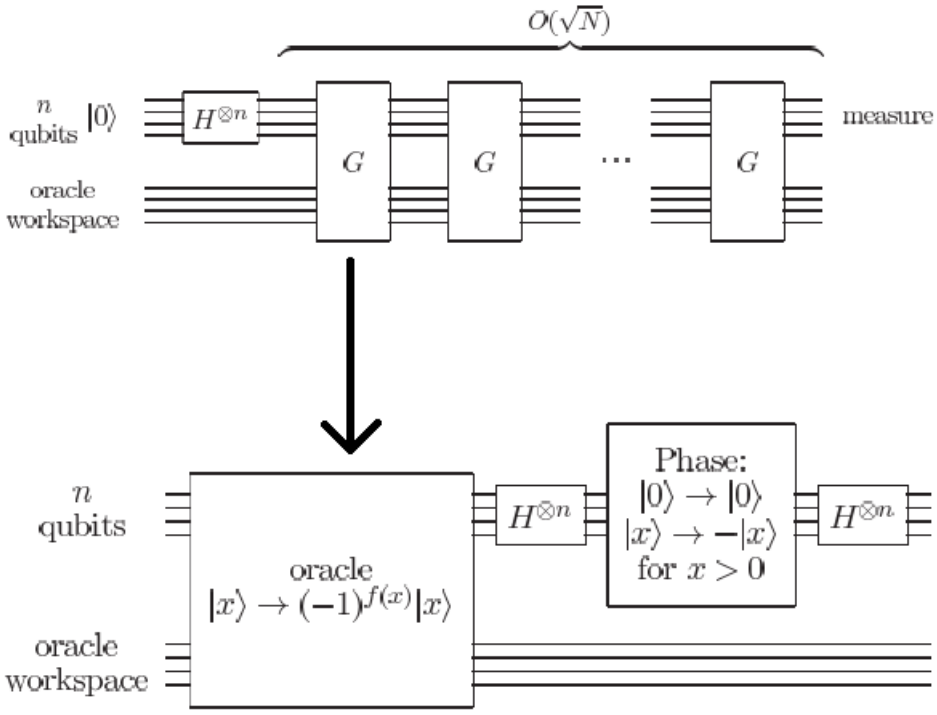
9.2 Het circuit

We gebruiken het oracle dus om de Quantum zoekalgoritmes uit te voeren. We geven nu in figuur 24 een volledig circuit, waar we zo een zoekalgoritme op laten werken. De precieze werking en inhoud van de oracle is niet belangrijk en verandert van probleem tot probleem. We hebben twee registers van qubits nodig om het circuit als input te geven, alleen het eerste register wordt gebruikt door het zoekalgoritme en bevat n qubits. Het tweede register wordt door het oracle gebruikt als werkruimte, de grootte van dit tweede register hangt dus af van de gebruikte oracle.

We beginnen met het register in de toestand $|0\rangle^{\otimes n}$ (dit geeft n qubits aan die in de positie $|0\rangle$zijn). Op deze qubits passen we een Hadamard-poort toe om ze in een toestand van superpositie te krijgen. Op dit moment passen we het eigenlijke algoritme toe, dat bestaat uit het zo weinig mogelijk herhalen van de grover operatie die we aanduiden als G. De grover operatie is uit te leggen in vier stappen.

1. Pas de operatie O toe (oracle).
2. Pas een Hadamard-poort toe op het eerste register.
3. Voer een phase shift van -1 uit op alle qubits in het register, behalve als ze in de toestand $|0\rangle$ zijn.
4. Pas een Hadamard-poort toe op het eerste register.

Hoe efficiënt is dit circuit nu? Het toepassen van de Hadamard-poorten vraagt tweemaal $O(\log\, n)$ operaties. De phase shift kunnen we maken door gebruiken te maken van $O(n)$ poorten. We beschrijven hier kort hoe zo een phase shift eruitziet. We willen dat de toestand $|0\rangle$ onveranderd blijft. En dat de toestand $|x\rangle$ als resultaat $-|x\rangle$ wordt. Dit resultaat kunnen we bekomen door een operatie $(2\,|0\rangle\,\langle 0| - I)$ te definiëren, die we de phase shift poort noemen [19].
De kost van het oracle is onbekend, omdat deze afhangt van de toepassing



Figuur 24: Quantum zoek circuit en daaronder Uitvergroting van G circuit (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

waarvoor we het zoekcircuit willen uitvoeren. Wel zien we dat elke iteratie maar één keer het oracle aanspreekt.
We merken ook op dat de combinatie van de Hadamard-poort, phase shift poort en tweede Hadamard-poort als volgt kan geschreven worden:

$$H^{\otimes n}(2\left|0\right\rangle\left\langle 0\right| - I)H^{\otimes n} = 2\left|\psi\right\rangle\left\langle\psi\right| - I$$

waarbij $\left|\psi\right\rangle \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \left|x\right\rangle$ het eerste register is nadat het in superpositie is gebracht. De Grover operatie schrijven we dus als

$$G = (2\left|\psi\right\rangle\left\langle\psi\right| - I)O$$

waarbij O de operatie van het oracle voorstelt.

9.3 Visualisatie en performantie

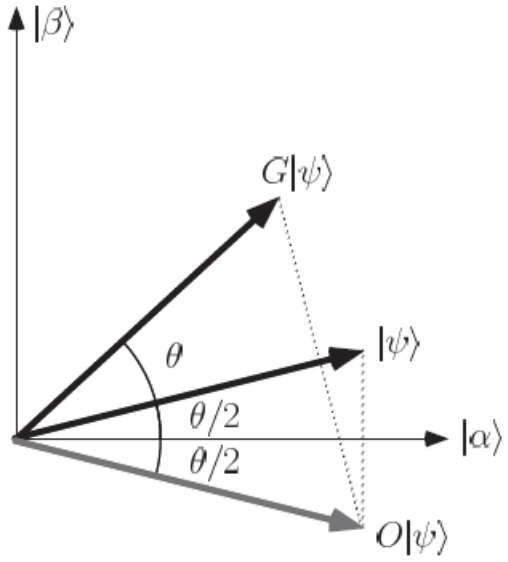
We hebben hierboven een circuit geconstrueerd om een Quantum zoekalgoritme uit te voeren. Dit circuit maakt gebruik van een reeks Grover operaties die achtereenvolgens worden uitgevoerd. We willen nu graag weten hoeveel van die iteraties we moeten uitvoeren om tot een resultaat te komen. Om dit te weten te komen, gaan we de Grover operatie op een andere manier voorstellen, namelijk als een rotatie in een tweedimensionale ruimte die bepaald wordt door de vector gegeven door $\left|\psi\right\rangle$ en de toestand bestaande uit de superpositie van de mogelijke oplossingen van het zoekprobleem. Om dit te helpen visualiseren definiëren we twee nieuwe notaties. De eerste is $\sum_x'$ die de som is over alle x, die een oplossing zijn voor het zoekprobleem. De tweede is $\sum_x''$ die de som is over alle x, die geen oplossing zijn van het zoekprobleem. Met behulp van deze nieuwe notaties definiëren we twee nieuwe toestanden:

$$\begin{aligned}
\left|\alpha\right\rangle &\equiv \frac{1}{\sqrt{N-M}} \sum_x{}'' \left|x\right\rangle \\
\left|\beta\right\rangle &\equiv \frac{1}{\sqrt{M}} \sum_x{}' \left|x\right\rangle
\end{aligned}$$

We kunnen nu de toestand $\left|\psi\right\rangle$ uitdrukken in functie van deze twee nieuwe toestanden als:

$$\left|\psi\right\rangle = \sqrt{\frac{N-M}{N}}\left|\alpha\right\rangle + \sqrt{\frac{M}{N}}\left|\beta\right\rangle$$

Dus kunnen we de initiële toestand van de Quantumcomputer die het zoekalgoritme uitvoert ook zien als de ruimte die wordt bepaald door de vectoren $\left|\alpha\right\rangle$ en $\left|\beta\right\rangle$. We kunnen dan nu de O operatie (oracle) voorstellen als een



Figuur 25: Visualisatie van een Grover operatie (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

spiegeling rond de vector $|\alpha\rangle$ in de ruimte die we zojuist hebben gedefinieerd. We kunnen dit schrijven als $O(a\,|\alpha\rangle + b\,|\beta\rangle) = a\,|\alpha\rangle - b\,|\beta\rangle$. Op dezelfde manier kunnen we ook het deel van Grover operatie $2\,|\psi\rangle\,\langle\psi| - I$ zien als een spiegeling in hetzelfde vlak maar deze keer rond de vector $|\psi\rangle$. Het product van deze twee spiegelingen geeft een rotatie die gelijk is aan de Grover operatie $G = (2\,|\psi\rangle\,\langle\psi| - I)O$. Dit betekent dat de toestand $G^k\,|\psi\rangle$ zich steeds in het zojuist gedefinieerde vlak zal bevinden voor elke waarde van k.
Vervolgens kunnen we nu ook de rotatiehoek definiëren. We nemen aan dat $\cos\theta/2 = \sqrt{(N-M)/N}$. Dit betekent dat $|\psi\rangle = cos\theta/2\,|\alpha\rangle + sin\theta/2\,|\alpha\rangle$. Als we nu de twee spiegelingen uitvoeren die samen G vormen dan krijgen we de rotatie:

$$G\,|\psi\rangle = cos\frac{3\theta}{2}\,|\alpha\rangle + sin\frac{3\theta}{2}\,|\beta\rangle$$

De hoek van de rotatie is dus gelijk aan θ. We proberen dit nu te visualiseren op afbeelding 25.

Wanneer we de Grover operatie meerdere malen achter elkaar uitvoeren krijgen we het resultaat

$$G^k\,|\psi\rangle = cos(\frac{2k+1}{2}\theta)\,|\alpha\rangle + sin(\frac{2k+1}{2}\theta)\,|\beta\rangle$$

Dit betekent dus dat we de Grover operatie kunnen voorstellen als een rotatie in de tweedimensionale ruimte gevormd door $|\alpha\rangle$ en $|\beta\rangle$. Per toepassing van G wordt er geroteerd met de hoek gelijk aan θ. Als we meerdere G operaties toepassen op de initiële vector $|\psi\rangle$ zien we dat ze door de rotatie steeds dichter in de buurt komt te staan van vector $|\beta\rangle$. We hebben in het begin deze vector gedefinieerd als de som van de oplossingen voor het zoekprobleem. Dit geeft ons het inzicht dat wanneer de rotatie de initiële vector laat samenvallen met de $|\beta\rangle$ vector, we een oplossing hebben bereikt voor het zoekprobleem.

We willen weten hoeveel iteraties van de Grover operatie we nodig hebben om tot een oplossing te komen. We kunnen met behulp van de visualisatie van hierboven dit probleem definiëren als: hoeveel rotaties zijn er nodig om te zorgen dat $|\psi\rangle \approx |\beta\rangle$. Omdat de initiële toestand van het systeem $|\psi\rangle = \sqrt{\frac{N-M}{N}}\,|\alpha\rangle + \sqrt{\frac{M}{N}}\,|\beta\rangle$ is, weten we dat we moeten roteren rond de hoek gegeven door $arccos(\sqrt{M/N})$. We moeten nu enkel een manier vinden om te weten te komen na hoeveel iteraties van de G-operatie we over deze hoek geroteerd hebben. Om dit uit te rekenen maken we een functie $CI(x)$. Deze functie geeft als resultaat het gehele getal dat het dichtste bij x ligt. We spreken af dat we naar onder afronden. Het aantal keer dat we de Grover operatie moeten uitvoeren wordt dan gegeven door:

$$R = CI(\frac{arccos\sqrt{M/N}}{\theta})$$

De vector $|\psi\rangle$ zal dan nog maximaal $\theta/2 \leq \pi/4$ van $|\beta\rangle$ verwijderd zijn. We kunnen dan de juiste oplossing meten met minstens een kans van $1/2$. Voor sommige waardes van M is het mogelijk om een grotere kans uit te komen, vooral wanneer M veel kleiner is dan N. We weten nu hoeveel iteraties van G we nodig hebben om een oplossing te vinden, maar we moeten hiervoor weten hoeveel oplossingen M er zijn. Van deze voorwaarde zouden we graag verlost zijn.
Ook is de vergelijking om te weten te komen hoeveel iteraties we nodig hebben een beetje te ingewikkeld. We zouden liever een eenvoudigere formule hebben om R te berekenen. Uit de vergelijking die we al hebben, kunnen we afleiden dat $R \leq [\pi/2\theta]$. Dus als we een ondergrens kunnen vinden voor θ vinden we ook een bovengrens voor R. We nemen nu aan dat $M \leq N/2$ dit zorgt ervoor dat:

$$\frac{\theta}{2} \geq sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$$

waaruit we een grens voor R kunnen halen

$$R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$$

We hebben dus $R = O(\sqrt{N/M})$ oproepen van het oracle nodig om tot een oplossing te komen met het Quantum zoekalgoritme. Dit is al een hele een verbetering ten opzichte van het klassieke algoritme dat $O(N/M)$ keer het oracle moet aanspreken. We geven hieronder nog eens een samenvatting van het algoritme voor het geval met één mogelijke oplossing ($M = 1$).

Quantum Search

Input:

1. Black box oracle die de transformatie $O(|x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle$ kan uitvoeren en waarbij $f(x) = 0$ voor alle $0 \leq x \leq 2^n$. Behalve voor x_0 dan geldt $f(x_x) = 1$ (x_0 is de oplossing).

2. $n + 1$qubits in de toestand $|0\rangle$. De extra qubit is de werkruimte van het oracle.

Output:

x_0

Runtime:

$O(\sqrt{2^n})$operaties,

Algoritme:

1. $|0\rangle^{\otimes n} |0\rangle$ Initiële toestand

2. $\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]$ Pas Hadamard-poorten toe op de eerste n qubits en op de laatste qubit

3. $\rightarrow [(2 |\psi\rangle \langle\psi| - I)O]^{\otimes R} \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right] \approx |x_0\rangle \left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]$ Pas de Grover operatie R keer toe waarbij $R \approx \left\lceil \pi\sqrt{2^n}/4 \right\rceil$

4. $\rightarrow x_0$ Meet de eerste n qubits

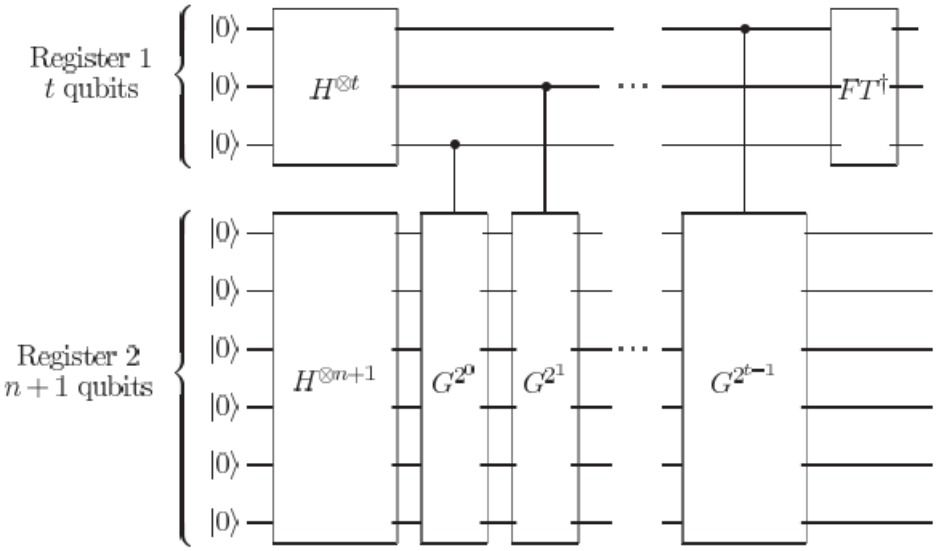
We hebben zojuist aangenomen dat het aantal oplossingen van een zoekprobleem niet groter mag zijn dan het aantal elementen waarin we gaan zoeken. Maar wat gebeurt er met de efficiëntie van het algoritme wanneer we het aantal oplossingen laten toenemen? Uit de vergelijking $R = CI(\frac{arccos\sqrt{M/N}}{\theta})$ zien we dat wanneer M toeneemt ook R zal toenemen. Dit is een rare eigenschap voor een algoritme dat zo snel mogelijk zoekt in een verzameling elementen. We zouden verwachten dat wanneer er meer mogelijke oplossingen zijn, we ook sneller een oplossing vinden. Er zijn twee manieren waarop

we dit probleem kunnen omzeilen. Het eerste is terug de klassieke manier gebruiken waarbij we elk element afgaan en controleren of het een oplossing is. Dit wordt nu effectief, omdat er zoveel mogelijke oplossingen zijn dat we meer dan 1/2 kans hebben, dat een willekeurig element een oplossing is. Het nadeel is dat we in dit geval op voorhand moeten weten of M groot genoeg is.
Als we de grootte van M niet kennen, kunnen we het probleem als volgt oplossen. We verdubbelen gewoon simpelweg het aantal elementen door N elementen toe te voegen die geen oplossing zijn. Dit vergt niet veel werk. We voegen gewoon een extra qubit toe aan de bits die de index weergeven. We moeten ook de oracle aanpassen, zodat deze de extra toegevoegde bit negeert.

9.4 Quantum counting

We hebben een algoritme gevonden voor Quantum zoekproblemen dat werkt in $O(\sqrt{N/M})$. Maar we merkten op dat we het aantal oplossingen M op voorhand moesten kennen om te bepalen hoe vaak we de Grover operatie moesten toepassen. We willen echter ook zoekproblemen oplossen wanneer we niet weten hoe groot M is. Het blijkt dat het op een Quantumcomputer mogelijk is om M te weten te komen, zonder alle elementen te moeten overlopen zoals we op een klassieke computer zouden doen.
We noemen dit het counting probleem, omdat we willen tellen hoeveel oplossingen een zoekprobleem heeft. We maken hiervoor gebruik van het eerder geziene algoritme van faseschatting met behulp van de Quantum Fouriertransformatie en het Grover zoekalgoritme. Niet alleen wordt het mogelijk om zo gebruik te maken van het Quantum zoekalgoritme wanneer we M niet kennen, we kunnen ook testen of een zoekprobleem wel een oplossing heeft door na te gaan of M niet gelijk is aan 0.
Bij het Quantum counting algoritme maken we gebruik van het faseschattingsalgoritme om de eigenwaarden van de Grover operatie te schatten en deze dan te gebruiken om een waarde voor M te bepalen. We hebben twee eigenvectors $|a\rangle$ en $|b\rangle$ van de Grover iteratie in de ruimte bepaald door de vectors$|\alpha\rangle$ en $|\beta\rangle$. We weten dat we de hoek van rotatie door de Grover operatie voorstellen met θ. De eigenwaarden worden dan $e^{i\theta}$ en $e^{i(2\pi-\theta)}$. We nemen ook aan dat we de set van elementen hebben uitgebreid tot $2N$ elementen zoals we hierboven hebben beschreven. We weten dan ook dat $sin^2(\theta/2) = M/2N$.
Het circuit dat we gebruiken om de faseschatting te doen ziet er uit zoals weergegeven op figuur 26.

Het doel van dit circuit is het schatten van de hoek θ met m bits nauwkeurigheid. Het eerste register bevat $t \equiv m + [\log(2 + 1/2\epsilon)]$ qubits, zoals we al



Figuur 26: Quantum counting (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

gedefinieerd hadden in onze beschrijving van het fase schattingsalgoritme. Het tweede register bevat $n+1$ qubits die nodig zijn om het Quantum zoekalgoritme te laten werken. Het circuit werkt in zijn geheel hetzelfde als dat van faseschatting waarbij we de U-poort vervangen door een circuit dat het Quantum zoekalgoritme implementeert. Wanneer we op het einde van dit circuit dan het eerste register meten, krijgen we een schatting van θ of een schatting van $2\pi - \theta$ wat equivalent is. Dit circuit geeft ons dus een schatting voor θ met een nauwkeurigheid van 2^{-m}.
Met behulp van de vergelijking $sin^2(\theta/2) = M/2N$ en onze schatting van θ, kunnen we dan een waarde vinden voor M.
We kunnen nu gebruik maken van dit counting probleem om te testen of een zoekprobleem een oplossing heeft, door na te gaan of de waarde van M die we van het counting probleem krijgen niet gelijk is aan 0. Tevens is het nu mogelijk om zoekproblemen op te lossen waarvan we niet weten hoe groot M is. We voeren hiervoor eerst het counting algoritme uit en gebruiken de gekregen M en θ om daarna te berekenen hoeveel iteraties van de Grover operatie nodig zijn om dit zoekprobleem op te lossen.

9.5 Versnellen van NP-complete problemen

We hebben al verschillende keren herhaald dat het grote voordeel van de Quantum zoekalgoritmes niet de snelheidswinst is, want deze is niet zo spec-

taculair. Wat wel een groot voordeel is zijn de vele toepassingen waarvoor we het Quantum zoekalgoritme kunnen gebruiken. Een van de meest in het oog springende is het versnellen van algoritmes die NP-complete problemen oplossen. In het bijzonder de algoritmes die gebruik maken van een zoekfunctie. Eerder hebben we al een voorbeeld gegeven waar we factoring versnellen met hulp van het Quantum zoekalgoritme. Bij factoring was de snelheidswinst echter niet zo spectaculair omdat we al betere Quantumalgoritmes kennen die niet vertrouwen op een zoekoplossing. We geven nu opnieuw een voorbeeld van een probleem dat we gaan versnellen door gebruik te maken van een Quantum zoekalgoritme. Dit keer gaat het om het Hamiltonpad probleem. Een Hamiltonpad is een pad in een graaf dat elke knoop van een graaf precies één keer bezoekt. Het Hamiltonpad probleem bestaat er uit om te beslissen of het mogelijk is voor een gegeven graaf een Hamiltonpad te vinden. Dit probleem is NP-compleet. Er is op een klassieke computer nog geen algoritme bekend dat dit probleem op een efficiënte manier oplost. Een eenvoudig algoritme dat het Hamiltonpad probleem oplost, is het afgaan van alle mogelijke paden een controleren of dit pad een Hamiltonpad is. Het nagaan of een pad een Hamiltonpad is kan zeer snel gebeuren.
In een graaf met n knopen kan de runtime van dit algoritme $2^{n \log n}$ bedragen als we alle mogelijkheden doorlopen. We kunnen voor de meeste problemen in NP een gelijkaardige oplossing vinden, waarbij we alle mogelijke inputs afgaan en controleren of deze juist zijn.

We kunnen het Quantum zoekalgoritme gebruiken om dit eenvoudige algoritme te versnellen doordat we de snelheid van het zoeken versnellen. In dit geval maken we gebruik van het counting algoritme om na te gaan of er een M bestaat die niet gelijk is aan 0. Als dit zo is weten we dat er een oplossing bestaat voor het probleem. Om dit circuit te laten werken moeten we een oracle definiëren die als operatie het volgende uitvoert:

$$O(|v_1, v_2, \ldots, v_n\rangle) = \begin{cases} |v_1, ..., v_n\rangle & \text{Als } v_1, ..., v_n \text{geen Hamiltonpad is} \\ -|v_1, ..., v_n\rangle & \text{Als } v_1, ..., v_n \text{een Hamiltonpad is} \end{cases}$$

Zo een oracle is makkelijk te maken wanneer we over de graaf beschikken. We maken een functie$f(x)$, die 1 geeft als x een Hamiltonpad is voor de graaf en 0 in het andere geval. We creëren dan een oracle met deze functie zoals we al eerder hebben gedaan. Het oracle zal dan in polynomiale tijd werken want het controleren of een pad een Hamiltonpad is kan in polynomiale tijd. Dit oracle gebruiken we dan om het counting circuit mee te bouwen. Zo kunnen we nagaan of er een oplossing bestaat als M niet gelijk is aan 0. Wanneer we vinden dat er een oplossing is voor dit probleem, dan kunnen we hetzelfde oracle gebruiken en de gevonden M om het Quantum

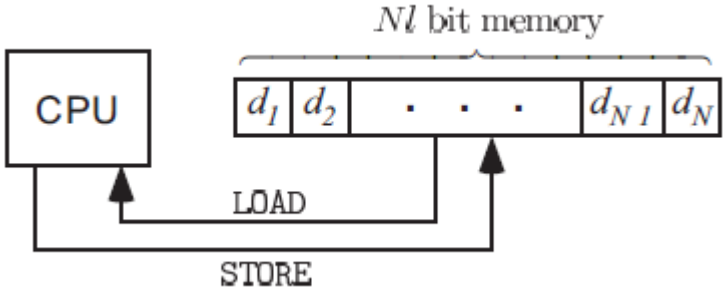
zoekalgoritme de oplossing te laten vinden. We kunnen in $O(2^{n(\log\ n)/2})$ nagaan of een graaf over een Hamiltonpad beschikt. Het zoeken van de Hamiltonpaden wanneer we weten hoeveel oplossingen er zijn, gaat in $O(\sqrt{X})$. Waarbij X de tijd is die het klassieke algoritme nodig heeft. Dit is dus geen grote verbetering. Een probleem dat een exponentiële oplossing heeft van de vorm $O(2^n)$ zal met behulp van het Quantum zoekalgoritme een versnelling hebben tot $O(2^{n/2})$ wat nog steeds exponentieel is.

9.6 Quantum zoeken in een ongestructureerde database

Niet alleen bij het versnellen van algoritmes kan het Quantum zoekalgoritme helpen, ook bij het versnellen van zoekopdrachten in een ongestructureerde database kan dit algoritme zorgen voor snelheidswinsten. Vooral omdat databases vaak grote hoeveelheden data bevatten. We richten ons tot ongestructureerde databases, omdat we bij databases die wel gestructureerd zijn dit feit kunnen uitbuiten om zo met verschillende algoritmes snel te zoeken. Stel dat we een database hebben van 1000 elementen die niet geordend zijn en we willen zoeken naar één van deze elementen, dan zouden we op een klassieke computer alle elementen moeten afgaan tot we het element dat we zoeken tegenkomen. In het beste geval komen we dit element vlug tegen, maar in het slechtste geval is het element dat we zoeken het allerlaatste in de rij. We zeggen dat we gemiddeld $N/2$ elementen zullen afgaan, voordat we een resultaat vinden.
We kunnen een Quantum zoekalgoritme gebruiken om dit proces te versnellen, maar we moeten hier wel voldoen aan een aantal vereisten.

Stel dat we over een ongestructureerde database beschikken met $N \equiv 2^n$ elementen die een lengte hebben van l bits. We geven elk element een label van $d_1, ..., d_n$. We willen nu nagaan of een element s van lengte l bits in onze database zit. Een klassieke computer die dit probleem oplost is typisch in twee delen gesplitst. Aan de ene kant hebben we de CPU die de data behandelt en verwerkt. De CPU beschikt maar over een klein geheugen dat niet blijvend is. De tweede kant is een groot geheugen waarin we de database opslaan in 2^n blokken van l - bits. Het geheugen kan de data niet zelf verwerken die het opslaat. De CPU kan niet de hele database in een keer bekijken en maakt gebruik van twee operaties die data uit het geheugen halen en in het geheugen plaatsen. Deze operaties noemen we LOAD en STORE. (figuur 27)

Wanneer we nu zo een klassieke computer gebruiken om het element s te zoeken in onze database, zal de CPU elk element één voor één opvragen uit het geheugen en nagaan of het overeenkomt met het element s waarnaar we op zoek zijn. Als het element dat we hebben opgevraagd het te zoeken



Figuur 27: Load & Store (Bron: Quantum Computation and Quantum Computing - M. Nielsen, I. Chuang)

element was, geven we dit als resultaat en stoppen we met zoeken. Is dit niet het geval dan blijven we doorgaan tot we alle elementen hebben gehad en geven dan het lege resultaat terug. Het is makkelijk in te zien dat deze manier $O(2^n)$ stappen zal nodig hebben om een resultaat te vinden. Ook is het niet mogelijk om een efficiënter algoritme te vinden op een klassieke computer.

We gaan nu een analoog algoritme ontwikkelen dat gebruik maakt van een Quantum circuit om zo een snelheidswinst te krijgen. Ook hier splitsen we de computer op in een CPU en een geheugen.
We nemen aan dat de CPU bestaat uit vier registers van qubits:

1. Een index register van n qubits die starten in toestand $|0\rangle$.
2. Een register van l qubits die starten in toestand $|s\rangle$ en voor de verdere uitvoering ook in deze toestand blijven.
3. Een data register van l qubits dat start in toestand $|0\rangle$.
4. Een qubit die start in toestand $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

Het geheugen kunnen we op twee manieren voorstellen. Als eerste kunnen we een Quantumgeheugen hebben met N cellen van l qubits waarin de database elementen $|d_x\rangle$ zijn opgeslagen. De tweede manier maakt gebruik van een klassiek geheugen van N cellen van l bits waarin we de database elementen d_x opslaan. Er is echter een verschil met een klassieke database. Deze elementen kunnen geadresseerd worden met een waarde x die een superpositie kan zijn van verschillende waardes. Deze Quantumindex laat ons toe om een superpositie van elementen te loaden in de CPU. Dit werkt als volgt wanneer het index-register in een toestand $|x\rangle$ is en het data-register in een toestand $|d\rangle$ is, dan wordt de inhoud van element d_x die in geheugen cell

x zit, toegevoegd aan het data-register met de volgende operatie wanneer we een load uitvoeren:

$$|d\rangle \rightarrow |d \oplus d_x\rangle$$

Wanneer we nu gaan zoeken in deze database met behulp van een Quantum circuit moeten we eerst een oracle hebben dat als taak heeft de index x te flippen, als deze de index is waar we het element s kunnen vinden. Stel dat de CPU zich in de volgende toestand bevindt:

$$|x\rangle\,|s\rangle\,|0\rangle\,\frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Als we nu een load operatie uitvoeren krijgen we de toestand:

$$|x\rangle\,|s\rangle\,|d_x\rangle\,\frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Het oracle vergelijkt nu het tweede en derde register en flipt het vierde register, wanneer register twee en drie overeenkomen.

$$|x\rangle\,|s\rangle\,|d_x\rangle\,\frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad \rightarrow \quad \begin{cases} -\,|x\rangle\,|s\rangle\,|d_x\rangle\,\frac{|0\rangle+|1\rangle}{\sqrt{2}} & \text{als } d_x = s \\ |x\rangle\,|s\rangle\,|d_x\rangle\,\frac{|0\rangle+|1\rangle}{\sqrt{2}} & \text{als } d_x \neq s \end{cases}$$

We voeren nu opnieuw een load uit om het data-register terug in de toestand $|0\rangle$ te plaatsen. We kunnen het oracle hierboven beschreven, gebruiken om een Quantum zoekopdracht uit te voeren, om s te vinden in de database. Het zoeken van s zal dan $O(\sqrt{N})$ operaties vragen wat een snelheidswinst is vergeleken met de $O(N)$ operaties die de klassieke computer nodig heeft. We kunnen dus zorgen voor een snelheidswinst bij het zoeken in een ongestructureerde database, maar in de realiteit komen zo databases zeer weinig voor. Heeft het dan nog nut om een Quantum zoekalgoritme te gebruiken? Het antwoord op deze vraag zal vaak nee zijn. Een database is vaak zo gestructureerd dat wanneer we er in gaan zoeken we deze eigenschap kunnen gebruiken om zeer snel een resultaat te vinden. Bijvoorbeeld het zoeken van een string in een database die alfabetisch geordend is heeft maar $O(log\ N)$ operaties nodig. Alleen in uitzonderlijke gevallen van complexe databases waarop we ingewikkelde query's gaan uitvoeren, kan het zijn dat we de structuur van de database niet kunnen gebruiken en lijkt het dus dat we in een ongestructureerde database zoeken. Maar zelfs in dat geval is het voorlopig eenvoudiger om klassieke computers gedistribueerd te laten werken in plaats van een Quantumcomputer te gebruiken. Vooral omdat deze

voorlopig nog te duur en ingewikkeld zijn. Hierdoor lijkt het dat het zoeken in databases niet de grootste troef gaat zijn van de Quantumcomputer. En zullen we Quantum zoekalgoritmes eerder gaan gebruiken om problemen in NP sneller op te lossen.

9.7 Limieten van Quantum zoeken

We kunnen bewijzen dat de $O(\sqrt{N})$ operaties die we nodig hebben om te zoeken met behulp van Quantumcomputers een optimale waarde zijn en we geen Quantumalgoritme kunnen vinden dat sneller zoekt. Dit bezorgt ons tezelfdertijd opluchting en teleurstelling. Opluchting omdat we weten dat we het maximale uit de Quantum Computing hebben gehaald wat betreft zoeken, en we zeker weten dat er geen verbetering mogelijk is. Teleurgesteld echter omdat deze snelheidswinst niet zo spectaculair is als we gehoopt hebben. We hadden misschien op een spectaculair zoekalgoritme gehoopt dat zou werken in $O(\log\ N)$ operaties. Wat dan zou betekenen dat we problemen uit NP efficiënt zouden kunnen oplossen door gebruik te maken van Quantum zoekalgoritmes. Dit geeft van de ene kant aan dat wanneer we een NP-probleem proberen op te lossen met een naïef algoritme dat alle oplossingen afgaat, we op voorhand al weten dat dit niet efficiënt gaat zijn. Als er voor problemen in NP geen andere algoritmes kunnen worden gevonden dan deze zoekalgoritmes, dan is dit slecht nieuws voor Quantum Computing want dit zou dan betekenen dat BQP geen problemen uit NP-compleet bevat.
We zijn hier echter nog niet zeker van en hopen dat we net als bij factoring en order finding een algoritme vinden dat de problemen uit NP haalt en in BQP plaatst. Hierin schuilt momenteel één van de grootste uitdagingen in Quantum Computing en klassieke computing. Het vinden van verborgen systemen die ons toelaten de Quantumcomputer en klassieke computer te gebruiken om problemen efficiënt op te lossen. Voorlopig ziet het er naar uit dat we met Quantumcomputer meer problemen kunnen oplossen, maar zoals al meerdere keren aangehaald is het zeer moeilijk om hiervoor een sluitend bewijs te vinden.

10 Realisaties in Quantum Computing

Tot nu toe hebben we het over de Quantumcomputer gehad in een puur theoretische context, vooral wanneer we algoritmes uitlegden. Maar hoe zit het met praktische Quantumcomputers? Hoelang gaat het nog duren voordat we deze algoritmes echt gaan kunnen gebruiken? Het idee van Quantumcomputers werd voor het eerst voorgesteld in de jaren 1980 door wetenschappers als Richard Feynman en David Deutsch. Op dat moment was het nog niet echt duidelijk waarvoor men Quantumcomputers zou kunnen gaan gebruiken, buiten voor het uitvoeren van Quantumsimulaties.
In 1994 ontdekte Peter Shor een algoritme dat op een Quantumcomputer het factoring probleem efficiënt kon oplossen en zelfs een exponentiële versnelling had ten opzichte van het klassieke algoritme. Vanaf dat moment was het duidelijk dat Quantumcomputers over een enorme kracht beschikken die we kunnen gebruiken om betere algoritmes te vinden voor problemen. Een andere reden waarom men nu plots wel geïnteresseerd was, komt uit het feit dat de meeste cryptosystemen berusten op het feit dat factoring en andere gelijkaardige problemen niet efficiënt op te lossen zijn. Daar kwam opeens (theoretisch) verandering in.
We kunnen dus gerust stellen, dat er ongeveer al 20 jaar gewerkt wordt aan een praktische Quantumcomputer. Toch is men op de dag van vandaag nog niet veel verder geraakt dan een Quantumcomputer die bestaat uit een klein aantal qubits. Waarom is het dan toch zo moeilijk om een Quantumcomputer te maken? Als eerste is het zeer moeilijk om qubits te creëren. Het wordt nog moeilijker als we deze qubits in een superpositie willen brengen en in deze superpositie houden. We hebben al gezegd dat wanneer we de qubits observeren ze de superpositie verlaten en een waarde aannemen 0 of 1. Maar dit gebeurt niet alleen wanneer we een qubit observeren, ook wanneer de qubit in contact komt met de buitenwereld verlaat hij zijn superpositie. We moeten de qubits dus afschermen van lucht, warmte, energie van buitenaf, ... Verder is het ook niet genoeg om alleen maar deze qubits te creëren: we willen er ook nog operaties op uitvoeren om zo berekeningen te maken en we willen de oplossing van deze berekening te weten komen. Dit allemaal zonder dat we de superpositie van de qubits verstoren en contact maken met de buitenwereld.
Als we kijken naar het aantal qubits waaruit een hedendaagse Quantumcomputer bestaat lijkt het alsof er weinig tot geen voortgang is geweest, terwijl dit helemaal niet zo is. Het aantal qubits waar men mee werkt is nog niet sterk toegenomen. Maar wel de kwaliteit van de qubit die men kan produceren. We kunnen tegenwoordig steeds beter de gecreëerde qubits beheersen. Onderzoekers gaan er momenteel van uit, dat hoe beter we de qubits kunnen beheersen, hoe makkelijker het wordt om het systeem uit te breiden naar meer qubits.

10.1 Ontdekkingen

Het is niet alleen enorm moeilijk om een Quantumcomputer te creëren. Men is er ook nog altijd niet uit wat de beste manier is om een Quantumcomputer te maken en hoe we het best qubits fysiek voorstellen. Verschillende onderzoekers hebben andere meningen over wat het beste zou werken en onderzoeken dus in verschillende richtingen hoe ze een praktische Quantumcomputer kunnen bouwen. We sommen hieronder in het kort enkele ontdekkingen op waarmee we een beeld proberen te geven van de evolutie van de Quantumcomputer.

- 1998: Onderzoekers van Los Alamos en MIT zijn er in geslaagd om één qubit te verspreiden over de drie verschillende spins binnen een molecule alanine of trichloroethylene met behulp van kernspinresonantie (NMR Nuclear Magnetic Resonance). Deze NMR kunnen we dan makkelijk uitlezen met behulp van instrumenten zoals deze ook voorkomen in ziekenhuizen (bv.: MRI). Door deze uitspreiding is de qubit makkelijker te behandelen. Door de qubit uit te spreiden konden de onderzoekers gebruik maken van entanglement om de qubit beter te onderzoeken [26].

- 2000: Onderzoekers van Los Alamos slagen er in om een Quantumcomputer te realiseren met zeven qubits in een druppel vloeistof. Er wordt opnieuw gebruik gemaakt van NMR om de qubits te manipuleren [7].

- 2000: Onderzoekers van IBM realiseren een Quantumcomputer met vijf qubits, door gebruik te maken van de kernen van vijf atomen fluor. Deze Quantumcomputer wordt geprogrammeerd via radiogolven en de resultaten uitgelezen via NMR-instrumenten. Onder leiding van Dr. Isaac Chuang slaagt dit team er in om een order finding algoritme uit te voeren op hun Quantumcomputer [16].

- 2001: Onderzoekers van IBM gebruiken een Quantumcomputer met zeven qubits (NMR) om Shor's algoritme uit te voeren om het getal 15 succesvol te factoren in 3 en 5 [32].

- 2007: Het bedrijf D-Wave stelt een Quantum chip voor met 16 qubits. De chip wordt voorgesteld door een sudoku op te lossen en nog enkele andere problemen. De wetenschappelijke wereld is eerder kritisch en niet overtuigd van deze Quantumchip.

- 2009: Onderzoekers aan de universiteit van Yale maken de eerste solid state Quantumprocessor die zeer eenvoudige berekeningen kan uitvoeren. De twee qubits werden elk voorgesteld door miljoenen aluminium atomen die zich samen gedroegen als één atoom [6].

- 2009: Aan de universiteit van Bristol wordt een silicon-based Quantumchip ontwikkeld, gebaseerd op Quantumoptics. Met behulp van deze chip is het mogelijk Shor's algoritme uit te voeren. In 2010 volgt er nog een verbetering hier op [24].
- 2011: Een team van wetenschappers uit Japan en Australië voeren een succesvolle Quantumteleportatie uit op een set Quantumdata [11].
- 2011: D-Wave kondigt de eerste commerciële Quantumcomputer aan de: D-Wave One. Deze zou beschikken over een chip met 128 qubits. Lockheed Martin is het eerste bedrijf dat zo een D-Wave One kopen [21, 9].
- 2011: Onderzoekers aan de universiteit van Bristol slagen er in een iteratieve versie van Shor's algoritme te laten werken en met behulp hiervan factoren ze succesvol het getal 21 [14].
- 2011: Er wordt een prototype ontwikkeld van een Quantumcomputer volgens de Von Neuman architectuur met een Quantum CPU en een Quantum RAM-geheugen [13].
- 2011: Onderzoekers gebruiken 4 qubits om het getal 143 succesvol te factoren [34].
- 2012: IBM beweert een belangrijke doorbraak te hebben gemaakt op het vlak van het bouwen van een Quantumcomputer door gebruik te maken van supergeleide integrated circuits [23].
- 2012: Een team van onderzoekers van verschillende universiteiten, waaronder de TU Delft, slagen er in een Quantumcomputer te maken met 2 qubits op een diamant kristal, door inbreng van een onzuiverheid. Deze Quantumcomputer blijkt goed schaalbaar en zou werken op kamertemperatuur. Het is mogelijk het Grover zoekalgoritme succesvol uit te voeren van de eerste keer met een kans van 95% [31].
- 2012: Australische wetenschappers beweren dat het binnen 5 jaar mogelijk moet zijn Quantumcomputers te bouwen, omdat ze de eerste werkende qubit op een atoom in silicium hebben vervaardigd. Ze beweren dat we nu op een gelijkaardige manier Quantumcomputers kunnen maken als momenteel al kan met klassieke computers [22].
- 2013: De TU Delft opent een groot project, waarmee ze hopen de eerste praktische Quantumcomputer te kunnen bouwen door verschillende manieren om qubits te creëren tegelijk te bestuderen [33].
- 2013: Google en NASA kopen samen een D-Wave Two. De D-Wave Two is de tweede commerciële Quantumcomputer en zou beschikken

over 512 qubits. Google kondigde ook aan dat het onderzoekers zou uitnodigen om de D-Wave Two uit te testen [17].

- 2014: Edward Snowden brengt allerlei geheimen over de NSA aan het licht, waaronder een onderzoeksprogramma (Breaking Hard Targets) naar Quantumcomputers met als hoofddoel het breken van huidige encryptiesystemen [28].

10.2 D-Wave

Wanneer we naar deze lijst kijken, zien we dat de meeste doorbraken systemen zijn van enkele qubits. Er is echter één naam die er een paar keer uitspringt met veel grotere getallen, namelijk het bedrijf D-Wave. D-wave beweert te beschikken over de eerste commerciële Quantumcomputer met maar liefst 512 qubits. Natuurlijk waren de meeste wetenschappers hier een beetje sceptisch over. In dit deel gaan we daarom hier iets dieper op in.

D-Wave is een Canadees bedrijf dat zich specialiseert in het bouwen van commerciële Quantumcomputers. D-Wave maakt gebruik van SQUIDs (Superconducting Quantum Interference Device) als bouwblokken van hun Quantumcomputers. Een SQUID is een ring gemaakt van Niobium. Wanneer deze ring sterk wordt afgekoeld (ongeveer 20 milliKelvin) wordt hij een supergeleider. De stroom die hier doorgaat maakt een magnetisch veld, dat normaal omhoog of omlaag gericht is. Maar wanneer de SQUID een supergeleider wordt, gebeuren deze twee magnetische velden tegelijk in een superpositie [30]. De Quantumcomputer die D-Wave maakt, is echter niet gebaseerd op de Quantumpoorten zoals wij ze uitgelegd hebben. D-Wave maakt gebruik van adiabatic anaeling. Dit zorgt ervoor dat deze Quantumcomputers makkelijker zijn om te maken, maar beperkt ze ook sterk in hun bruikbaarheid. De Quantumcomputers van D-Wave kunnen alleen maar gebruikt worden om zeer specifieke problemen op te lossen, namelijk optimaliseringsproblemen. Optimaliseringsproblemen komen echter zeer vaak voor en kunnen moeilijk op te lossen zijn op klassieke computers. Er was dus onmiddelijk een markt voor deze vorm van Quantumcomputers. Deze Quantumcomputers kunnen echter niet gebruikt worden om algoritmes uit te voeren zoals wij ze hierboven beschreven hebben.

Annealing werkt op een klassieke computer als volgt. We vertalen het probleem via een wiskundige operatie in een landschap met bergen en dalen. We gaan nu op zoek naar het laagste dal en dit stelt de ideale toestand voor die we willen vinden. Klassieke algoritmes vinden echter vaak diepe dalen waar ze moeilijk uit geraken en beschouwen deze dan als de oplossing. We kunnen echter de algoritmes aanpassen om hier toch mee overweg te kunnen. Maar deze algoritmes hebben dan meer tijd nodig om een oplossing te vinden. Dit

kan oplopen tot een exponentiële kost. Wanneer we echter gebruik maken van Quantum annealing op een machine van D-Wave, kunnen we gebruik maken van Quantumeffecten zoals Quantumtunneling en entanglement om uit deze dalen te geraken en sneller juiste oplossingen te vinden. Deze uitleg is eerder wat vaag omdat D-Wave de details van hun werkwijze liever geheim houdt.

10.2.1 Orion Prototype

In 2007 stelt D-Wave hun eerste prototype van een Quantumcomputer voor met 16 qubits. Ze maken gebruik van drie problemen om de kracht van hun Quantumcomputer te tonen. Het eerste probleem is het oplossen van een Sudoku. Als tweede lossen ze een probleem op van het plaatsen van mensen aan een tafel, waarbij rekening moet gehouden worden wie wel of niet naast elkaar mag of moet zitten. Het laatste probleem dat werd gebruikt is het zoeken van een molecule in een database van moleculen. Het oplossen van deze problemen was echter geen grote prestatie op zich, want een klassieke computer zou deze opdrachten ook zonder moeite kunnen uitvoeren. Maar deze voorstelling werd eerder gebruikt om aan te tonen dat ze een werkende Quantumcomputer konden bouwen.

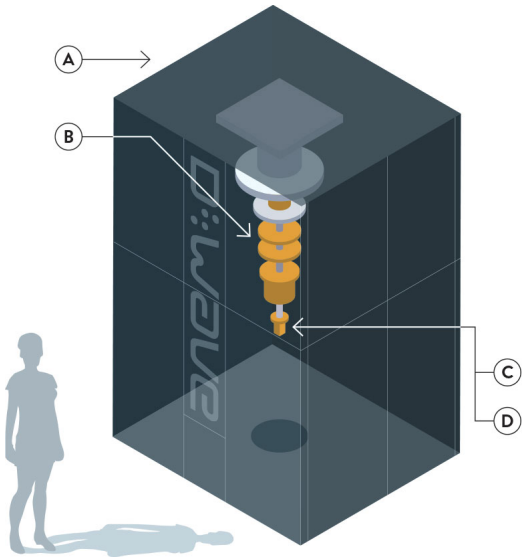
10.2.2 D-Wave One

In 2011 lanceert D-Wave de eerste commerciële Quantumcomputer met een 128 qubits processor. Deze Quantumcomputer kan een taak uitvoeren, namelijk discrete optimalisatie en maakt hiervoor gebruik van Quantum annealing. De D-Wave One is letterlijk en figuurlijk een black box. Zie figuur 28.

Lockheed Martin, een luchtvaartbedrijf onder meer bekend van de F-16, leaset de D-Wave One voor $10.000.000 om hun te helpen met optimalisatieproblemen in hun vluchtcontrole-software. Lockheed Martin laten hun pas verkregen Quantumcomputer testen door een team van wetenschappers onder leiding van Matthias Troyer en Daniel Lidar. Deze besluiten dat er zeker sprake is van een vorm van Quantum annealing in kleinere mate maar zeker geen 128 qubits. Ook merken ze op dat de D-Wave One niet sneller is dan gespecialiseerde algoritmes op klassieke computers [30].

10.2.3 D-Wave Two

In 2012 komt D-Wave met een opvolger voor hun eerste Quantumcomputer en lanceren de D-Wave Two met een 512 qubit Quantumprocessor. Google



A. Deep Freezer A massive refrigeration system uses liquid helium to cool the D-Wave chip to 20 millikelvin—or 150 times colder than interstellar space.
B. Heat Exhaust Gold-plated copper disks draw heat up and away from the chip to keep vibration and other energy from disturbing the quantum state of the processor.
C. Niobium Loops A grid of hundreds of tiny niobium loops serve as the quantum bits, or qubits, the heart of the processor. When cooled, they exhibit quantum-mechanical behavior.
D. Noise Shields The 190-plus wires that connect the components of the chip are wrapped in metal to shield against magnetic fields. Just one channel transmits information to the outside world—an optical fiber cable.

Figuur 28: D-Wave One (Bron: `http://www.wired.com/2014/05/quantum-computing/`)

is wel geïnteresseerd in deze machine, maar is nog niet helemaal overtuigd, na de D-Wave One, die volgens de tests niet zo een grote versnelling brengt en ook op Quantumvlak tegenvalt. D-Wave laat daarom tests uitvoeren door Catherine McGeoch, die de D-Wave Two test ten opzichte van klassieke computers, die algemene optimalisatie software pakketten draaien voor optimalisatie [15]. Uit deze tests blijkt dat de Quantumcomputer maar liefst 3600 keer sneller is. Er wordt wel bij vermeld dat deze test niet helemaal eerlijk is. Men test een gewone computer tegen een machine speciaal ontworpen voor optimalisatie. In 2013 vinden onderzoekers in Londen bewijs dat er in de chips van D-Wave Quantumentaglement plaatsvindt, maar wel maar op kleine schaal. Later dat jaar besluiten Google en NASA om samen de D-Wave Two aan te kopen voor $15.000.000 .

10.2.4 Sceptici

Zoals eerder gezegd waren er veel wetenschappers, die bezig waren met Quantumcomputing, die zeer kritisch keken naar de aankondigingen van D-Wave. Zeker omdat hun Quantumprocessors zo ver voor stonden op hetgeen de meeste onderzoekers deden. D-Wave beweerde over Quantumcomputers te beschikken met 512 qubits, terwijl de meeste onderzoekers problemen hadden met een handvol qubits stabiel te houden. Na de demo van hun eerste prototype zei Scott Aaronson, Een MIT-proffessor op het vlak van theoretische informatica en Quantum computing en ook de schrijver van het boek “Quantum Computing Since Democritus”[1], dat de demo niks bewees over het al dan niet optreden van Quantumeffecten in hun computer. Nadat de tests uitgevoerd waren op de D-Wave One was hij meer overtuigd [12]. Maar na de aankondiging van de D-Wave Two en het resultaat van de tests uitgevoerd door McGeoch, zegt Aaronson weer dat de claims die door D-Wave gemaakt worden, overdreven zijn. Hij gaat verder en zegt dat de D-Wave niet echt een Quantumcomputer is, omdat men de qubits niet kan beheersen: het is eerder een klassieke computer waar af en toe Quantumeffecten in optreden. Ze nemen qubits die moeilijk te beheersen zijn en bouwen hier dan grote computers mee, terwijl de meeste onderzoekers proberen de weinige qubits die ze creëren in controle te houden.
In de jaren na de lancering van de D-Wave One en D-Wave Two worden verschillende tests uitgevoerd op deze machines. Zowel Lockheed Martin [4] en Google [29] laten onderzoekers toe om te achterhalen wat hun D-Wave computers nu precies kunnen. Uit deze test blijken twee dingen. Ten eerste is men het er vrijwel over eens dat er Quantumeffecten optreden in de machines, maar deze lijken niet beheersbaar. Ten tweede vindt men dat de D-Wave Quantumcomputers niet sneller zijn dan gewone computers met gespecialiseerde algoritmes. In sommige gevallen wordt de Quantumcomputer zelfs verslagen in snelheid door de klassieke computer.

10.2.5 Google

Nadat Google in samenwerking met NASA de D-Wave Two had gekocht, hebben ze onderzoekers van over de hele wereld toegelaten om tests uit te voeren op de Quantumcomputer om uit te zoeken wat ze nu precies gekocht hadden. Een Quantumcomputer of een heel dure klassieke computer. In het begin van 2014 heeft Google de resultaten van deze test vrijgegeven en samengevat in een blog post [29]. Hieronder volgt een korte samenvatting waarin we goed kunnen zien waar de D-Wave toe in staat is.
De tests bestaan erin om een optimalisatieprobleem op te lossen op verschillende klassieke systemen en op de D-Wave Two en de resultaten te vergelijken. Niet de snelheid waarmee de oplossing wordt berekend is hierbij het belangrijkste. Er wordt meer aandacht geschonken aan de toename in rekentijd als het aantal variabelen stijgt.
De eerste test bestond erin de machine te testen ten opzichte van de algemene software pakketten, die ook gebruikt werden door McGeoch [15]. Uit deze test bleek dat de D-Wave Two veel sneller tot een oplossing komt dan de klassieke computers. In tegenstelling tot de versnelling van 3600 keer stellen ze bij deze tests vast dat de Quantumcomputer zelfs 35.000 keer sneller is dan de klassieke computers. Zoals eerder gezegd zijn deze tests niet echt eerlijk, omdat ze de algemene algoritmes tegenover een gespecialiseerde machine plaatsen.
Voor de tweede tests worden er twee teams ingezet die gespecialiseerde software schrijven. Het eerste team onder leiding van Matthias Troyer schrijft een optimalisatieprogramma dat gebruik maakt van klassieke simulated annealing en werkt op GPU's. Het tweede programma wordt geschreven door Alex Selby, bekend van het vinden van de oplossing voor de Eternity puzzle. Wanneer deze programma's werden gebruikt in de tests, merkten ze dat de Quantumcomputer niet meer duidelijk de snelste was. Sommige problemen werden sneller of even snel opgelost met de klassieke computers. Andere problemen waren sneller opgelost met de Quantumcomputer. Het werd ook duidelijk dat willekeurige problemen niet sneller opgelost werden door de Quantumcomputer. Maar problemen waarbij een structuur aanwezig, was bleken een voordeel te geven voor de Quantumcomputer. Men veronderstelt dat dit komt doordat de Quantumeffecten dan duidelijker een voordeel bieden. Maar wanneer we de verschillende programma's groeperen in een portfolio merken we dat de beste oplossing uit de portfolio de Quantumcomputer kan bijhouden. Meer zelfs, de Quantumcomputer wordt in sommige problemen verslagen door software die op een gewone Intel desktop werkt. Nog drastischer was het resultaat dat wanneer het aantal variabelen toenam, ook de tijd die nodig was om een oplossing te vinden op de Quantumcomputer in dezelfde mate meegroeit als op de klassieke computers. Volgens de

onderzoekers was de reden hiervoor dat de qubits in de D-Wave Two niet stabiel genoeg zijn om hier voordeel uit te halen. Met andere woorden de Quantumeffecten die voor de versnelling moeten zorgen, kunnen niet ten volste worden gebruikt. Een nadeel dat volgt uit het feit dat de qubits niet lang genoeg in hun superpositie blijven bestaan, is dat ze nooit in grote groepen met elkaar verbonden geraken en ook moeilijk entangled raken.
Bij Google zijn ze echter positief en kijken uit naar de volgende stap in hardware. Die waarschijnlijk meer zal kunnen op het vlak van Quantumeffecten om problemen op te lossen. Ook duiden ze aan dat de tests die ze gedaan hebben, eerder beperkt waren, maar dat ze steeds meer resultaten blijven verzamelen. Het blijkt dat de huidige hardware nog niet in staat is om de klassieke programma's te verslaan in de meeste gevallen. Daarom wordt er nu ook gezocht naar de problemen waarvoor de huidige hardware wel sneller is. Om zo te proberen een set van problemen te identificeren met een bepaalde eigenschap, waarvoor het gebruik van de Quantumcomputer wel een versnelling oplevert.

Referenties

[1] Scott Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of mathematics*, pages 781–793, 2004.

[3] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *in Proc. 25th Annual ACM Symposium on Theory of Computing, ACM*, pages 11–20, 1993.

[4] Sergio Boixo, Tameem Albash, Federico M Spedalieri, Nicholas Chancellor, and Daniel A Lidar. Experimental signature of programmable quantum annealing. *Nature communications*, 4, 2013.

[5] MIT Open courseware. `http://ocw.mit.edu/courses/mathematics/18-335j-introduction-to-numerical-methods-fall-2004/lecture-notes/lecture2.pdf`. Last retrieved 11/06/2014.

[6] L DiCarlo, JM Chow, JM Gambetta, Lev S Bishop, BR Johnson, DI Schuster, J Majer, A Blais, L Frunzio, SM Girvin, et al. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature*, 460(7252):240–244, 2009.

[7] Joseph D Dumas II. *Computer architecture: fundamentals and principles of computer design*. CRC Press, 2005.

[8] Juris Hartmanis. On the weight of computations. *EATCS Bulletin*, 55:136–138, 1995.

[9] MW Johnson, MHS Amin, S Gildert, T Lanting, F Hamze, N Dickson, R Harris, AJ Berkley, J Johansson, P Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[10] Richard E Ladner. On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22(1):155–171, 1975.

[11] Richard Lai. First light wave quantum teleportation achieved, opens door to ultra fast data transmission. `http://www.engadget.com/2011/04/18/first-light-wave-quantum-teleportation-achieved-opens-door-to-u/`, 18 April 2011. Last retrieved 11/06/2014.

[12] Timothy B. Lee. Confused about the NSA's quantum computing project? this MIT computer scientist can explain. *The Washington Post*, 2 January 2014.

[13] Matteo Mariantoni, H Wang, T Yamamoto, M Neeley, Radoslaw C Bialczak, Y Chen, M Lenander, Erik Lucero, AD O'connell, D Sank, et al. Implementing the quantum von Neumann architecture with superconducting circuits. *Science*, 334(6052):61–65, 2011.

[14] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776, 2012.

[15] Catherine C McGeoch and Cong Wang. Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers*, page 23. ACM, 2013.

[16] Matthew McMahon Michael Ross. IBM-led team demonstrates most-advanced quantum computer. Technical report, IBM, 2000.

[17] Hartmut Neven. Launching the quantum artificial intelligence lab. `http://googleresearch.blogspot.co.uk/2013/05/launching-quantum-artificial.html`, 16 May 2013. Last retrieved 11/06/2014.

[18] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2000.

[19] University of Delaware. `http://www.physics.udel.edu/~msafrono/650/Lecture%2013.pdf`. Last retrieved 11/06/2014.

[20] Georgia Tech School of Mathematics. `http://people.math.gatech.edu/~jeanbel/4782/Year06/Homework06/problem206.pdf`. Last retrieved 11/06/2014.

[21] Robert Perkins. Operational quantum computing center established at usc. `http://viterbi.usc.edu/news/news/2011/operational-quantum-computing334119.htm`, 29 October 2011. Last retrieved 11/06/2014.

[22] Jarryd Pla. Breakthrough in bid to create first quantum computer. `https://newsroom.unsw.edu.au/news/technology/breakthrough-bid-create-first-quantum-computer`, 20 September 2012. Last retrieved 11/06/2014.

[23] Damon Poeter. IBM says it's on the cusp of building a quantum computer. `http://www.pcmag.com/article2/0,2817,2400930,00.asp`, 28 February 2012. Last retrieved 11/06/2014.

[24] Alberto Politi, Jonathan CF Matthews, and Jeremy L O'Brien. Shors quantum factoring algorithm on a photonic chip. *Science*, 325(5945):1221–1221, 2009.

[25] Carl Pomerance. The number field sieve. In *Proceedings of Symposia in Applied Mathematics*, volume 48, pages 465–480, 1994.

[26] Vishal Sahni. *Quantum computing*. Tata McGraw-Hill Education, 2007.

[27] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.

[28] Barton Gellman Steven Rich. NSA seeks to build quantum computer that could crack most types of encryption. *The Washington Post*, 2 January 2014.

[29] Google Quantum A.I. Lab Team. Where do we stand on benchmarking the D-Wave 2? `https://plus.google.com/+QuantumAILab/posts/DymNo8DzAYi`, 20 January 2014. Last retrieved 11/06/2014.

[30] Clive Thompson. The revolutionary quantum computer that may not be quantum at all. `http://www.wired.com/2014/05/quantum-computing/`, 20 May 2014. Last retrieved 11/06/2014.

[31] T Van der Sar, ZH Wang, MS Blok, H Bernien, TH Taminiau, DM Toyli, DA Lidar, DD Awschalom, R Hanson, and VV Dobrovitski. Decoherence-protected quantum gates for a hybrid solid-state spin register. *Nature*, 484(7392):82–86, 2012.

[32] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.

[33] Jos Wassink. Tu richt zich op quantumcomputer. `http://www.delta.tudelft.nl/artikel/tu-richt-zich-op-quantumcomputer/27244`, 3 October 2013. Last retrieved 11/06/2014.

[34] Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum factorization of 143 on a dipolar-coupling NMR system. *arXiv preprint arXiv:1111.3726*, 2011.

A Artikel

Quantum Computing

De dag van vandaag gaat men ervan uit en vindt men het haast vanzelfsprekend dat onze computers steeds sneller worden. Alsook dat we meer en meer problemen kunnen oplossen door gebruik te maken van informatica. Toch bereiken we steeds sneller de grens van wat mogelijk is. De computer en ook de computerchip hebben een immens snelle evolutie gekend. Gaande van de eerste enorme computers, die hele kamers in beslag namen, tot op de tijd van heden, waarin we bijna overal computer en computerchips terugvinden, met een veelvoud aan rekenkracht. Deze evolutie van computerchips wordt sinds 1965 bijgehouden in wat we de wet van Moore noemen. Moore stelde dat het aantal transistors op een computerchip elke 2 jaar zou verdubbelen. Het aantal transistoren op een chip is recht evenredig met de snelheid van die chip. Meer transistoren is meer snelheid. Maar wat te doen met al die rekenkracht? Is het echt nodig dat we zulke snelle computers hebben?

Computers worden vooral gebruikt om problemen op te lossen met behulp van algoritmes. Een voorbeeld van zo een algoritme is dat van de optelling. Hiermee kunnen we twee getallen optellen. We verwachten dat we een algoritme altijd een input meegeven, waarop het dan operaties uitvoert om tot de gezochte oplossing te komen. In ons voorbeeld van de optelling geven we twee getallen mee als input en verwachten we als output het resultaat van de som van deze twee getallen. Wanneer we zo een algoritme hebben om een probleem op te lossen, willen we graag weten of dit algoritme ook de snelste en efficiëntste manier is om tot deze oplossing te komen. Er zijn twee belangrijke factoren waar we rekening mee houden als we een algoritme beoordelen. Het eerste is de tijd die nodig is om tot de oplossing te komen. Als tweede willen we graag zo weinig mogelijk geheugenruimte gebruiken om een oplossing te vinden. Deze tijd en geheugenruimte die nodig zijn, drukken we niet uit in seconden en megabytes, omdat dit dan voor elke PC een andere waarde zou zijn. In plaats daarvan opteren we ervoor om dit uit te drukken in een functie van de inputlengte. Deze functie gebruiken we dan als een bovengrens voor de tijd en geheugenruimte die het algoritme nodig heeft. Aan de hand van deze bovengrens kunnen we dan zien hoe efficiënt een algoritme is en hoe de uitvoeringstijd en opslagruimte zal toenemen, als we de lengte van de input vergroten. We kennen tegenwoordig voor veel problemen een efficiënte oplossing. Maar er zijn echter ook veel problemen, waarvoor we er geen kennen. Deze kunnen door een computer niet snel opgelost worden in een aanvaardbare hoeveelheid geheugenruimte. De tijd die sommige van deze inefficiënte algoritmes nodig hebben, kan heel snel oplopen als de inputlengte groter wordt. Nog meer zelfs dan de lengte

van een mensenleven. Net om deze problemen op te lossen hebben we nood aan steeds snellere computers. Maar zoals reeds aangehaald, bereiken we een fysieke grens die ons tegenhoudt. We moeten zoveel transistors op een chip plaatsen, zodat deze zo klein worden en zo dicht tegen elkaar komen te staan, dat er op atomair en subatomair niveau storingen beginnen op te treden. Deze storingen worden veroorzaakt door de Quantumeffecten die op dat niveau optreden en zorgen ervoor dat deze chips fouten beginnen te maken en dus niet bruikbaar zijn.

Wat als dit probleem van Quantumeffecten ook de oplossing kan zijn in onze zoektocht naar snelheid? We kunnen deze Quantumeffecten gebruiken om een computer te maken met meer rekenkracht dan de PC's dan waarover we nu beschikken. Hoe werkt zo een Quantumcomputer dan? Bij een klassieke computer maken we gebruik van bits om informatie voor te stellen. Deze bits kunnen ofwel 0 ofwel 1 zijn. Waar we 0 vaak beschrijven als uit en 1 als aan. Met behulp van deze bits kunnen we dan informatie digitaal weergeven en verwerken met behulp van een computer. Wanneer we nu een Quantumcomputer gebruiken, maken we niet langer gebruik van bits, maar van Quantumbits, ook wel qubits genoemd. Zo een qubit kan net als de klassieke bits 0 of 1 zijn. Maar een qubit heeft nog een bijzondere eigenschap. Een qubit kan zich namelijk ook in een speciale toestand bevinden, waarin hij zowel 0 als 1 is op hetzelfde moment. Deze toestand noemen we de superpositie van een qubit. Er gebeurt echter iets eigenaardigs wanneer we deze superpositie observeren. De qubit "kiest" dan om ofwel 1 ofwel 0 te zijn, maar zal niet meer bestaan in de superpositie. Deze eigenschap van superpositie gebruiken we in de Quantumcomputer om meer rekenkracht te krijgen. We kunnen deze superpositie van een qubit ook bekijken als een kansberekening. Een willekeurige qubit wordt dan voorgesteld als een combinatie van een kans om 0 te zijn plus een kans om 1 te zijn. Deze kansen verschillen echter van hoe wij in het dagelijkse leven omgaan met kansen. Wij verwachten dat een kans positief is en dat alle kansen samengeteld gelijk zijn aan 1 of 100%. Als voorbeeld nemen we een dobbelsteen. De kans dat we een aantal ogen gooien met de dobbelsteen is 1/6 voor elke zijde van de dobbelsteen en het totaal is dan 1. Wanneer we echter kansen voorstellen met behulp van Quantummechanica, kunnen deze kansen ook negatief of imaginair zijn. Dit is voor ons zeer vreemd om voor te stellen. Maar in deze kansen zit net de kracht van Quantum computing. Als we deze kansen kunnen beïnvloeden, kunnen we ze gebruiken om mee te rekenen. Vaak wordt Quantum computing voorgesteld als volgt: we laten de qubits in superpositie alle mogelijke oplossingen aannemen op hetzelfde moment. Een Quantumcomputer zou dit in principe kunnen. We kunnen de qubits zo beïnvloeden dat ze alle mogelijke waardes voor een functie tegelijk aannemen. Maar hoe selecteren we hier de juiste oplossing uit? Dit is de moeilijkheid van Quantum computing. We hebben al aangehaald dat wanneer we een

qubit observeren, hij niet meer bestaat in een superpositie maar zich als een gewone bit gedraagt. Ook de kansen waaruit de superpositie bestaat, gaan op dat moment verloren. Wanneer we dus de qubits gaan meten, krijgen we niet alle mogelijke oplossingen, maar slechts een willekeurig antwoord van alle mogelijkheden die er zijn. We moeten dus een manier vinden om met deze kansen te rekenen zonder dat we de superpositie laten verdwijnen.
Er zijn tegenwoordig al enkele algoritmes bekend waarvan we weten dat ze problemen kunnen oplossen, waarvoor we tot nu toe nog geen efficiënte oplossing kennen op een klassieke computer. Het bekendste voorbeeld van zo een Quantumalgoritme is het factoring algoritme van Shor. Dit algoritme heeft ervoor gezorgd dat er plots veel mensen geïnteresseerd waren in Quantum Computing, omdat het factoring probleem door ons beschouwd wordt als een probleem dat zo enorm veel tijd vraagt om op te lossen voor een redelijke inputlengte, dat we het als onoplosbaar beschouwen. Daarom zijn er verschillende systemen op gebaseerd die zorgen voor encryptie en beveiliging op het internet. Wanneer we in de mogelijkheid zouden verkeren om dit algoritme te gebruiken, zouden we deze systemen makkelijk kunnen kraken.

Als Quantum computing dan toch kan zorgen voor zoveel snelheidswinst bij het oplossen van problemen, waarom beschikken we dan nog steeds niet over zo een Quantumcomputer? Dit komt vooral omdat het zeer moeilijk is om een qubit te maken en deze daarna te laten bestaan, zonder zijn superpositie te verstoren. Dit wordt nog moeilijker, wanneer we er ook nog eens berekeningen op willen uitvoeren. Onderzoekers uit verschillende labo's over de hele wereld proberen daarom nog steeds stabiele Quantumcomputers te creëren met maar een handvol qubits. Een Canadees bedrijf D-Wave beweert reeds over een Quantumcomputer te beschikken van maar liefst 512 qubits. Hoewel D-Wave al twee van deze machines verkocht heeft voor enkele miljoenen aan Lockheed-Martin en aan NASA en Google, staat de wetenschappelijke wereld hier zeer kritisch tegenover. Deze kritiek bleek later trouwens terecht, nadat de twee machines getest werden door verschillende experts. Toch is het zeker interessant om te blijven zoeken naar een manier om Quantumcomputers te maken en naar algoritmes die we op zo een Quantumcomputer kunnen gebruiken om problemen op te lossen. Problemen, die op dit moment voor ons niet oplosbaar zijn in aanvaardbare tijd en geheugenruimte.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Quantum Computing

Richting: **master in de informatica-multimedia**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Anthonissen, Jan

Datum: **25/06/2014**