Tampereen yliopisto

Valtteri Ala-Salmi

GROVERIN ALGORITMIN SOVELTUVUUS HAKUALGORITMINA

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Elokuu 2022

TIIVISTELMÄ

Valtteri Ala-Salmi: Groverin algoritmin soveltuvuus hakualgoritmina
Kandidaatintutkielma
Tampereen yliopisto
Tietotekniikka
Elokuu 2022

Groverin hakualgoritmi on ollut viime vuosina tarkastelun kohteena uutena vaihtoehtona etsiä alkiota tietokannasta. Sen kvanttimekaniikkaan pohjautuva toteutus on todettu ajalliselta kompleksisuudeltaan tehokkaammaksi löytää alkio järjestämättömästä tietokannasta kuin minkään muun hakualgoritmin.

Tämän tutkielman tarkoituksena on selvittää Groverin algoritmin soveltuvuutta hakualgoritmina teorian sekä käytännön näkökulmasta. Teorian näkökulmasta tutkielma selvittää, millaisilla tietokannan ominaisuuksilla Groverin algoritmi on asymptoottisesti tehokkaampi verrattuna klassisiin hakualgoritmeihin. Klassisina hakualgoritmeina tutkielma käsittelee peräkkäishakua, puolitushakualgoritmia sekä hajautustaulua. Käytännön näkökulmana tutkielma tekee kirjallisuuskatsauksen tutkimuksiin, jossa Groverin hakualgoritmi on toteutettu kvanttitietokoneella.

Tutkielmassa selvisi, että Groverin algoritmin asymptoottinen tehokkuus on riittävän dynaamisessa tietokannassa melkein aina klassisia hakualgoritmeja parempi haettaessa yksittäistä alkiota tietokannasta. Tietokannan suuruuden kasvaessa, Groverin algoritmin asymptoottinen tehokkuus on yhä useammin parempi verrattuna klassisiin hakualgoritmeihin. Ainoastaan staattisissa tietokannoissa klassiset hakualgoritmit ovat huomattavasti Groverin algoritmia tehokkaampia.

Kirjallisuuskatsauksen tulokset antoivat kuitenkin päinvastaisen kuvan algoritmin suorituskyvystä. Tutkimuksissa Groverin algoritmin suorituskyky oli teoreettista alkion löytämisen todennäköisyyttä huomattavasti pienempi. Tämä johti siihen, että jo hyvin pienillä kvanttitietokannoilla, algoritmi ei kyennyt löytämään alkiota riittävän useasti. Ero teorian ja käytännön välillä, perustuen tutkimuksien hypoteeseihin, uskotaan liittyvän kvanttitietokoneiden virhealttiuteen muun muassa porteissa tapahtuvien virheiden sekä dekoherenssin vuoksi.

Tutkimuksen lopputuloksena on, että nykyinen kvanttitietokoneiden teknologia ei ole riittävällä tasolla Groverin algoritmin toteuttamiseksi riittävän suurella onnistumisen todennäköisyydellä. Kuitenkin teoreettisesti Groverin algoritmilla on tietyillä tietokannan ominaisuuksilla huomattavasti parempi asymptoottinen tehokkuus kuin muilla hakualgoritmeilla. Groverin hakualgoritmin todellisen soveltuvuuden tutkielma määrittelee muodostuvan kvanttitietokoneiden tulevaisuuden kehityksen tuomista mahdollisista ratkaisuista.

Avainsanat: Groverin algoritmi, kvanttihakualgoritmi, hakualgoritmi, asymptoottinen tehokkuus

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

IBM	International Business Machines
$\vee$	Looginen tai
$\wedge$	Looginen ja
$\otimes$	Tensoritulo
α	Todennäköisyysamplitudi tilalle $\binom{1}{0}$
β	Todennäköisyysamplitudi tilalle $\binom{0}{1}$
Φ	Kvanttimekaaninen tila
Θ	Theetanotaatio
a	Kvanttimekaanisen tilan keskimääräinen amplitudi
at_b	Puolitushaun asymptoottinen kokonaisaika
at_g	Groverin algoritmin asymptoottinen kokonaisaika
b	Kubittien määrä
D	Tietokanta
h	Hajautusfunktio
i	Tietorakenteen indeksi
k	Yksittäinen avain
K	Avainten lukumäärä tietokannassa
n	Yksittäinen alkio
N	Alkioiden määrä tietokannassa
M	Hakukriteerit täyttävä alkio tietokannassa
O	Ordo-notaatio
OP_1	Hakuoperaation todennäköisyys tietokantaan
OP_2	Järjestysoperaation todennäköisyys tietokantaan
t	Groverin operaation toistomäärä

1. JOHDANTO

Kvanttialgoritmit ovat nousseet suurempaan tarkasteluun viimeisien vuosien aikana johtuen kvanttitietokoneiden teknologisesta kehityksestä. Tätä ennen, vaikka kvanttialgoritmeja oli olemassa, niitä ei voitu toteuttaa käytännössä, koska kvanttitietokoneet olivat lähinnä teorian tasolla.

Yksi paljon kiinnostusta herättäneistä kvanttialgoritmeista on Groverin algoritmi. Se on kvanttimekaniikkaa hyödyntävä hakualgoritmi, joka löytää epäjärjestyksessä olevasta tietokannasta haettavan alkion nopeammin kuin yksikään muu hakualgoritmi (Grover, 1996). Tämän lisäksi algoritmia on tutkittu myös useaan muuhun tarkoitukseen tietokannassa tehtäviin erilaisiin hakuihin.

Tämän tutkielman tarkoituksena on selvittää Groverin algoritmin soveltuvuutta hakualgoritmina. Tähän kysymykseen vastatakseen tutkielma tekee vertailun asymptoottisessa tehokkuudessa Groverin algoritmille ja sitä vastaaville klassisille hakualgoritmeille, kun etsitään yksittäistä alkiota tietokannasta. Tämän lisäksi tutkielma tekee kirjallisuuskatsauksen Groverin algoritmin suoriutumisessa todellisessa kvanttitietokoneessa eri suuruisilla tietokannoilla.

Luvussa kaksi käsitellään kvanttialgoritmien taustaa tutustumalla, mistä komponenteista kvanttialgoritmi koostuu. Luvussa kolme esitellään tutkielmassa vertailtavat hakualgoritmit. Luvussa neljä käydään läpi asymptoottisen tehokkuuden arviointia hakualgoritmien välillä, kun haetaan yhtä alkiota tietokannasta. Luvussa viisi tehdään kirjallisuuskatsaus Groverin algoritmin suoriutumisesta kvanttitietokoneissa. Luvussa kuusi käsitellään tulokset ja tehdään yhteenveto.

2. KVANTTIALGORITMIEN TAUSTAA

Tässä luvussa käsitellään kvanttialgoritmien toiminnan taustaa. Luku käsittelee, miten kvanttimekaniikkaa käyttävä kvanttialgoritmi eroaa rakenteeltaan tavallisesta algoritmista. Lisäksi luku esittelee, miten kvanttialgoritmi muodostetaan sekä millä tavoin sen toimintaa kuvataan.

2.1 Kubitti ja superpositio

Klassisissa tietokoneissa tietoa käsitellään bitteinä, jonka tila abstraktilla tasolla on joko tosi tai epätosi eli lukuarvona 0 ∨ 1. Jotta bitille voidaan antaa tila käytännössä, on sen logiikka teknisesti jotenkin toteutettava tietokoneessa. Klassisissa tietokoneissa bitin toteutuksia ovat esimerkiksi niiden tiedon ylläpitäminen kondensaattoreissa tai transistoreissa hyödyntäen sähkövarauksia, joita muuttamalla bitin tilaa voidaan muuttaa.

Kvanttitietokoneissa bitin toteutus poikkeaa klassisesta kahden tilan systeemistä. Bitin toteutus kvanttitietokoneissa hyödyntää nimensä mukaisesti kvanttimekaniikkaa ja toteutusta kutsutaan bitin sijasta kubitiksi tai kvanttibitiksi. Tarkemmin kubitti on määritelty kaksiulotteiseksi kvanttimekaaniseksi tilaksi (Abhijith ym., 2022, s. 3). Sen sijaan, että kubitti olisi puhtaasti joko tilaltaan 0 ∨ 1, sen kvanttimekaaninen tila muodostuu seuraavasta yhtälöstä:

$$|\Phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle. \qquad (1)$$

Yhtälö 1 koostuu kahdesta kompleksisesta todennäköisyysamplitudista α ja β, jotka yhdessä toteuttavat seuraavan vaatimuksen: $|\alpha^2| + |\beta^2| = 1$. Yhtälöstä $|0\rangle$ ja $|1\rangle$ taasen ovat ket-notaatioita, jotka kuvaavat kubitin kahta perustilaa eli arvoa *0* ∨ *1* kaksiulotteisessa vektorin tilassa $|0\rangle$ ollessa $\binom{1}{0}$ ja $|1\rangle$ ollessa $\binom{0}{1}$. Yhdessä todennäköisyysamplitudit ja vektorit muodostavat niin kutsutun Hilbertin avaruuden, jolla kuvataan mitä tahansa kvanttimekaanista tilaa kompleksisina vektoreina $|\Phi\rangle$. (Abhijith ym., 2022, s. 3)

Yhtälön 1 todennäköisyysamplitudit määrittävät todennäköisyyden sille, kumpaan arvoon kubitti luettaessa päätyy. Amplitudi $|\alpha^2|$ kertoo todennäköisyyden, että kubitin arvo on 0 ja $|\beta^2|$ taasen todennäköisyyden arvolle 1. Tämä kuitenkin koskee kubittia ainoastaan silloin, kun sen tila luetaan. Kun kubitin tilaa ei tarkastella ulkopuolelta, se voi olla tilaltaan *0* ∨ *1* tai superpositiossa (Kommadi, 2021 osa II, luku 3). Superpositiossa oleva kubitti on tilaltaan samanaikaisesti sekä tosi että epätosi eli $0 \wedge 1$.

Toisin kuin klassinen bitti, kubitin tila muuttuu sitä luettaessa (Abhijith ym., 2022, s. 3). Tämän aiheuttaa se, että luettaessa kubitti voi antaa tilakseen ainoastaan toden tai epätoden. Tällöin superpositiossa oleva tila romahtaa ulkopuolisen tarkastelun seurauksena annettujen todennäköisyyksien $|\alpha^2|$ ja $|\beta^2|$ mukaisesti toiseen tiloista. Tämän seurauksena yhtälön 1 mukainen tila menetetään lukemisen jälkeen jättäen tulokseksi 0 ∨ 1 kuten klassisessa bitissä.

2.2 Usean kubitin järjestelmä ja kvanttiportit

Tarkastellaan seuraavaksi järjestelmää, joka koostuukin yhden kubitin sijasta useammasta kubitista. Kubittien muodostamaa järjestelmää voidaan kuvata tensoritulona ⊗, joka vastaa Kroneckerin tulon ottamista kubitin tilojen vektorista (Abhijith ym., 2022, s. 3). Tällöin kubitit muodostavat järjestelmän tilaksi seuraavan tensoritulon:

$$|\Phi_{1,2\dots b}\rangle = |\Phi_1\rangle \otimes |\Phi_2\rangle \otimes \dots \otimes |\Phi_b\rangle$$

$$= \alpha_1\alpha_{\mathbf{2}} \dots \alpha_b|00 \dots 0\rangle + \alpha_1\alpha_{\mathbf{2}} \dots \beta_b|00 \dots 1\rangle + \dots + \beta_1\beta_{\mathbf{2}} \dots \beta_b|11 \dots 1\rangle, \qquad (2)$$

jossa *b* on kubittien määrä järjestelmässä. Tensoritulosta 2 voidaan nähdä, että muodostuvia ket-notaatioita muodostuu 2^b kappaletta, jotka kaikki ovat mahdollisia järjestelmän tiloja, riippuen todennäköisyysamplitudien arvoista.

Jotta kubiteista tehdystä järjestelmästä olisi käytännön hyötyä, on sen tilaa pystyttävä muuttamaan. Tätä varten kubiteille on tehtävä unitaarisia muunnoksia tilan muuttamiseksi. Unitaarista muunnosta kuvataan kompleksisella matriisilla, jonka tulo matriisin transpoosin kanssa muodostaa identiteettimatriisin. (Abhijith ym., 2022, s. 6)

Eri unitaarisia muunnoksia kubiteille kuvataan kubiteista koostuvissa järjestelmissä kvanttiportteina. Kvanttiportit ovat 2*b* * 2*b* unitaarisia matriiseja, jotka sijaitsevat 2*b* suuruisessa Hilbertin avaruudessa (Wang, 2012, s. 379). Toisin kuin klassiset logiikkaportit, kvanttiporttien on toimittava myös käänteisesti, jotta ne täyttävät unitaarisen matriisin ehdot (Abhijith ym., 2022, s. 6). Seuraavaksi esitellään tutkielman kannalta olennaiset kvanttiportit.

Hadamart-portti *H* on yhden kubitin portti, joka asettaa kubitin tilan seuraavalla muunnoksella: $|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$, $|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ (Wang, 2012, s. 379). Kyseisestä muunnoksesta nähdään, että kubitin tilat muuttuvat täydelliseen superpositioon. Unitaarisena matriisina Hadamart-porttia merkitään $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Tutkielmassa käsitellään myös muita yksinkertaisia yhden kubitin portteja, jotka muuttavat kubitin tilaa. Näitä kutsutaan Pauli-porteiksi, jotka jaetaan X-, Y- ja Z-porteiksi. X-

portti muuttaa kubitin todennäköisyysamplitudit päinvastaisiksi vastaten klassista NOT-porttia ja on unitaarisena matriisina $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Z-portti muuttaa kubitin vaihetta piin verran $|\Phi\rangle \rightarrow -|\Phi\rangle$ ja on unitaarisena matriisina täten $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. Y-portti muuttaa sekä todennäköisyysamplitudia että vaihetta päinvastaisiksi sekä imaginäärisiksi seuraavalla muunnoksella: $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$. (De ym., 2022, s. 3)

Control-NOT portti CNOT on kahden kubitin portti, jonka sisääntulosta toinen kubitti on kohdekubitti ja toista käytetään kontrollikubittina. Kontrollikubitin tilan ollessa $|1\rangle$ kohdekubitin tila vaihtuu X-portin tavoin $|0\rangle \rightarrow |1\rangle$ tai $|1\rangle \rightarrow |0\rangle$ (Wang, 2012, s. 379). On olemassa myös kolmen kubitin portti Toffoli, jossa kontrollikubitteja on yhden sijasta kaksi, joista molempien on oltava tosia, jotta kohdekubitin muunnos toteutuu. Control-portteja voidaan luoda myös muille yhden kubitin porteille kuten Z-portille.

2.3 Lomittuminen

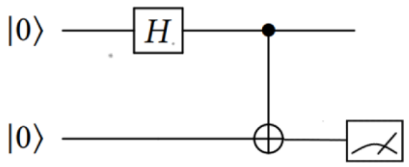
Kubittien muodostama järjestelmä voidaan saattaa tilaan, jossa mittaamalla minkä tahansa kubitin tila järjestelmässä, voidaan tietää järjestelmän muiden kubittien tila. Tilaa kutsutaan lomittumiseksi, joka on kvanttimekaaninen ilmiö. Lomittunutta järjestelmää ei voida kuvata tensoritulona (Abhijith ym., 2022, s. 5). Lomittuminen on superposition ohella toinen tekijä, jotka tekevät kvanttilaskennan niin tehokkaaksi (Balynsky ym., 2021, s. 1).

Lomittuminen saadaan aikaan esimerkiksi seuraavasti: Tehdään CNOT-portilla operaatio, jossa kohdekubitti b_1 on tilaltaan $|0\rangle$ ja kontrollikubitti b_2 on superpositiossa $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ Hadamart-portin muunnoksesta. Jos operaation jälkeen kohdekubitin b_1 tilaksi mitataan $|1\rangle$, on kontrollikubitti b_2 mitattaessa myös $|1\rangle$, koska kohdekubitti vaihtaa tilaansa CNOT-portissa ainoastaan, jos kontrollikubitti on $|1\rangle$. Toisaalta, jos kohdekubitin tilaksi mitataan $|0\rangle$, on kontrollikubitin tila $|0\rangle$. Tällöin kubittien b_1 ja b_2 tila ennen mittausta on $|\Phi_{1,2}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Havaitaan kyseessä olevan lomittunut tila, missä tietämällä yhden kubitin tila tiedetään myös toisen tila.

2.4 Kvanttipiirit

Kvanttipiirit ovat kvanttiporteista koostuvia rakennusosia, joista voidaan koostaa kvanttitietokone (Kommadi, 2021 osa II, luku 3). Myös kvanttialgoritmien toimintaa kuvataan usein kvanttipiirinä (Abhijith ym., 2022, s. 9).

Kvanttipiirissä kubitteja kuvataan vaakasuorassa olevalla linjalla, jotka etenevät vasemmalta oikealle, kulkien niille tehtävien eri operaatioiden eli porttien läpi. Kuvassa 1 on esitetty kvanttipiirinä edellisessä luvussa käsitelty lomittumisen esimerkki. Hadamart-porttia on merkitty neliön sisällä olevalla H:lla, symboli • on CNOT-portin kontrollibitti ja ⊕ kohdebitti. Viimeinen operaatio kohdebitille on mittausportti.



Kuva 1. *Lomittumisen esimerkki kvanttipiirinä. Muunnettu (Abhijith ym., 2022, s. 9).*

3. HAKUALGORITMIT

Tässä luvussa esitellään tutkielmassa käsiteltävät hakualgoritmit. Lisäksi selvitetään millä tavalla esiteltyjä hakualgoritmeja tullaan vertailemaan keskenään.

3.1 Hakualgoritmin tehokkuus

Hakualgoritmi on algoritmi, joka noutaa halutun tiedon tai reitin hakualueelta. Hakualue voi olla osa tietokantaa tai se voi kattaa koko tietokannan. Tieto tai reitti koostuu halutuista alkioista tietokannassa, ja niiden muodostaman kokonaisuuden palauttaminen haussa on hakualgoritmin tavoite. (Leider ym., 2020, s. 448)

Hakualgoritmin tehokkuuden määrittää sen suorituskyky sekä ajassa että tilassa eli muistissa (Leider ym., 2020, s. 448). Nopea ajallinen suoritusaika on tärkeää, sillä hakualgoritmin oletetaan pystyvän antamaan vastaus hakuun riittävän nopeasti. Liian suuri muistin tarve taas estää haun, koska muistia ei ole algoritmia käyttävässä laitteessa tarpeeksi.

Jotta hakualgoritmeja voi vertailla keskenään, on otettava huomioon jokaisen algoritmin suoritusaika sekä muistin tarve. Ongelmaksi kuitenkin nousee, että algoritmien suorituskyky riippuu hyvin useasta tekijästä kuten laitteistosta, toteutuksesta sekä tietokannan ominaisuuksista. Ratkaisuna algoritmeja vertaillaan niiden kompleksisuudella.

Algoritmin kompleksisuutta ilmaistaan asymptoottisina kasvunopeuksina, jotka asettavat algoritmin suoritusajan sekä muistin tarpeen kasvunopeuden tietokannassa olevien alkioiden lukumäärään nähden. Tutkielmassa kompleksisuutta merkitään pääasiassa asymptoottisella Ordo-notaatiolla O, jolla osoitetaan hakualgoritmin suorituksen kompleksisuuden ylärajaa eli suoritusta huonoimmassa tapauksessa (Leider ym., 2020, s. 448). Lisäksi keskimääräistä suoritusta merkitään tutkielmassa theetanotaatiolla Θ. Koska algoritmien välisen tehokkuuden vertailu tapahtuu käyttämällä asymptoottisia kasvunopeuksia, käsitettä on tutkielmassa kuvattu termillä asymptoottinen tehokkuus.

3.2 Klassiset hakualgoritmit

Perinteiselle tietokoneelle on olemassa kolme hakualgoritmia ja niiden variaatioita, joita käytetään alkioiden etsimiseen tietokannasta. Klassisia hakualgoritmeja ovat peräkkäishaku, puolitushaku sekä hajautustaulu. Kyseiset hakualgoritmit eroavat huomattavasti toisistaan siinä, minkälaisessa tilanteessa hakualgoritmi on kaikkein tehokkain, ja täten hakualgoritmin valinta perustuukin algoritmien eroavaisuuksiin.

Yksinkertaisin tapa etsiä alkiota tietokannasta on edetä alkio kerrallaan tietokannassa, kunnes päädytään haluttuun alkioon. Tällaista algoritmia kutsutaan lineaariseksi tai peräkkäiseksi hauksi. Jos tietokanta on mielivaltaisessa järjestyksessä, algoritmi löytää halutun alkion tietokannasta keskimäärin, kun puolet alkioista on tutkittu. Tällöin algoritmin suoritusaika ajallisesti on $\Theta(N)$, jossa *N* on alkioiden lukumäärä tietokannassa. Muistin tarve algoritmilla on $O(1)$, koska algoritmi käsittelee kerrallaan ainoastaan yhtä alkiota.

Heuristiikan avulla peräkkäishaun suoritusaikaa on mahdollista optimoida. Tällainen tilanne on esimerkiksi silloin, kun tiedetään, että tietokannassa etsitään yleensä vain tiettyjä alkioita. Jos tietokanta on järjestyksessä siten, että alkioiden järjestys määräytyy niiden etsinnän suosion mukaan, algoritmin suositusaika pienenee sen mukaan, kuinka pieneen osuuteen alkioista etsintöjen kiinnostus keskimäärin kohdistuu. (Hester & Hirschberg, 1985)

Seuraavan hakualgoritmin periaate perustuu tietokannan alkioiden järjestämiseen tietyllä tavalla hakutuloksen nopeuttamiseksi. Sitä kutsutaan binääri- tai puolitushauksi. Puolitushaussa tietokanta on alkioiden tietyn ominaisuuden mukaisessa suuruusjärjestyksessä. Tällöin lähdettäessä hakemaan etsittävää alkiota tietokannassa voidaan edetä seuraavan rekursiivisen ohjeen mukaisesti:

1. Valitaan järjestetyn hakualueen keskimmäinen alkio $n_{N/2}$.
2. Tarkistetaan, onko alkio $n_{N/2}$ etsitty alkio, jos ei jatketaan vaiheeseen kolme.
3. Jos alkion $n_{N/2}$ arvo on pienempi kuin etsittävä alkio, aloitetaan uudelleen vaiheesta 1 niistä alkioista, jotka ovat $n < n_{N/2}$. Jos $n_{N/2}$ arvo on suurempi, hakualue on alkioista, jotka ovat $n > n_{N/2}$.

Havaitaan, että kunkin toiston jälkeen haettava hakualue puolittuu. Puolitushaun suoritusajaksi saadaan tällöin $O\big(log_2(N)\big)$.

Tietokannan alkioiden tilan muuttuessa esimerkiksi uuden alkion lisäyksen myötä tarvitaan järjestämisalgoritmia, jotta tietokanta saadaan uudestaan puolitusalgoritmin vaatimaan järjestykseen. Puolitushakualgoritmin kokonaissuoritusaika muodostuukin sekä itse algoritmin suoritusajasta että järjestämisalgoritmin suoritusajasta ottaen huomioon hakuoperaation sekä järjestämisoperaation välisen todennäköisyyden suhteen, jota käsitellään luvussa neljä. Nopeimpien järjestämisalgoritmien kesto on $O(N \log(N))$, eikä kompleksisesti nopeampaa järjestämisalgoritmia ole olemassa (Cormen, 2009, s. 193–194). Muistillisesti peräkkäishaun tavoin puolitushakualgoritmi tarvitsee ainoastaan käsiteltävän alkion kerrallaan eli $O(1)$.

Viimeinen klassinen hakualgoritmi on hajautustaulu. Hajautustaulussa alkion etsintään käytetyn parametrinä toimivan avaimen ja tietorakenteessa olevan alkion väliin on

asetettu hajautusfunktio. Hajautusfunktio muuntaa sille annetun avaimen hajautusarvoksi, joka toimii etsittävän alkion indeksinä tietorakenteessa. Eli käyttäjän hakiessa alkiota n_x avaimella k_x, muuntaa hajautusfunktio h avaimen $h(k_x) = i_x$, jossa i_x on haettavan alkion indeksinä toimiva hajautusarvo.

Koska hajautusfunktio laskee haettavan alkion hajautusfunktion perusteella, algoritmin suoritusaika tulee muodostumaan sekä hajautusfunktion suoritusajasta, että sen antaman tuloksen käsittelystä. Olkoon $h(k_x) = i_x$ niin, että tietokannassa jokaisella mahdollisella alkiolla n_x on oma uniikki avain k_x. Lisäksi hajautusfunktio h antaa jokaiselle mahdolliselle avaimelle, joka tietokanta voi sisältää, oman uniikin indeksin i_x. Tällöin kaikille alkioille, jotka ovat tai voivat olla olemassa tietokannassa, pätee $D[i_x] = n_x,$ jossa *D* on tietorakenne.

Jos edellä mainitut vaatimukset pätevät, muodostuu hajautusfunktion suoritusaika ainoastaan hajautusfunktion kestosta, joten asymptoottinen suoritusaika on $O(1)$. Koska hajautustaulu muodostuu tässä tapauksessa *N* määrästä eri indekseillä olevista alkioista, on luotava *N* suuruinen taulukko (engl. array) tai suora osoitetaulukko (engl. direct address table). Tällöin muistillinen kompleksisuus on $O(N)$. (Cormen, 2009, s. 254–256)

Kuitenkin, jos alkioilla voi olla sama avain tai hajautusfunktio h antaa useammalle kuin yhdelle avaimelle saman indeksin arvon, tilanne muuttuu. Indeksiä i_x tulee vastaamaan $\sum_{n=D[h(k_{x1})]}^{n=D[h(k_{xn})]} n$ alkiota, jossa k_x on ensimmäinen mahdollinen avain ja k_{xn} viimeinen avain, jotka hajautusfunktio osoittaa indeksiin i_x. Sen sijaan, että indeksi osoittaisi suoraan alkioon, on sen osoitettava tietorakenteeseen, missä edellä mainitut n alkiota ovat.

Jokaista indeksiä kohtaan oleva tietorakenne voidaan luoda linkitettynä listana. Muistin kompleksisuudeksi saadaan $O(K)$, jossa K on avaimien lukumäärä hajautustaulussa. Kuitenkin pahimmassa tapauksessa kaikki alkiot voivat päätyä samaan linkitettyyn listaan, jolloin saadaan peräkkäishaun suoritusajan kompleksisuus $\Theta(N)$. Keskimääräisessä tapauksessa suoritusajan kompleksisuus on $\Theta(1 + \alpha)$, missä $\alpha = \frac{N}{K}$. Jos K on riittävän suuri suhteessa *N*:ään, keskimääräinen kompleksisuus on $O(1)$. (Cormen, 2009, s. 256–260) Käyttämällä linkitetyn listan sijaan hajautustauluja niin kutsutussa täydellisessä hajautuksessa, huonoinkin mahdollinen suoritusajan kompleksisuus on $O(1)$. Kuitenkin muistin kompleksisuus kasvaa jälleen arvoon $O(N)$. (Cormen, 2009, s. 277–280)

3.3 Groverin algoritmi

Groverin algoritmi on kvanttihakualgoritmi, jonka on kehittänyt Lov Grover vuonna 1996. Algoritmi löytää alkion epäjärjestyksessä olevasta tietokannasta kompleksisella suoritusajalla $O(\sqrt{N})$ ja $\Theta(\sqrt{N})$. Klassiset hakualgoritmit sen sijaan pystyvät löytämään epäjärjestyksellisestä tietokannasta alkion alimmillaan suoritusaikaan $O(N)$. (Grover, 1996)

Algoritmi pystyy etsimään myös useamman hakukriteerin täyttävän alkion tietokannasta. Siinä tapauksessa algoritmin kompleksinen suoritusaika on $O(\sqrt{N/M})$, jossa *M* on hakukriteerit täyttävät alkiot tietokannassa. Muistin kompleksisuus algoritmilla on $O(\log(N))$ (Kommadi, 2021 osa II, luku 4). Groverin algoritmi on kolmivaiheinen:

1. Alustetaan systeemin kubitit kvanttiporttien muunnoksilla täydelliseen superpositioon. Tämän vaiheen asymptoottinen suoritusaika on $O(log_2(N))$.
2. Toistetaan seuraavat alavaiheet $O(\sqrt{N})$ kertaa:
    a. Oraakkeli
    b. Diffuusio

    Tätä vaihetta kutsutaan kokonaisuudessaan Grooverin operaatioksi- tai iteraatioksi. Sen asymptoottinen suoritusaika on $O(\sqrt{N})$.
3. Mitataan systeemin kubitit mittausporteilla, jossa oikean alkion löytämisen todennäköisyys on suurempi kuin 50 %.

Oraakkelin tarkoitus on erottaa etsittävä alkio muista. Erotuksen oraakkeli tekee siirtämällä etsittävän alkion vaihetta radiaanin verran (Grover, 1996, s. 214). Tämä aiheuttaa seuraavan muutoksen: $|\Phi\rangle \rightarrow -|\Phi\rangle$, jossa etsittävänä alkiona toimivan kvanttimekaanisen tilan y vaihe vaihtuu päinvastaiseksi ja muita alkioita pienemmäksi. Diffuusion tehtävänä on vähentää kaikkien alkioiden todennäköisyysamplitudit $(2\langle a\rangle - a_x)|\Phi\rangle$, jossa a on alkioiden keskimääräinen amplitudi ja a_x yksittäisen alkion amplitudi (Abhijith ym., 2022, s. 16–18). Koska oraakkeli muutti etsittävän alkion vaiheen päinvastaiseksi, on se diffuusion jälkeen $(2\langle a\rangle + a_x)|\Phi\rangle$ muiden alkioiden amplitudin laskiessa.

Groverin algoritmi antaa kaikista todennäköisemmin oikean hakutuloksen, kun Groverin operaatiota toistetaan $\frac{\pi}{4}\sqrt{N}$ kertaa (Zalka, 1999). Todennäköisyydeksi saadaan tällä toistomäärällä $1 - O(N^{-1/2})$ (Vrana ym., 2014 s.7). Todennäköisyys mille tahansa toistomäärälle *N* suuruisessa tietokannassa saadaan seuraavasta yhtälöstä:

$$t\left(\frac{\frac{N-2t}{N} + 2\frac{N-t}{N}}{\sqrt{N}}\right), \tag{3}$$

jossa t on Groverin operaation toistomäärä (Strömberg & Karlson, 2018, s. 10).

4. YKSITTÄISEN ALKION HAKEMINEN

Edellä olevassa luvussa esitettiin kolme klassista hakualgoritmia, joiden avulla voidaan löytää haluttu alkio *N* suuruisesta tietokannasta. Niiden lisäksi esiteltiin Groverin algoritmi, joka tarjoaa uuden tavan löytää alkio epäjärjestyksessä olevasta tietokannasta. Tässä luvussa selvitetään, missä tilanteissa Groverin algoritmin käyttö on asymptoottisesti tehokkaampi vaihtoehto yksittäisen alkion etsintään klassisiin algoritmeihin verrattuna.

4.1 Vertaus peräkkäishakuun

Groverin algoritmia verrattaessa peräkkäishakuun huomataan, että ajallisesti Groverin algoritmin asymptoottinen suoritusaika on neliöllisesti pienempi, koska Groverin algoritmin suoritusajan huonoin kompleksisuus on peräkkäishaun keskimääräistä kompleksisuutta pienempi $O(\sqrt{N}) < \Theta(N)$. Toisaalta muistillisesti peräkkäishaku pysyy vakiona, kun taas Groverin algoritmin muistin tarve kasvaa logaritmisesti.

Kuitenkin, jos tietokannan alkioihin kohdistuva kiinnostus on vahvasti priorisoitunut vain pieneen osaan alkioita, tilanne muuttuu. Kuten edellisessä luvussa todettiin, asettamalla tietokannan alkiot hakumäärän perusteella järjestykseen, voidaan peräkkäishaun suoritusaikaa optimoida. Jos on tietokanta, missä suoritettavien hakuoperaatioiden kiinnostus keskimäärin kohdistuu $\sqrt{N}$ osaan alkioista, käyttämällä edellä mainittua optimointia, peräkkäishaun suoritusaika keskimäärin on $\Theta(\sqrt{N})$. Tämä johtuu siitä, että koska tietokannan alkiot ovat hakumäärän perusteella järjestyksessä, hakualgoritmi pysähtyy keskimäärin ensimmäisiin $\sqrt{N}$ alkioon, joihin hakuoperaatiot keskimäärin kohdistuvat.

Tällöin keskimäärin peräkkäishaku olisi yhtä nopea kuin Groverin algoritmin kompleksinen suoritusaika. On kuitenkin huomattava, että suoritusajan yläraja pysyy edelleen arvossa $O(N)$, sillä hakuoperaatio voi kohdistua $\sqrt{N}$ alkion ulkopuolelle. Lisäksi ratkaisu vaatisi aina tarkastuksen, onko alkion hakuoperaatioiden määrä ylittänyt edellä olevia alkioita. Olettamalla kuitenkin alkioihin kohdistuvien hakuoperaatioiden olevan tarpeeksi erotettuna toisistaan olisi tarkastuksen suoritusaika $\Theta(1)$.

Koska Groverin algoritmin kompleksisesti nopeammin kasvava ominaisuus on suoritusaika, on Groverin algoritmin asymptoottinen tehokkuus $O(\sqrt{N})$. Peräkkäishaussa nopeimmin kasvava ominaisuus on myös suoritusaika ja asymptoottinen tehokkuus on $\Theta(N)$. Täten Groverin algoritmi on ilman peräkkäishaulle tehtävää heuristiikkaa

asymptoottisesti tehokkaampi. Jos edellä mainittu tietokannan ominaisuus hakuoperaatioiden kohdistumisesta tiettyyn osaan alkioista pitää paikkansa, peräkkäishaku voi olla asymptoottisesti yhtä tehokas tai tehokkaampi, jos hakuoperaatiot kohdistuvat <= $\Theta(\sqrt{N})$ alkioon.

4.2 Vertaus puolitushakuun

Kun selvitetään tehokkuutta puolitushakualgoritmin ja Groverin algoritmin välillä, on huomioitava, että puolitushakualgoritmin ajallinen suoritusaika koostuu kahdesta komponentista. Kuten edellisessä luvussa mainittiin, puolitushakualgoritmin suoritusaika koostuu sekä itse algoritmin suoritusajasta sekä algoritmin vaatiman järjestämisalgoritmin kestosta. Tällöin vertailtaessa algoritmeja ajallisessa suorituskyvyssä, on otettava huomioon tietokannassa tapahtuvien hakuoperaatioiden, että järjestämisoperaatioiden välinen todennäköisyyden suhde.

Olkoon OP_1 tietokantaan tehtävän hakuoperaation todennäköisyys ja OP_2 tietokantaan tehtävän järjestämisen todennäköisyys. Jätetään muut mahdolliset operaatiot tarkastelun ulkopuolelle, jolloin mainitut operaatiot ovat ainoat tietokantaan tehtävät operaatiot. Tällöin:

$$OP_1 + OP_2 = 1. \tag{4}$$

Tietokannan asymptoottinen kokonaisaika O-notaatiossa $O(at)$ tulee perustumaan hakuoperaation sekä järjestämisoperaation suoritusajoista sekä niiden välisestä todennäköisyydestä täten:

$$at = OP_1\big(O(hakuoperaatio)\big) + OP_2\big(O(järjestysalgoritmi)\big).$$

Puolitushaun asymptoottiseksi kokonaisajaksi at_b muodostuu operaatioiden perusteella

$$\begin{aligned} at_b &= OP_1\big(O(hakuoperaatio)\big) + OP_2\big(O(järjestysalgoritmi)\big) \\ &= OP_1(O\big(log_2(N)\big) + OP_2(O\big(Nlog_2(N)\big). \end{aligned} \tag{5}$$

Groverin algoritmissa ei ole tarpeellista käyttää tietokannan järjestämisen operaatiota. Tällöin asymptoottinen kokonaisaika at_g on

$$at_g = OP_1\big(O(hakuoperaatio)\big) = OP_1(O\big(\sqrt{N}\big) \tag{6}$$

Selvitetään, millä arvoilla Groverin algoritmin asymptoottinen kokonaisaika on pienempi kuin puolitushakualgoritmissa. Saadaan epäyhtälö, kun kaavojen 5 ja 6 arvot asetetaan epäyhtälöksi:

$$at_g < at_b$$

$$=OP_1(O(\sqrt{N})) < OP_1(O(log_2(N)) + OP_2(O(Nlog_2(N)). \quad (7)$$

Selvitetään epäyhtälöstä 7 OP_2 ratkaisemalla se ensin yhtälössä 4 OP_1.n arvoksi saadaan:

$$OP_1 = 1 - OP_2.$$

Sijoitetaan vastaus epäyhtälöön 7 ja ratkaistaan OP_2 jättäen O-notaatio merkitsemättä selvemmän laskennan vuoksi välivaiheista:

$$(1 - OP_2)\sqrt{N} < (1 - OP_2)log_2(N) + OP_2\mathrm{N}log_2(N)$$

$$= N - OP_2N < log_2(N) - OP_2log_2(N) + OP_2Nlog_2(N)$$

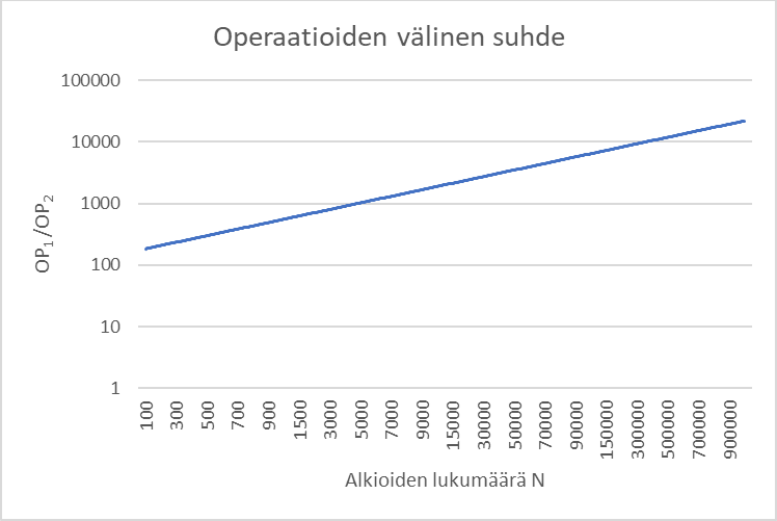
$$= -OP_2\sqrt{N} < log_2(N) - OP_2log_2(N) + OP_2Nlog_2(N) - \sqrt{N}$$

$$= -OP_2\sqrt{N} - OP_2Nlog_2\mathrm{N} + OP_2log_2(N) < log_2(N) - \sqrt{N}$$

$$= OP_2(-N - Nlog_2(N) - log_2(N)) < log_2(N) - \sqrt{N}$$

$$= OP_2 > O(\frac{log_2(N)-\sqrt{N}}{-\sqrt{N}-Nlog_2(N)+log_2(N)}); -\sqrt{N} - Nlog_2(N) + log_2(N) < 0. \quad (8)$$

Saadaan selville, että kun tietokannan järjestämisen todennäköisyyden suuruus täyttää saadun epäyhtälön 8 vaatimuksen, Groverin algoritmin asymptoottinen kokonaisaika on teoreettisesti aina pienempi huonoimmassa tilanteessa kuin puolitushaussa O-notaation mukaisesti. Kuvassa 2 on esitettynä se määrä hakuoperaatioita yhtä järjestämisoperaatiota kohtaan, jossa epäyhtälö 8 on tosi. Viivan alapuolella on ne operaatioiden välisetsuhteet, jossa Groverin algoritmin kokonaissuoritusaika on pienempi kuin puolitushakualgoritmissa ja yläpuolella toisinpäin.



Kuva 2. *Hakuoperaatioiden ja järjestämisalgoritmien välisen todennäköisyyden suhde epäyhtälön 8 mukaisesti.*

Kuvasta 2 nähdään, että tietokannan alkioiden kokonaismäärän kasvaessa, enimmäismäärä hakuoperaatioita jokaista järjestämisoperaatiota kohtaan kasvaa. Esimerkiksi alkioiden määrän ollessa sata, noin alle 200 hakuoperaatiota yhtä järjestämisoperaatiota kohtaan tarkoittaa, että Groverin algoritmin asymptoottinen kokonaisaika on puolitushakua pienempi. Alkioiden määrän lähestyessä miljoonaa, Groverin algoritmi on nopeampi, kun hakuoperaatioita on enintään noin 20000 jokaista järjestämisoperaatiota kohtaan.

Voidaan päätellä, että mitä enemmän tietokannassa olevia alkioita on, sen staattisempi tietokannan on oltava, jotta puolitushaun asymptoottinen kokonaisaika on Groverin algoritmia pienempi. Tämä tarkoittaa, että Groverin algoritmi on alkioiden määrän kasvaessa yhä useammin nopeampi verrattuna puolitushakuun, ellei tietokanta ole luonteeltaan staattinen. Staattisella tietokannalla tarkoitetaan tietokantaa, jonka alkiot eivät muutu eli järjestämistä ei tarvitse tehdä.

Koska puolitushaun kokonaissuoritusajan kompleksisuus on vähintään $log_2(N)$ ja muistin kasvun kompleksisuus on vakio, tulee puolitushaun asymptoottinen tehokkuus muodostumaan kokonaissuoritusajan kompleksisuudesta. Groverin algoritmin asymptoottinen tehokkuus muodostuu myös suoritusajasta kuten peräkkäishaun vertailussa todettiin. Täten esitetty epäyhtälö 8 määrittää myös sen, kumman vertailtavan algoritmin asymptoottinen tehokkuus on tietyillä operaation todennäköisyyksillä ja *N*:n arvolla pienempi.

4.3 Vertaus hajautustauluun

Kuten edellisessä luvussa mainittiin, hajautustaululla on tietokannan ominaisuuksista riippuen erilaisia suoritusajan sekä muistin kasvamisen kompleksisuuksia. Jos tietokannan jokaiselle alkiolle on oma uniikki avaimensa, tai tietokanta on toteutettu käyttämällä täydellistä hajautusta, on suoritusajan kompleksisuus enimmillään $O(1)$ ja muistin kompleksisuus on $O(N)$. Jos tietokanta on toteutettu linkitetyillä listoilla, suoritusajan kompleksisuus on pahimmillaan $O(N)$, keskimäärin $\Theta(1+\alpha)$ ja riittävän suurella avaimien arvolla keskimäärin $O(1)$. Muistin kasvun kompleksisuus on $O(K)$.

Havaitaan, että tietokanta uniikeilla avaimilla tai täydellisellä hajautuksella, on suoritusajan kasvultaan huomattavasti Groverin algoritmia parempi $O(1) < O\left(\sqrt{N}\right)$. Kuitenkin muistin kasvu on näillä hajautustauluilla huomattavasti suurempi $O(N) > O(\log(N))$. Koska hajautustaulun muistin kasvu on $O(N)$, on sen asymptoottinen tehokkuus

Groverin algoritmia neliöllisesti suurempi. Täten voidaan todeta Groverin algoritmin olevan näissä tapauksissa hajautustaulua asymptoottisesti tehokkaampi.

Kun kyseessä on linkitetty lista, selvitetään kompleksisuuksia avaimien K eri arvoilla. Todetaan muistin kasvun kompleksisuuden olevan edellä mainittu $O(K)$ ja suoritusajan olevan $\Theta(1+\alpha) = \Theta\left(1+\frac{N}{K}\right)$, koska linkitetyn listan läpikäymisen kompleksisuus riippuu sen sisältämien alkioiden määrästä suhteessa tietokannan alkioiden määrään. *K*:n ollessa riittävän lähellä *N*:ää, suoritusaika laskee keskimäärin kasvunopeuteen $O(1)$ luvun kolme mukaan mutta vain jos $\Theta\left(1+\frac{N}{K}\right) \approx O(1)$.

Koska Groverin algoritmin asymptoottinen tehokkuus on $O(\sqrt{N})$, on hajautustaulun muistin sekä suoritusajan kompleksisuuden molempien oltava enimmillään $O(\sqrt{N})$. Koska suoritusajan kompleksinen yläraja hajautustaululle on tässäkin tapauksessa $O(N)$, ei hajautustaulu pysty huonoimmassa tapauksessa olemaan yhtä asymptoottisesti tehokas kuin Groverin algoritmi. Tarkastellaan asymptoottista tehokkuutta keskimääräisellä kasvunopeudella. Tuolloin on pädettävä seuraavat epäyhtälöt:

$$\Theta(K) \leq \Theta(\sqrt{N}), \tag{9}$$

$$\Theta\left(1+\frac{N}{K}\right) \leq \Theta(\sqrt{N}). \tag{10}$$

Havaitaan epäyhtälöstä 9, että $K \leq \sqrt{N}$. Epäyhtälöstä 10 havaitaan, että $1+\frac{N}{K} \leq \sqrt{N} = \frac{N}{\sqrt{N}-1} \leq K \approx \sqrt{N} \leq K$. On siis oltava $K \leq \sqrt{N}$ ja $\sqrt{N} \leq K$, tällöin *K* voi olla ainoastaan $\sqrt{N}$. Todetaan, että kun $K = \sqrt{N}$, hajautustaulu on kompleksisesti yhtä tehokas kuin Groverin algoritmi keskimäärin. Kuitenkin Groverin algoritmin muistillinen kasvu on tässäkin tapauksessa pienempi kuin hajautustaululla.

5. GROVERIN ALGORITMIN TOTEUTUKSET KIRJALLISUUDESSA

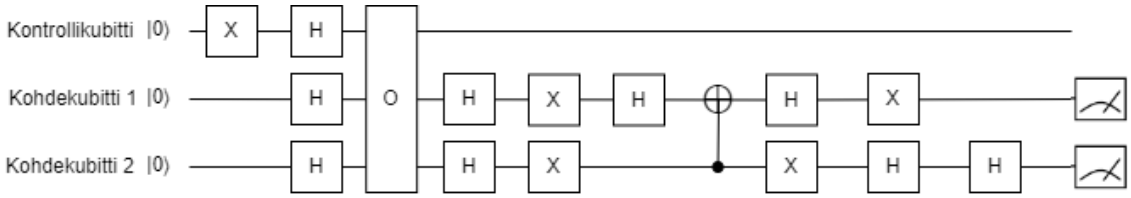
Tässä luvussa tutustutaan tutkimuksiin, jossa Groverin algoritmia on kokeiltu käytännössä kvanttitietokoneilla kubiteista koostuvista tietokannoista. Kubiteista tehdyn tietokannan koko on tutkimuksissa ollut 2–4 kubittia. Seuraavaksi tarkastellaan tutkimusten toteutuksia ja Groverin algoritmin suorituskykyä todellisissa tietokantahauissa eri suuruisten kubittien tietokannoissa.

5.1 Kahden kubitin tietokanta

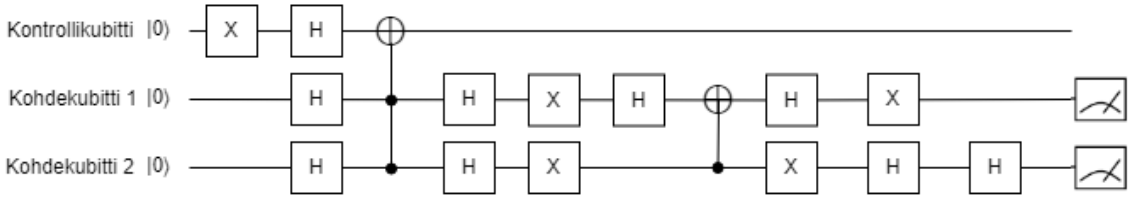
Kirjallisuudessa tehtyjen hakujen perusteella kahden kubitin tietokannasta etsivä Groverin algoritmi on toteutettu kahdella eri menetelmällä. Ne ovat Groverin algoritmin oraakkelin toteuttaminen Boolen logiikalla tai kubittien vaiheita muuttamalla.

Boolen logiikkaa käyttävä oraakkeli hyödyntää AND-, OR- ja NOT-logiikkaa, kuten perinteiset bittien logiikkaportit (kuva 3). Toteutus vaatii tietokannan kubittien lisäksi myös kontrollikubitin, jotta kvanttiporttien käänteisyys säilyy. Ensimmäisenä kontrollikubitille tehdään NOT-muunnos ja tämän jälkeen kaikki kvanttipiirin kubitit asetetaan Hadamart-portilla superpositioon. Tämän jälkeen oraakkelina toimii Boolen logiikkaa käyttävä kvanttiporttien joukko, joka pyrkii heikentämään haettavan alkion todennäköisyysamplitudia verrattuna kaikkien alkioiden amplitudien keskiarvoon. Tämän jälkeen H-, X- ja CNOT-porteilla toteutetaan Groverin Diffuusio ennen mittausportteja, joka kasvattaa haettavan alkion todennäköisyysamplitudia keskiarvon yläpuolelle. (Abhijith ym., 2022, s. 16–17)

Tutkimuksessa, jossa käytettiin Boolen logiikkaa, oraakkeli luotiin kahdella tavalla. Ensimmäisessä käytettiin alkion (1,1) etsimiseen Toffoli-porttia (kuva 4) ja todennäköisyydeksi oikealle hakutulokselle saatiin noin 64,6 %. Toisella toteutuksella kokeiltiin oraakkelin muodostamista CNOT-porteilla ja todennäköisyydeksi saatiin 48 %. Käyttöympäristönä käytettiin IBM:n (International Business Machines) viiden kubitin kvanttitietokonetta ibmqx4-arkkitehtuurilla. Teoreettisesti todennäköisyys löytää oikea alkio tietokannasta pitäisi kahdella kubitilla olla 100 % yhtälön 3 mukaisesti. Tutkimus toteaa alhaisen todennäköisyyden johtuvan kvanttitietokoneessa itsessään tapahtuvista virheistä. (Abhijith ym., 2022, s. 15–18)



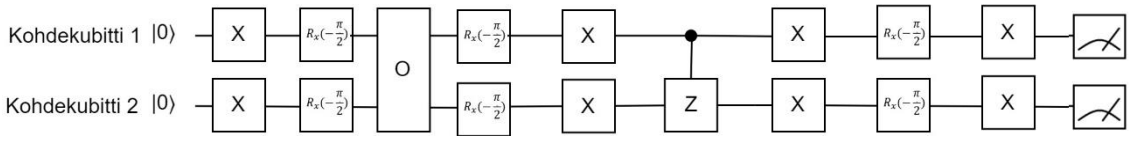
Kuva 3. *Kahden kubitin tietokannan Groverin algoritmi Boolen logiikalla, jossa O on oraakkeli. Perustuu (Abhijith ym., 2022, s. 16).*



Kuva 4. *Kahden kubitin tietokannan Groverin algoritmi Boolen logiikalla, kun haetaan alkiota (1,1). Perustuu (Abhijith ym., 2022, s. 16).*

Oraakkelin toteutus vaihemuunnoksia hyödyntämällä perustuu kubitin vaihetta muuttaviin portteihin (kuva 5). Tässä toteutuksessa ei tarvita ylimääräistä kontrollikubittia. Ennen oraakkelia Hadamart-portin voi korvata R_x-portilla muuntamalla kubitin x-akselin vaihetta $-\frac{\pi}{2}$. Tätä muunnosta ei kuitenkaan ole toteutettu kaikissa vaihemuunnosta käyttävissä toteutuksissa. Oraakkeli muuttaa nimensä perusteella kubittien vaihetta, mutta toteuttaa matemaattisesti ekvivalenssin ratkaisun Boolen logiikkaa käyttävän oraakkelin kanssa (Figgatt ym., 2017, s. 2).

Tutkimuksessa oraakkelina kahden kubitin tietokannan vaihemuunnosta hyödyntävässä toteutuksessa käytettiin C-vaihe-porttia. Tutkimus tehtiin kahden identiteettisen suprajohtavan kubitin kvanttipiirillä. Groverin algoritmin onnistumisen todennäköisyydeksi saatiin kaikilla hakutukoksilla yli 90 % onnistumisen todennäköisyys. (Sakhouf ym., 2020)

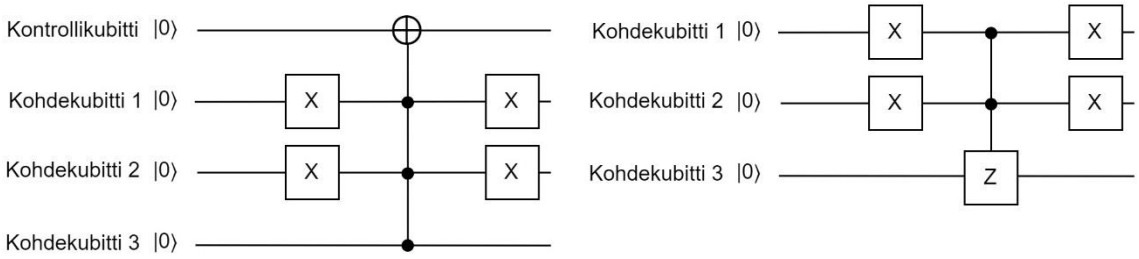


Kuva 5. *Kahden kubitin tietokannan Groverin algoritmi vaihemuunnosten avulla. Perustuu (Sakhouf ym., 2020, s. 3438–3443)-*

5.2 Kolmen kubitin tietokanta

Kolmen kubitin tietokannan Groverin algoritmi tehdään kirjallisuudessa kahden kubitin tietokannan tavoin joko Boolen logiikkaa tai vaihemuunnosta hyödyntävällä oraakkelilla. Tutkimuksessa, missä Groverin algoritmi toteutettiin kolmen kubitin toteutuksena, käytettiin viiden kubitin kvanttitietokonetta, joka koostui viidestä Yb^+-ionista. Oraakkeli toteutettiin Boolen logiikalla sekä vaihemuunnoksilla. (Figgatt ym., 2017)

Boolen logiikkaa käyttävät oraakkelit toteutettiin aiemmin esitellyn kahden kubitin tietokannan toteutuksen tavoin käyttämällä Toffoli-portteja (kuva 6). Johtuen yhdestä uudesta kubitista tietokannassa, porttina toimi kuitenkin Toffoli-3-portti, jossa porttiin on lisätty yksi uusi kohdekubitti. Portin molemmille puolille oli asetettuna eri määrä X-portteja riippuen, mitä alkiota oraakkeli etsi. Vaihemuunnosta käyttävät oraakkelit oli C-vaiheportin sijasta toteutettu kontrolloiduilla Z-porteilla sekä X-porteilla (kuva 6).



Kuva 6. *Oraakkelin toteutus kolmen kubitin tietokannassa etsittäessä alkiota (0,0,1). Vasemmalla Boolen logiikkaa ja oikealla vaihemuunnosta käyttävä toteutus. Perustuu (Figgatt ym., 2017, s. 4).*

Tutkimuksessa oikean alkion löytämisen todennäköisyys Boolen logiikkaa käyttävillä oraakkeleilla oli 38,9 % kun taas vaihemuunnoksia käyttävillä oraakkeleilla todennäköisyys oli 43,7 %. Teoreettisesti alkion löytäminen yhden Groverin operaation toteutuksella on 78,1 % kolmen kubitin tietokannassa. Tutkimuksessa kokeiltiin myös Groverin algoritmia tilanteessa, jossa kaksi kahdeksasta alkiosta oli oikea vastaus. Tällöin 100 % teoreettisella todennäköisyydellä Boolen logiikkaa hyödyntämä toteutus sai 67,9 % todennäköisyyden ja vaihemuunnosta käyttämällä 75,3 %. (Figgatt ym., 2017)

5.3 Neljän kubitin tietokanta

Neljän kubitin tietokantoihin tehty Groverin algoritmi on löydettyjen tutkimusten perusteella toteutettu käyttämällä edellisten tietokantojen kokojen tavoin vaihemuunnosta, että Boolen logiikkaa hyödyntävää oraakkelia.

Ensimmäinen tutkimus on tehty käyttämällä IBM:n ibmqx5-arkkitehtuuria, jossa arkkitehtuurin mukainen kvanttitietokone on 16 suprajohtavan transmon-kubitin suuruinen. Kyseisessä tutkimuksessa luotiin neljän kubitin tietokanta ja siitä etsivä Groverin algoritmi käyttäen vaihemuunnosta hyödyntävää oraakkelia. Neljän kubitin toteutus todettiin tutkimuksessa sen hetken suurimmaksi määräksi toteuttaa Groverin algoritmi ibmqx5-tietokoneella sen teknisten rajoituksien vuoksi, vaikka kubittien määrä mahdollistaisi suuremman toteutuksen. (Strömberg & Karlson, 2018)

Todennäköisyys löytää oikea alkio yhdellä Groverin operaatiolla neljän kubitin tietokannasta on teoreettisesti keskimäärin 47,27 %. Tutkimuksen tulokseksi saatiin alkion

löytämisen keskiarvoksi 6,62 %, joka on 16 mahdollisen tuloksen joukosta vain 0,37 % yli keskiarvon, jos tulos otettaisiin satunnaisesti. Toteutus tehtiin myös kvanttitietokonetta simuloivalla ohjelmalla, joka antoi tulokseksi 47,39 %, joka on lähellä teoreettista keskiarvoa.

Tutkimuksessa suurta eroa toteutuksen ja teoreettisen sekä simuloidun tuloksen välillä epäiltiin laitteiston kapasiteetin riittämättömyydellä, joka johti suoritusajan kasvuun ja tätä myöten porteissa tapahtuviin virheisiin. Tämän lisäksi dekoherenssi esitettiin toisena ilmiönä alhaiseen algoritmin suorituskykyyn. Tutkimus toteaa algoritmin onnistumisen simulaatiossa tukevan tätä hypoteesia. (Strömberg & Karlson, 2018, s. 25)

Toisessa tutkimuksessa oraakkeli toteutettiin käyttämällä Boolen logiikkaa hyödyntävää oraakkelia. Laitteistona käytettiin IBM:n Canary r3 -prosessoria käyttävää 5 kubitin kvanttitietokonetta. Tutkimuksen tulokset antoivat algoritmin onnistumisen todennäköisyydeksi 8192 algoritmin suorituskerran keskiarvona 5,93 %. Kuten edellä mainittiin, teoreettinen keskiarvo neljän kubitin tietokannasta löytämiselle yhdellä Groverin operaatiolla on 47,27 %. (Joshi & Gupta, 2021) Huomataan, että algoritmin onnistumisen todennäköisyys on itseasiassa alhaisempi, kuin mikä satunnaisesti valitsemalla olisi saatu.

Tutkimus toteutti algoritmin myös kahden sekä kolmen kerran Groverin operaatiolla. Kahdella operaatiolla todennäköisyys löytää alkio neljän kubitin tietokannasta on teoreettisesti 78,13 % ja kolmella operaatiolla 84,67 %. Keskiarvona kahdella operaatiolla algoritmi löysi oikean alkion 4,97 % todennäköisyydellä ja kolmella operaatiolla todennäköisyys oli 5,73 %. Todennäköisyys löytää oikea alkio aleni suuremmalla määrällä operaatioita, vaikka teoreettisesti sen olisi pitänyt kasvaa.

Groverin algoritmi toteutettiin tutkimuksessa myös simulaatiossa edellä esitellyn tutkimuksen tavoin. Alkion löytämisen todennäköisyyden arvoksi simulaatiossa saatiin myös tässä tutkimuksessa lähelle teoreettisia todennäköisyyksiä. Tutkimuksen hypoteesi suurille eroille todellisen toteutuksen ja simulaation sekä teoreettisen arvon välille epäillään kvanttitiekoneessa tapahtuvan kvanttikohinan aiheuttamien virheellisten tulosten takia. Lisäksi eroavaisuus laitteiston topologiassa sekä kubittien laadun heikkous esitetään mahdollisina syinä vääriin tuloksiin. (Joshi & Gupta, 2021, s. 147)

6. TULOKSET JA YHTEENVETO

Tässä tutkielmassa selvitettiin Groverin algoritmin soveltuvuutta hakualgoritmina. Selvitys tapahtui vertaamalla Groverin algoritmin asymptoottista tehokkuutta perinteisiin hakualgoritmeihin haettaessa yhtä alkiota tietokannasta. Tämän lisäksi tehtiin kirjallisuuskatsaus, jossa tarkasteltiin tutkimuksia, missä Groverin algoritmi toteutettiin kvanttitietokoneella ja algoritmin suorituskykyä tarkasteltiin.

Verrattaessa Groverin algoritmin ja peräkkäishaun asymptoottista tehokkuutta selvisi, että elleivät tietokantaan tehtävät hakuoperaatiot kohdistu keskimäärin <= $\Theta(\sqrt{N})$ alkioon, on Groverin algoritmi keskimäärin asymptoottiselta tehokkuudeltaan peräkkäishakua parempi. Muussa tapauksessa peräkkäishaun asymptoottinen tehokkuus on $\Theta(N)$ ja Groverin algoritmin $O(\sqrt{N})$ asymptoottinen tehokkuus on sitä parempi.

Groverin algoritmin vertauksessa puolitushakuun saatiin selville epäyhtälö 8, jossa on esitetty se todennäköisyys tietokannan järjestämisoperaation ja hakuoperaation välillä tietyllä *N*, jolla Groverin algoritmi on asymptoottisesti puolitushakua tehokkaampi. Selvisi, että tietokannan alkioiden määrän kasvaessa, Groverin algoritmi yhä useammin on puolitushakua asymptoottisesti tehokkaampi. Tietokannan kuitenkin ollessa staattinen puolitushaku voittaa Groverin algoritmin $O(log\ (N)) < O(\sqrt{N})$.

Hajautustauluun verrattaessa Groverin algoritmi osoittautui asymptoottisesti tehokkaammaksi miltei jokaisessa tarkastellussa. Kuitenkin tietokannassa, missä on mahdollista luoda $\sqrt{N}$ avainta, hajautustaulun asymptoottinen tehokkuus on keskimäärin Groverin algoritmin asymptoottisen tehokkuuden suuruinen.

Esitelty vertailu algoritmien välillä perustui ainoastaan yhden oikean alkion etsimisestä tietokannasta. Tutkielma ei ottanut kantaa useamman alkion etsimisestä tietokannasta. Lisäksi tutkielma ei selvittänyt Groverin algoritmin suorituskykyä haasteellisimmissa hakuoperaatioissa, kuten reittien noutamisesta tietokannassa. Vertailu kuitenkin antoi viitteitä siitä, että tietokannassa, jossa alkioiden oletetaan lisääntyvän tai muuttumaan riittävällä todennäköisyydellä, Groverin algoritmi on teoreettisesti tehokkaampi perinteisiin hakualgoritmeihin nähden. Väite olettaa, että ei ole olemassa perinteistä hakualgoritmia, jonka asymptoottinen tehokkuus olisi tarkasteltuja algoritmeja vähäisempi.

Kirjallisuuskatsauksessa esiteltiin kuusi tutkimusta, jotka käsittelivät Groverin algoritmin onnistumista oikealla kvanttitietokoneella. Tutkimuksen tulokset on asetettu taulukkoon 1, jossa ilmoitetaan kubittien lukumäärä tietokannassa, käytetyn oraakkelin arkkitehtuuri,

laitteisto sekä algoritmin suoriutuminen kokeessa verrattuna sen teoreettisen onnistumisen todennäköisyyteen, kun Groverin operaatio toistetaan yhden kerran.

Taulukko 1. *Groverin algoritmin suoriutuminen tutkimuksissa.*

Kubittien lukumäärä	Oraakkeli	Laitteisto	Algoritmin onnistuminen	Teoreettinen onnistuminen
2	Boolen logiikka	Ibmqx4	64,6 %	100 %
2	Vaihemuunnos	Kahden suprajohtuvan kubitin kvanttitietokone	>90 %	100 %
3	Boolen logiikka	Viiden Yb^+-ionin kvanttitietokone	38,9 %	78,1 %
3	Vaihemuunnos	Viiden Yb^+-ionin kvanttitietokone	43,7 %	78,1 %
4	Boolen logiikka	IBM Canary r3	5,9 %	47,4 %
4	Vaihemuunnos	Ibmqx5	6,6 %	47,4 %

Taulukosta 1 havaitaan, kuinka suhteellinen ero teoreettisen onnistumisen ja algoritmin todellisen onnistumisen välillä kasvaa kubittien lukumäärän kasvaessa. Havaitaan, että kubittien lukumäärän ollessa neljä, on todennäköisyys kasvanut niin pieneksi, että algoritmi löytää alkion keskimäärin noin samalla todennäköisyydellä, mikä alkion ottamisella satunnaisesti saataisiin. Todetaan, että käytännössä Groverin algoritmi ei suoriudu tällä hetkellä riittävän hyvin tietokannasta tehdyistä hauista kvanttitietokoneilla, johtuen tutkimuksien perusteella siitä, että laitteistojen suorituskyky ei riitä ylläpitämään kvanttimekaanisia tiloja algoritmin toimivuuden mahdollistamiseksi.

Tuloksista voidaan kuitenkin huomata, että tutkimuksissa vaihemuunnosta käyttävä oraakkeli löytää suuremmalla todennäköisyydellä halutun alkion tietokannasta jokaisessa tarkastellussa tietokannan koossa. Ottaen huomioon kuitenkin tutkimuksien otannan määrän, varmuutta siitä, että vaihemuunnosta käyttävä oraakkeli löytäisi alkion todennäköisemmin ei voida varmistaa. Voidaan kuitenkin pitää todennäköisenä, että vaihemuunnosta käyttävä oraakkeli voisi olla arkkitehtuuriltaan paremmin resistanttinen mainittuihin laitteiston ongelmiin.

Tutkielmassa selvisi, että Groverin algoritmin suorituskyvyn erot teoreettisen ja käytännön välillä ovat huomattavia. Vaikka Groverin algoritmi pystyy teoreettisilla arvoilla olemaan perinteisiä hakualgoritmeja tehokkaampi, käytännössä Groverin algoritmi ei pystynyt tutkimuksissa löytämään alkiota riittävän suurella todennäköisyydellä tietokannasta. Groverin algoritmin soveltuvuuden hakualgoritmina tulee ratkaisemaan kvanttitietokoneiden teknologinen kehitys tulevaisuudessa. Jos kvanttitietokoneet pystyvät

vastamaan kirjallisuuskatsauksen tutkimuksissa esitettyihin ongelmiin, Groverin algoritmin suorituskyvyn oletetaan kasvavan lähelle teoreettista tasoa. Siinä tapauksessa Groverin algoritmia voidaan tutkielman tuloksiin vedoten pitää soveltuvana hakualgoritmiksi.

LÄHTEET

Abhijith, J., Adedoyin, A., Ambrosiano, J., Anisimov, P., Casper, W., Chennupati, G., Coffrin, C., Djidjev, H., Gunter, D., Karra, S., Lemons, N., Lin, S., Malyzhenkov, A., Mascarenas, D., Mniszewski, S., Nadiga, B., O'Malley, D., Oyen, D., Pakin, S., … Lokhov, A. Y. (2022). Quantum Algorithm Implementations for Beginners. *ACM Transactions on Quantum Computing*, 3517340. https://doi.org/10.1145/3517340

Balynsky, M., Chiang, H., Gutierrez, D., Kozhevnikov, A., Filimonov, Y., & Khitun, A. (2021). Quantum computing without quantum computers: Database search and data processing using classical wave superposition. *Journal of Applied Physics*, *130*(16), 164903. https://doi.org/10.1063/5.0068316

Cormen, T. H. (Toim.). (2009). *Introduction to algorithms* (3rd ed). MIT Press.

De, P., Ranwa, S., & Mukhopadhyay, S. (2022). Intensity and phase encoding for realization of integrated Pauli X, Y and Z gates using 2D photonic crystal. *Optics & Laser Technology*, *152*, 108141. https://doi.org/10.1016/j.optlastec.2022.108141

Figgatt, C., Maslov, D., Landsman, K. A., Linke, N. M., Debnath, S., & Monroe, C. (2017). Complete 3-Qubit Grover search on a programmable quantum computer. *Nature Communications*, *8*(1), 1918. https://doi.org/10.1038/s41467-017-01904-7

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*, 212–219. https://doi.org/10.1145/237814.237866

Hester, J. H., & Hirschberg, D. S. (1985). Self-organizing linear search. *ACM Computing Surveys*, *17*(3), 295–311. https://doi.org/10.1145/5505.5507

Joshi, S., & Gupta, D. (2021). Grover's Algorithm in a 4-Qubit Search Space. *Journal of Quantum Computing*, *3*(4), 137–150. https://doi.org/10.32604/jqc.2021.018114

Kommadi, B. (2021). *Quantum Computing Solutions Solving Real-World Problems Using Quantum Computing and Algorithms*. http://www.vlebooks.com/vleweb/product/openreader?id=none&isbn=9781484265161

Leider, A., Siddiqui, S., Sabol, D. A., & Tappert, C. C. (2020). Quantum Computer Search Algorithms: Can We Outperform the Classical Search Algorithms? Teoksessa K. Arai, R.

Bhatia, & S. Kapoor (Toim.), *Proceedings of the Future Technologies Conference (FTC) 2019* (Vsk. 1069, ss. 447–459). Springer International Publishing. https://doi.org/10.1007/978-3-030-32520-6_34

Sakhouf, H., Daoud, M., & Laamara, R. A. (2020). Implementation of Grover's Search Algorithm in the QED Circuit for Two Superconducting Qubits. *International Journal of Theoretical Physics*, *59*(11), 3436–3448. https://doi.org/10.1007/s10773-020-04602-1

Strömberg, P., & Karlson, V. (2018). *4-qubit Grover's algorithm implemented for the ibmqx5 architecture* [School of Electrical Engineering and Computer Science]. http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1214481&dswid=3449

Vrana, P., Reeb, D., Reitzner, D., & Wolf, M. M. (2014). Fault-ignorant quantum search. *New Journal of Physics*, *16*(7), 073033. https://doi.org/10.1088/1367-2630/16/7/073033

Wang, Y. (2012). Quantum Computation and Quantum Information. *Statistical Science*, *27*(3). https://doi.org/10.1214/11-STS378

Zalka, C. (1999). Grover's quantum searching algorithm is optimal. *Physical Review A*, *60*(4), 2746–2751. https://doi.org/10.1103/PhysRevA.60.2746