KVANTEVANDRINGER

THOMAS WILSKOW THORBJØRNSEN

15. august 2021

INNHOLD

1 Introduksjon 3
2 Kvanteberegninger 3
2.1 Postulatene i kvantemekanikk . . . 3
2.2 Qubits og kvantekretser . . . 5
2.3 Kvantealgoritmer og orakler . . . 7
3 Kvantevandringer 8
3.1 Grover's algoritme og metoden av amplitude amplifikasjon . . . 8
3.2 Kvantevandringer basert på kvantemyntkast . . . 11
3.3 Umyntede kvantevandringer . . . 16
3.4 Kvantesøk . . . 20
4 Q# og Implementasjon av kvantevandringer 21
4.1 Q# intro . . . 21
4.2 Kvantevandringer . . . 24
4.3 Kvantesøk . . . 30
A Q# intro kode 32

FIGURER

Figur 1 EPR-sammenfiltrings kvantekrets . . . 6
Figur 2 Kontrollerte kvanteporter . . . 7
Figur 4 Standardkonstruksjon av faseorakel . . . 8
Figur 5 Grover's algoritme . . . 9
Figur 7 4-regulær graf . . . 12
Figur 9 Simpel graf med løkker . . . 15
Figur 10 Shuriken graf . . . 17
Figur 12 Grover's algoritme kjørt på 0100010000000010 . . . 23
Figur 13 Kvantevandring over 7 . . . 25
Figur 15 Kvantevandring over 9 . . . 26
Figur 17 Kvantevadnring over 10 . . . 28
Figur 19 QSS søk over 10, med tidligere kvantevandring . . . 30

ABSTRAKT

I denne rapporten utreder vi forskjellige kvantevandringsalgoritmer. Vi skal spesielt se på Grover's algoritme, position-coin notation, arc-notation og the staggered model. Dette er forskjellige algoritmer som man kan anvende for å utføre graftraversering. Disse graftraverseringene blir så implementert og testet med programmeringsspråket Q#.

1 INTRODUKSJON

I klassisk datavitenskap er sortering, søk og optimering viktige klasser med problemer. Disse problemene kan modelleres til å operere over ulike datatyper. Blant sortering er det ofte lurt å modellere dataen i binærtrær, i søk er det hjelpsomt å modellere datatypen som grafer og optimering kan ta form som maksimal flyt i et flytnettverk. For å løse disse problemene står graf traversering sentralt. Bredde først og dybde først algoritmene bruker ofte som en underrutine i de algoritmene vi har for å løse disse problemene. Man kan også vise fra optimeringsteori via den Ungarske metoden at bruken av disse traverseringsalgoritmene faktisk gir optimale algoritmer.

I denne rapporten skal vi undersøke noen kvante graf traverseringsalgoritmer over endelige grafer. Vi starter med å gi en kort introduksjon til kvantemekanikk og kvanteberegninger, før vi ser på kvantevandringsalgoritmer. Her starter vi med Grover's algoritme, før denne generaliseres til kvantevandringer over endelige grafer. Til slutt ser vi på en implementasjon i programmeringsspråket Q# og hvordan en kvantevandring kan se ut.

2 KVANTEBEREGNINGER

2.1 Postulatene i kvantemekanikk

Kvantemekanikk er en beskrivelse av fysiske systemer på små størrelser. Reglene for hvordan disse systemene oppfører seg er formulert utifra 6 postulater. [1] og [2] definerer postulatene som følger

1. På hvert øyeblikk er tilstanden til det fysiske systemet beskrevet av en ket $|\psi\rangle$ i rommet av tilstander.

2. Enhver observabel fysisk egenskap av systemet er beskrevet av en operator som virker på ketten som beskriver systemet.

3. De eneste mulige resultatene av en måling av en observabel $\mathcal{A}$ er egenverdiene til den assosierte operatoren A.

4. Når en måling er gjort på en tilstand $|\psi\rangle$ er sannsynligheten for å få en egenverdi a_n gitt ved kvadratet av indreproduktet til $|\psi\rangle$ sammen med egenvektoren $|a_n\rangle$.

$$p_{a_n} = |\langle a_n|\psi\rangle|^2$$

5. Umiddelbart etter en måling av en observabel $\mathcal{A}$ har gitt egenverdien a_n, er systemet i tilstanden til den normaliserte egenprojeksjonen $P_{a_n}|\psi\rangle$.

6. Tidsutviklingen til et system bevarer normen til en ket $|\psi\rangle$.

For å forklare postulatene sammen med et eksempel antar vi at det finnes en partikkel som har to observable tilstander, vi kaller de spin opp og spin ned, som er beskrevet av vektorer i et rom av tilstander. Her tolkes rommet av tilstander som et (kompleks separabelt) Hilbertrom. En tilstand eller ket $|\psi\rangle$ er dermed en vektor i $\mathcal{H}$. Siden $\mathcal{H}$ er et Hilbertrom har den også en basis $\{|\beta_\lambda\rangle \mid \lambda : \Lambda\}$, hvor Λ er en indeksmengde. Ettersom vi har to observable tilstander holder det å anta at $\mathcal{H} = \mathbb{C}^2$ og at $|\beta_i\rangle = e_i$ for $i = 1, 2$. Man kan skrive $|\psi\rangle$ som en lineærkombinasjon av basisen, dette er også kalt for en superposisjon av tilstandene $\{|\beta_\lambda\rangle \mid \lambda : \Lambda\}$ (eller $\{e_1, e_2\}$ for spin eksemplet).

$$|\psi\rangle = \Sigma\psi_i|\beta_i\rangle$$
$$(|\psi\rangle = \psi_1 e_1 + \psi_2 e_2)$$

Gitt at vi har en observable $\mathcal{A}$, altså en fysisk egenskap ved systemet som kan måles, så vet vi at dette er gitt ved en lineærtransformasjon A som virker på Hilbertrommet $\mathcal{H}$. De fysiske målingene til systemet skal være gitt ved egenverdiene av denne lineærtransformasjonen, noe som krever den til å være en endomorfi, aka. $A : \mathcal{H} \to \mathcal{H}$. I spin eksemplet kan man måle spin opp og spin ned med en observable hvor f.eks. spin opp har egenverdien 1 og spin ned har egenverdien -1. En vanlig antagelse er at alle de observable egenskapene skal være reelle verdier, derfor velger man i tillegg å anta at A må være en Hermitisk operator (selvadjungert).

Når man gjør en måling av en observabel er det tilfeldig hva man måler. Sannsynlighetene for å måle de forskjellige egenverdiene er gitt ved formelen over. Etter måling vil systemet kollapse ned i egenrommet til vektoren $\frac{1}{\sqrt{p_{a_n}}} P_{a_n}|\psi\rangle$. Hvis den algebraiske multiplisiteten til egenverdien a_n er 1 så tilsvarer dette vektoren $\frac{|a_n\rangle}{|||a_n\rangle||}$. Målinger som er beskrevet av projeksjons operatorer kalles for projektive målinger. Det finnes også delvise målinger, hvor man ikke kollapser hele systemet etter en måling.

En konsekvens av målinger er at observabelen som har tilstanden $|\psi\rangle$ som en egenvektor med egenverdi lik 1 vil ikke endre tilstanden til systemet etter måling. Hvis man derimot måler denne observabelen, vil man normalisere tilstanden. I spin eksemplet vil dette si at hvis man måler verdien 1, så vil tilstanden kollapse til den tilsvarende egenvektoren, normalisert. En konsekvens av dette er at en tilstand er bedre definert som ekvivalensklasser langs linjer i Hilbertrommet. En tilstand er dermed et element i randen av enhetskulen til Hilbertrommet.

$$|\psi\rangle : \partial D(\mathcal{H}) = \{v : \mathcal{H} \mid \|v\| = 1\}$$

Det siste postulatet forteller oss hvordan et system utvikler seg. Formelen $|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle$ brukes ofte for å beskrive hvordan dette ser ut. Siden vi krever at $U(t, t_0)$ skal bevare normen til $|\psi(t_0)\rangle$, dvs. at det finnes en virkning $U(t, t_0) : \partial D(\mathcal{H}) \to \partial D(\mathcal{H})$, følger det at denne operatoren er unitær. Mengden $U(\mathcal{H})$ vil betegne de unitære operatorene som operer på det Hilbertrommet. For vårt formål kan man tenke på et kvantesystem som et element i $U(\mathcal{H})$-mengden $\partial D(\mathcal{H})$.

Som beskrevet av [2] kan man slå sammen kvantesystemer med det algebraiske tensorproduktet. Gitt to foskjellige kvantesystemer beskrevet av to forskjellige Hilbertrom $\mathcal{H}_1$ og $\mathcal{H}_2$ så er rommet av sammensatte tilstander $\mathcal{H}_1 \otimes \mathcal{H}_2$. Vi får da en klasse med observable og en klasse med operatorer som virker på systemene gjennom tensorproduktet. Man kan vise at tensorproduktet av to unitære og hermitiske matriser er igjen unitære og hermitiske, det er derfor veldefinert å betrakte tensoren for

sammensatte systemer. Sammenfiltringsfenomenet foregår når man konstruerer slike sammensatte systemer. Hvis vi antar at vi har to partikler med spin egenskapen ϕ og ψ og det sammensatte systemet $|\phi\psi\rangle = \sigma_0 e_0 \otimes e_0 + \sigma_1 e_1 \otimes e_1$, så vil en måling av den ene partikkelen ende opp med å måle den andre partikkelen. Dersom man måler egenverdien til e_0 for ϕ så vil systemet kollapse til $\frac{1}{\sigma_0} P_1 \otimes I(|\phi\psi\rangle) = e_0 \otimes e_0$. Dette medfører at alle målinger av ψ vil gi egenverdien 1.

2.2 Qubits og kvantekretser

Klassiske bits har to tilstander: 0 eller 1. Kvantebits, eller qubits er et fysisk system som har en observabel som måler to diskrete tilstander. Disse tilstandene bruker vi for å representere 0 og 1. Ettersom at operatoren som måler 0 og 1 er hermitisk, så finnes det en ortonormal basis for Hilbertrommet som diagonaliserer denne operatoren. Elementene i denne basisen vil bli betegnet som $|0\rangle$ og $|1\rangle$. En qubit q er dermed et element i $\partial D(\mathbb{C}^2)$ på formen $q = q_0|0\rangle + q_1|1\rangle$.

En streng av bits er sammensettingen av flere bits. På samme måte konstruerer vi en streng av qubits til å være sammensettingen av flere qubits. Denne sammensettingen er gitt av tensorproduktet mellom de algebraiske qubitsene. F.eks. er en 2-qubit streng er et element i $\partial D(\mathbb{C}^2 \otimes \mathbb{C}^2)$ på formen under.

$$q = q_{00}|0\rangle \otimes |0\rangle + q_{01}|0\rangle \otimes |1\rangle + q_{10}|1\rangle \otimes |0\rangle + q_{11}|1\rangle \otimes |1\rangle$$

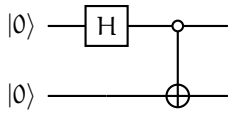
For kortfatthetens skyld skriver vi $|ab\rangle = |a\rangle|b\rangle = |a\rangle \otimes |b\rangle$. Notasjonen $|\_\rangle$ vil få en ekstra presisjon i denne rapporten som ikke brukes andre steder. La $n\mathtt{Bit}$ være mengden av strenger med n-bits, vi definerer $|\_\rangle : \bigcup_{n=0}^{\infty} n\mathtt{Bits} \to \bigoplus_{n=1}^{\infty} \mathbb{C}^{2^{\otimes n}}$ til å være en funksjon fra alle strenger og inn i tensoralgebraen til $\mathbb{C}^2$. Den er definert på $|0\rangle$ og $|1\rangle$ som over, også utvides den lineært og fritt over tensoralgebraen. En av de viktigste egenskapene qubits har som bits ikke har er nemlig at to eller flere qubits kan bli sammenfiltret.

EPR paret (Einstein, Rosen og Podolsky) er et gjenngående eksempel på sammenfiltring av qubits. Man kan se at et sammensatt system av qubits er sammenfiltret hvis det ikke kan skrives som en elementær tensor, $a \otimes b$. Et EPR par er et 2-qubit system på formen $\psi = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Man kan se at dette systemet er sammenfiltret, ettersom de to elementære tensorene ikke har noen felles faktorer. Hvis vi derimot måler den første qubiten i systemet vil vi ende opp med at det er en $(\frac{1}{\sqrt{2}})^2 = 50\%$ sjanse for å måle 0 og 50% sjanse for å måle 1. Hvis vi derimot har målt 0 på den første qubiten, så vil systemet kollapse til $\psi = |00\rangle$, og vi vet dermed at den andre qubiten må være i tilstand $|0\rangle$.

På samme måte som at klassiske bits kan manipuleres med kretser, kan man manipulere qubits med kvantekretser. En kvantekrets er et flytdiagram med et register, en arbeidsplass, logiske kvanteporter og målinger. Registeret er inputtet av qubits, arbeidsplassen er tilleggs qubits som man kan bruke til å utføre/definere operasjoner. Se figur 1 for et eksempel av en kvantekrets. I motsetning til klassiske kretser kan ikke kvantekretser ødelegge qubits, og alle prosessene må være unitære og reversible. Alle logiske kvanteporter er derfor unitære transformasjoner. Målinger følger ikke disse reglene, og disse er gitt ved hermitiske operatorer. Bemerk at en måling gjør om en qubit om til en klassisk bit.

De elementære logiske kvanteportene er unære, binære og trinære unitære operatorer over $\mathbb{C}^2$. De unære operatorene er kjent som Pauli matrisene I, X, Y, Z, sammen

Figur 1: EPR-sammenfiltrings kvantekrets



H boksen viser til at man bruker Hadamard operatoren, mens sirkelen kontrollerer en X operator som er $\oplus$.

med Hadamard operatoren H og fase skift operatoren R_θ. Man kan observere at X operatoren flipper qubiten, Z operatoren snur fasen hvis argumentet var $|1\rangle$ og Y operatoren er en kombinasjon av X og Z ganget med skalaren i.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \text{ og } R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

Den elementære binære operatoren kalles controlled not og skrives CNOT. CNOT fliper qubiten til det andre argumentet hvis den første qubiten er $|1\rangle$. SWAP porten er en binær port som bytter om rekkefølgen på argumentene.

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},\ SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

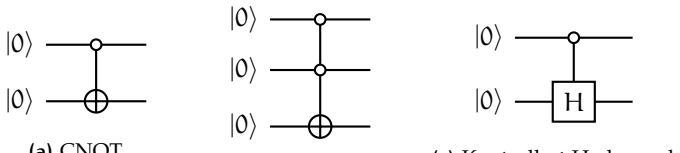
Den trinære porten som er av stor interesse er Toffoli porten. Toffoli porten kalles også CCNOT, ettersom det er en dobbel kontrollert not. Hvis de to første argumentene har verdien $|1\rangle$ så flippes qubiten i det tredje argumentet.

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Disse portene er universelle i den forstand av at alle andre logiske kvanteporter kan uttrykkes som en komposisjon av disse portene. Vi har i tillegg at Hadamard porten og Toffoli porten kan konstruere alle porter med reelle innlegg, som beskrevet av [3].

Som sagt tidligere er CNOT og Toffoli portene kontrollerte porter. En kontrollert port er en port som kun blir aktivert, gitt at tilstanden til en annen qubit tilfredstiller en betingelse. CNOT er kontrollert i den forstanden at man kun anvender X operatoren hvis den første qubiten er i tilstanden $|1\rangle$. Toffoli porten er et eksempel

Figur 2: Kontrollerte kvanteporter



(a) CNOT **(b)** Toffoli **(c)** Kontrollert Hadamard

på en port som er multikontrollert. Alle logiske kvanteporter kan kontrolleres av andre qubits. Se figur 2 for eksempler.

2.3 Kvantealgoritmer og orakler

Klassiske algoritmer er metoder som løser problemer basert på input av bits, kvantealgoritmer kan dermed ses på som metoder som løser problemer basert på qubits. Bits brukes for å representere datastrukturer som tall, lister og grafer. Qubits kan brukes for å representere de samme strukturene. Kvantekretser blir dermed den naturlige måten for å representere algoritmene, en kvantealgoritme er dermed en komposisjon av unitære operatorer og målinger på en tilstand ψ i $\partial D(\mathcal{H})$.

Kvanteparallellisme er en egenskap kvantealgoritmer får fra kvantemekanikken. Dette fenomenet er beskrevet som at en beregning kan inneholde informasjonen fra flere. For å se dette ser vi på en funksjon $f : n\mathrm{Bits} \to 1\mathrm{Bits}$ og vi antar at det finnes en unitær operator $\mathcal{O}_f$ slik at $\mathcal{O}_f(|z\rangle |0\rangle) = |z\rangle |f(z)\rangle$. Ved å anvende $\mathcal{O}_f$ på en tilstand som er i en superposisjon av alle baisiselementene får man følgende:

$$\mathcal{O}_f(\Sigma_{z=0}^{n} |z\rangle |0\rangle) = \Sigma_{z=0}^{n} |z\rangle |f(z)\rangle .$$

Man kan se at $\mathcal{O}_f$ har kun blitt anvendt en gang, men informasjon om alle evalueringene er i den nye tilstanden. Når man måler tilstanden i standard basisen vil den kollapse til en av evalueringene, så klassisk er ikke Kvanteparallellisme noe bedre, men interferens og sammenfiltring kan gi effekter som gir bedre utslag enn med klassiske algoritmer.

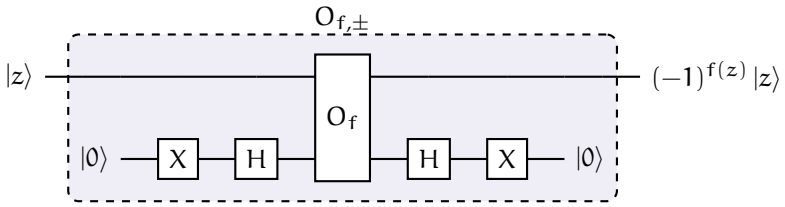
Nesten alle kvantealgoritmer bruker en slags *query*. Disse kommer som oftest i form som en evaluering av en klassisk funksjon. Den unitære operatoren ovenfor er et eksempel på en slik *query*. De operatorene som utfører *queries* kalles for orakler eller black-boxes. En unitær operator $\mathcal{O} : \mathcal{H} \otimes \mathcal{H}' \to \mathcal{H} \otimes \mathcal{H}'$ som gjør en *query* på rommet $\mathcal{H}$ og merker tilstandene i $\mathcal{H}'$ basert på utfallet kalles for et merkeorakel. Operatoren $\mathcal{O}_f$ som definert over er et eksempel på et merkeorakel. En annen klasse med orakler er faseorakler, disse er operatorer på formen $\mathcal{O}_\pm : \mathcal{H} \to \mathcal{H}$, disse gjør en *query* på rommet $\mathcal{H}$ og endrer fasen basert på utfallet.

I tilfellet med merkeorakelet $\mathcal{O}_f$, så finnes det en metode for å gjøre det om til et faseorakel $\mathcal{O}_{f,\pm}$. Bemerk først at merkeorakelet er definert som $\mathcal{O}_f(|z\rangle |w\rangle) = |z\rangle |w \oplus f(z)\rangle$ på basisen. Vi kan definere $\mathcal{O}_{f,\pm}$ som følgende:

$$\begin{aligned} &\mathcal{O}_f(|z\rangle \otimes H |1\rangle) = (-1)^{f(z)} |z\rangle \otimes H |1\rangle \\ \implies &\mathcal{O}_{f,\pm}(|z\rangle) = (-1)^{f(z)} |z\rangle \end{aligned}$$

Figur 4 beskriver hvordan denne konstruksjonen ser ut som med kvantekretser.

Figur 4: Standardkonstruksjon av faseorakel



3 KVANTEVANDRINGER

3.1 Grover's algoritme og metoden av amplitude amplifikasjon

Grover's algoritme løser problemet med ustrukturert søk, og teknikken amplitude amplifikasjon som den bruker er av stor interesse. Problemet går som følger: Tenk at man er gitt en bistring med $N = 2^n$ bits, hvor t bits er satt til 1. Finn minst 1 bit som har verdi 1. Dette problemet kan åpenbart løses i "worstcase" lineær tid med konstant minne ved å randomisert iterere gjennom alle bitene og sjekke om de er 1 eller 0. Om den er 1 kan man terminere programmet, og returnere den posisjon som ga 1. Grover's algoritme har en kvadratisk hastighetsøkning på dette problemet, og man kan dermed løse det i worst case kvadratisk tid.

For å beskrive problemet med et fysisk kvantesystem trenger vi å oversette problemet først. La $(b_k)_N$ være bitstringen med lengde N, definer så orakelet $\mathcal{O}_{(b_k)_N} : \mathbb{C}^{2^n} \otimes \mathbb{C}^2 \to \mathbb{C}^{2^n} \otimes \mathbb{C}^2$ til å merke målbiten hvis registerbiten var en løsning. Dette vil si at hvis $\mathcal{O}_{(b_k)_N}(|r\rangle \otimes |0\rangle) = |r\rangle \otimes |1\rangle$ så følger det at $b_r = 1$. For å fullføre Grover's algoritme trenger man matrisen R som flipper fortegnet til registeret hvis den ikke er tilstanden $|0\rangle^{\otimes n}$.

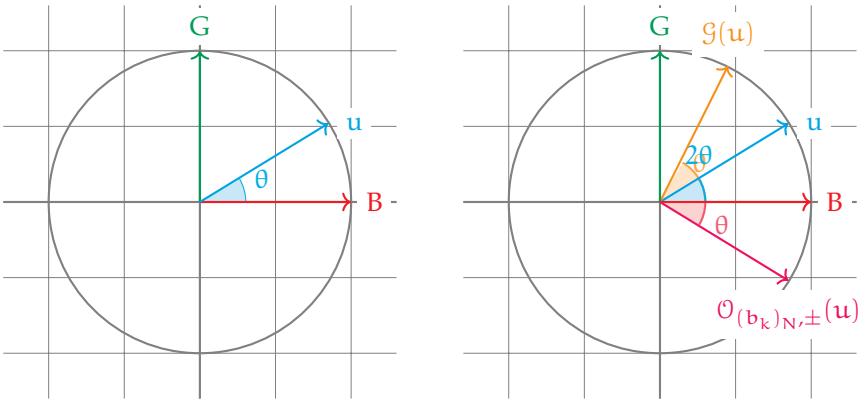
$$R = \begin{pmatrix} 1 & 0 & \dots \\ 0 & -1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

En Grover iterate $\mathcal{G}$ er definert som

$$\mathcal{G} = H^{\otimes n} R H^{\otimes n} \mathcal{O}_{(b_k)_N,\pm}.$$

Gorver's algoritme er komposisjonen av operatorene $G = M\mathcal{G}^k \circ H^{\otimes n}$, hvor M er en projektiv måling, og k er en konstant. Man skal da kunne fastslå med en høy sannsynlighet at målingen gir deg posisjonen til en bit i $(b_k)_N$ som er 1. For å finne denne k-en som man bruker for å kjøre algoritmen trenger vi å se på metoden av amplitude amplifikasjon.

Figur 5: Grover's algoritme



(a) Initiell tilstand **(b)** En Grover iteration

Definer tre tilstander hvor t er antall 1-ere i $(b_r)_N$

$$u = H^{\otimes n}|0\rangle^{\otimes n}$$
$$G = \frac{1}{\sqrt{t}}\Sigma_{|r\rangle|b_r=1}|r\rangle$$
$$B = \frac{1}{\sqrt{N-t}}\Sigma_{|r\rangle|b_r=0}|r\rangle$$

.

Man kan se at G (Good) og B (Bad) vektorene er ortogonale, ettersom de er en sum av ortogonale vektorerer. I det 2 dimensjonale underrommet av $\mathbb{C}^{2^n}$ utspent av G og B, finner man vektoren u.

$$\begin{aligned} u &= \frac{1}{\sqrt{N}}\Sigma_{|r\rangle}|r\rangle = \frac{\sqrt{t}}{\sqrt{N}}G + \frac{\sqrt{N-t}}{\sqrt{N}}B \\ &= \sin\circ\arcsin(\frac{\sqrt{t}}{\sqrt{N}})G + \cos\circ\arcsin(\frac{\sqrt{t}}{\sqrt{N}})B \\ &= \sin(\theta)G + \cos(\theta)B \end{aligned}$$

Her er $\theta = \arcsin(\frac{\sqrt{t}}{\sqrt{N}})$. Vi ønsker nå å manipulere tilstanden til u i underrommet utspent av G og B for å maksimere $\sin(\theta)$. Dette vil maksimere sannsynligheten for at algoritmen avslutter i en tilstand hvor man har maksimal sjanse for å måle en qubit som er merket. La $\alpha : \mathbb{R}$ være en vinkel og $T = \sin(\alpha)G + \cos(\alpha)B$ være en tilstand. For å se hva orakelet gjør med T kan vi se på hva den gjør med G og B.

$$\begin{aligned} &\mathcal{O}_{(b_k)_N,\pm}(B) = B \\ &\mathcal{O}_{(b_k)_N,\pm}(G) = -G \\ \implies &\mathcal{O}_{(b_k)_N,\pm}(T) = -\sin(\alpha)G + \cos(\alpha)B = \sin(-\alpha)G + \cos(-\alpha)B \end{aligned}$$

Operatoren R har en annen beskrivelse som en refleksjon om en enhetsvektor.

$$R = 2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I$$

Det følger at den andre komponenten i en Grover's iterate er en refleksjon om tilstanden u.

$$\begin{aligned}
&H^{\otimes n} R H^{\otimes n} \\
&= H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I)H^{\otimes n} \\
&= 2H^{\otimes n}|0\rangle^{\otimes n}\langle 0|^{\otimes n}H^{\otimes n} - H^{\otimes n}H^{\otimes n} \\
&= 2uu^* - I
\end{aligned}$$

En Grover's iterate kan derfor også bli betegnet som $G = (2uu^* - I)\mathcal{O}_{(b_k)_N,\pm}$. Dette gjør at vi kan observere hva tilstanden til T er etter å anvende $H^{\otimes n} R H^{\otimes n}$ operatoren, og vi kan se hva Grover's iterate k ganger gjør.

$$\begin{aligned}
&H^{\otimes n} R H^{\otimes n}(T) = \sin(-\alpha + 2\theta)G + \cos(-\alpha + 2\theta)B \\
\implies &G^k(T) = \sin(\alpha + 2k\theta)G + \cos(\alpha + 2k\theta)B
\end{aligned}$$

Hvis man initialiserer $\alpha = \theta$ vil k Grover's iterate gi

$$G^k(T) = \sin((1 + 2k)\theta)G + \cos((1 + 2k)\theta)B.$$

En naiv k for når man skal stoppe Grover's algoritme er den tilstanden som er nærmest G først.

$$\begin{aligned}
&\sin((2k' + 1)\theta) = 1 \\
\implies &(2k' + 1)\theta = \frac{\pi}{2} \\
\implies &k \approx \frac{\pi}{4\theta} - \frac{1}{2} = \lfloor \pi/4 \arcsin(\sqrt{\frac{t}{N}}) \rfloor.
\end{aligned}$$

Denne verdien av k vil gi sannsynligheten for å treffe et merket element

$$p = \sin((1 + 2k)\theta) = \sin((1 + \lfloor \pi/2 \arcsin(\sqrt{\frac{t}{N}}) \rfloor) \arcsin(\sqrt{\frac{t}{N}}))$$

3.1.1 Sannsynlighet forsterkning og Amplitude forsterkning

Sannsynlighetsforsterkningsmetoden opererer på klassen av klassiske Monte Carlo algoritmer. En Monte Carlo algoritme er definert ved at den alltid returnerer et svar etter en endelig forhåndsbestemt tidsbegrensing, men svaret kan være feil. La p være sannsynligheten for at algoritmen returnerer det riktige svaret innen $O(f(n))$. Sannsynlighetsforstningsmetoden virker ved å kjøre algoritmen flere ganger, og dermed øke sjansen for at det riktige svaret har blitt avgitt. Hvis vi kjører algoritmen n ganger så er sjansen for at minst et riktig svar har blitt gitt $1 - (1 - p)^n$. Gitt at $p \ll n$, så er sannsynligheten tilnærmet $1 - (1 - p)^n \approx np$. Ved å kjøre algoritmen $1/p$ ganger vil være en god tilnærming for å maksimere sannsynligheten for at det riktige svaret har blitt avgitt med kjøretid $O(f(n)/p)$.

Amplitudeforsterkningsmetoden, bruker den samme ideen for å forbedre en kvantealgoritme, uten å bruke målinger. Kjøretiden forbedres til $O(f(n)/\sqrt{p})$. Anta at det finnes en kvantealgoritme $A : \mathbb{C}n\text{Bits} \to \mathbb{C}n\text{Bits}$ som finner en tilstand som er merket. Funksjonen $f : n\text{Bits} \to 1\text{Bits}$, bestemmer om et element er merket eller ikke. La p være sannsynligheten for at $\psi = A|n\rangle$ er et merket element, hvor $|n\rangle$ er den beste initielle tilstanden til algoritmen A. Amplitude forsterkning virker ved å definere en ny operator

$$U = (2\psi \cdot \psi^* - I)\mathcal{O}_{f,\pm}.$$

Her er $\mathcal{O}_{f,\pm}$ faseoraklet definert utifra funksjonen f. Ved å anvende operatoren U på ψ $t = \lfloor \pi/4\sqrt{p} \rfloor$ ganger, sender sannsynligheten mot 1 for at utfallet av at algoritmen gir en merket bitstring. Analysen av denne algoritmen er nesten identisk som analysen av Grover's algoritme, for en fullstendig analyse se [2].

3.2 Kvantevandringer basert på kvantemyntkast

Kvantevandringer prøver å ta ideen til Grover's algoritme for søk og å generalisere den til andre datatyper, som grafer. For å gjøre denne generaliseringen deler vi opp problemet inn i traversering og oppdagelse. Det er mange metoder for å traversere over grafer, og her er det noen viktige klasser med grafer som vi vil studere.

For å illustrere hvordan kvantevandring kan virke, starter vi med å se på en klassisk tilfeldig vandring. Se for deg at det er en vandrer som vandrer gjennom en skog med forgreninger. Når vandreren møter på en forgrening kaster de en mynt for å velge hvilken retning de går. En slik vandring vil være et eksempel på en rettet tilfeldig vandring over et binærtre. Denne ideen kan man gjøre om til en kvante vandringsalgoritme ved at man gjør om vandreren til en kvantepartikkel, med en kvantemynt som kan være i superposisjon av 2 forskjellige tilstander. Partikkelen vandrer gjennom skogen avhengig av tilstanden til mynten, akkurat som den klassiske vandreren. Dette tillater partikkelen til å flytte seg gjennom skogen som en superposisjon av forskjellige muligheter. Det vil først være når vi måler partikkelen sin posisjon at vi vil få vite hvor den er, og hvilke utfall mynten har gitt.

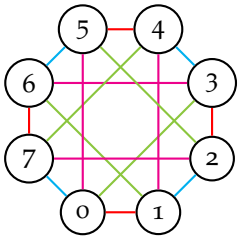
For å være mer presis kan man tilegne to Hilbertrom til en slik kvantevandringsalgoritme. $\mathcal{H}_V$ representerer posisjonene til vandreren, og $\mathcal{H}_C$ representerer utfallene av kvantemyntkastet. Et kvantesteg defineres som komposisjonen av to operatorer $U = S(I \otimes C)$, en myntkast operator $C : \mathcal{H}_C \to \mathcal{H}_C$ som kaster mynten og en forflyttings operator (skift operater) $S : \mathcal{H}_V \otimes \mathcal{H}_C \to \mathcal{H}_V \otimes \mathcal{H}_C$ som leser av myntkastet og forflytter seg henholdsvis. En kvantevandring vil være en algoritme på formen MU^kT, hvor M er en måling, U er et kvantesteg og T er en operator som setter systemet i starttilstanden. Dette er det som vi kaller for en myntbasert kvantevandringsalgoritme og er den formen for vandring som er standarisert i litteraturen.

3.2.1 d-regulære grafer

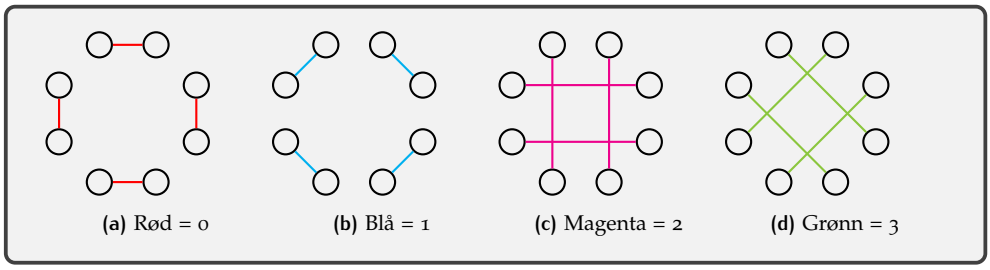
Den første kvantevandringsmetoden som vi skal se på er vandring over d-regulære grafer, den kalles for *position-coin notation*. For at denne metoden skal virke krever vi tillegg at det maksimale kantkromatiske tallet er det samme som d. Hvis man i tillegg ikke tillater at grafen har noen løkker vil spektraldekomposisjonen til algoritmen bli simplere.

La $G = (V, E)$ være en d-regulær graf slik at det maksimale kantkromatiske tallet også er d. Siden kantene i grafen kan fargelegges med d forskjellige farger, så kan vi separere grafen i d forskjellige undergrafer. I hver undergraf er en node koblet til nøyaktig en annen node. La $\mathcal{H}_V = \mathbb{C}V$ være det frie Hilbertrommet over nodene og $\mathcal{H}_C = \mathbb{C}^d$ være myntrommet. Vi definerer forflyttingsoperatoren S på følgende

Figur 7: 4-regulær graf



Fargepartisjon av grafen



(a) Rød = 0 **(b)** Blå = 1 **(c)** Magenta = 2 **(d)** Grønn = 3

måte: La f være en farge og $v : V$ en node. Assosiert med denne noden og fargen finnes det en unik node $v' : V$, slik at det er en kant $(v, v') : E$ som har fargen f.

$$S(v \otimes f) = v' \otimes f$$

Denne operatoren kalles for *flip-flop operatoren*. En egenskap ved denne operaatoren som er enkel å bemerke er at $S^2 = I$, hvilket som gjør at den er hermitisk.

Myntoperatoren C kan velges litt mer vilkårlig, men det er noen mynter som er bedre enn andre. En ønskelig egenskap fra mynten er at den er uniformt fordelt. Det kan finnes tilfeller hvor det er interessant å se på mynter som er vektet slik at en farge er vektet mer enn andre. Forskjellige normale valg av myntoperatorer kommer vi tilbake til senere.

Et kvantesteg langs denne grafen kan man nå definere som $U = S(I \otimes C)$. Vi bemerker oss at egenskapen som lar oss bruke $I \otimes C$ er at grafen er d-regulær. Hvis grafen ikke hadde hatt denne egenskapen, men heller at den maksimale graden var lik det maksimale kantkromatiske tallet kan man fremdeles bruke det samme prinsippet. Siden vi ikke lengre kan være sikre på at alle noder har d tilstøtende farger, så må man ha en mynt for hver node som tilordner ny farge langs den noden.

Vi illustrer denne metoden med et eksempel: Grafen som vi skal vandre over er illustrert i figur 7. La $G = (V, E)$ være grafen. Siden vi har 8 noder velges $\mathcal{H}_V = \mathbb{C}V = \mathbb{C}^8 = (\mathbb{C}^2)^{\otimes 3}$, og 4-regulariteten sier at $\mathcal{H}_C = \mathbb{C}\mathrm{Farger} = \mathbb{C}^4 = (\mathbb{C}^2)^{\otimes 2}$.

$$\mathcal{H}_V \otimes \mathcal{H}_C = \mathbb{C}^8 \otimes \mathbb{C}^4 = \mathbb{C}^{32}$$

Ettersom at det er bijeksjoner $V \simeq 3\mathrm{Bits}$ og $\mathrm{Farger} \simeq 2\mathrm{Bits}$ kan vi identifisere $|n\rangle$ med enten den n-te noden eller n-te fargen. Den n-te noden og n-te fargen er definert i figur 7.

Flip-flop operatoren er definert utifra fargepartisjonen gitt i figur 7. For å illustrere dette med et eksempel velger vi en tilstand $\psi = |0\rangle$ og fargen rød som tilsvarer $|0\rangle$. Da har vi at $S(\psi \otimes |0\rangle) = |1\rangle \otimes |0\rangle$. Her byttes ut ψ med naboen til node 0 i den røde

undergrafen, som er node 1. På samme virker det at for en vilkårlig tilstand ψ og en farge, så finner vi naboen til tilstanden ψ i undergrafen med tilsvarende farge.

Som nevnt kan myntoperatoren velges mer vilkårlig. I vårt tilfelle kan vi velge en mynt som kalles for Hadamardmynten. Denne mynten er generelt definert som $H^{\otimes n} : \mathbb{C}^{2^{\otimes n}} \to \mathbb{C}^{2^{\otimes n}}$, og vi bruker varianten hvor $n = 2$. Matrisen for denne mynten ser ut som følger:

$$H^{\otimes 2} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

Denne er laget slik at når vi andvender den på en farge vil den sette fargen i en ny tilstand som er en likevektet superposisjon av alle de andre fargene. Her kan fasen være forskjellig, men det endrer ikke utfallet av et steg. Som et eksempel vil $H^{\otimes 2}$ anvendet på fargen rød være:

$$H^{\otimes 2} |0\rangle = 1/2 \Sigma_{n=0}^{3} |n\rangle .$$

Nå som vi har komponentene til kvantestegsoperatoren kan vi definere den som:

$$U = S(I \otimes H^{\otimes 2}).$$

Vi kan definere en starttilstand $\psi = |0\rangle \otimes |0\rangle$ til å være lokalisert i node 0 med fargen rød. Et kvantesteg vil dermed plassere oss i følgende tilstand:

$$\begin{aligned} U(\psi) &= S(I \otimes H^{\otimes 2})(\psi) = S(|0\rangle \otimes H^{\otimes 2}(|0\rangle)) = S(|0\rangle \otimes 1/2 \Sigma_{n=0}^{3} |n\rangle) \\ &= 1/2 \Sigma_{n=0}^{3} S(|0\rangle \otimes |n\rangle) = 1/2(|1\rangle \otimes |0\rangle + |7\rangle \otimes |1\rangle + |5\rangle \otimes |2\rangle + |3\rangle \otimes |3\rangle). \end{aligned}$$

Her ser vi at tilstanden etter et kvantesteg faktisk er i en likevektet superposisjon av alle naboene til starttilstanden. I dette tilfellet har vi at posisjonen er sammenfiltret med fargen. Når vi måler hvor partikkelen er, så vet vi også hvilken farge mynten har. Denne informasjonen har vi ettersom vi vet hvor vi startet fra, om dette var uvisst ville vi ikke nødvendigivis ha en slik sammenfiltring. Det neste kvantesteget kan vi observere til å være:

$$\begin{aligned} U^2(|0\rangle |0\rangle) = \frac{1}{4}(&|0\rangle \otimes (|0\rangle - |1\rangle - |2\rangle + |3\rangle) \\ &|2\rangle \otimes (|0\rangle + |1\rangle + |2\rangle - |3\rangle) \\ &|4\rangle \otimes (|0\rangle - |1\rangle + |2\rangle - |3\rangle) \\ &|6\rangle \otimes (|0\rangle + |1\rangle - |2\rangle + |3\rangle)). \end{aligned}$$

Her ser vi at fargene ikke er sammenfiltret med posisjonen, og vi kan ikke beslutte hvor vi kom fra, dersom vi måler en posisjon. Vi kan heller ikke beslutte hvor vi er dersom vi måler en farge. En egenskap ved denne vandringen som vi kan observere er at tilstanden til posisjonen vil enten være på en partalls node eller en oddetalls node. Hvis vi velger den initelle tilstanden til å være

$$\psi = H^{\otimes 3} |0\rangle = 1/\sqrt{8} \Sigma_{n=0}^{7} |n\rangle ,$$

så kan man se at sannsynlighetsfordelingen til posisjonen ikke endrer seg etter et kvantesteg.

3.2.2 Generelle grafer

En myntet metode for kvantevandring som virker på en generell klasse av simple grafer er den som kalles for *arc notation*. Denne metoden er ofte beskrevet som at

man vandrer over kantene, istedenfor at man vandrer over nodene. Dette gjør at antallet qubits krevet for å beskrive grafen kan øke dramatisk.

La $G = (V, E)$ være en graf. Det finnes en funksjon $noder : E \to \mathcal{P}(V)$, som sender hver kant til mengden av noder den er koblet til. Dersom $e : E$ binder sammen u og v i V, så er $noder(e) = \{u, v\}$. Dette gjør at hvert element $e : E$ kan representeres på formen $(u, v) : VxV$, her er u start noden og v er slutt noden. Merk at to kanter (u, v) og (v, u) identifiseres, og denne identifikasjonen gir opphavet til en ekvivalens relasjon $\sim$. Vi kan dermed finne en isomorfi fra E til et underrom av kvotienten til $V \times V$ under denne ekvivalens relasjonen. Mengden E kan vi beskrive som:

$$E \simeq \{(u, v) : V \times V \mid \exists e : E \text{ s.a. } noder(e) = \{u, v\}\}/\sim .$$

For å beskrive vandringen retter vi E på en slik måte at (u, v) og (v, u) ikke lenger blir identifisert. Vi betrakter følgende mengde:

$$\vec{E} = \{(u, v) : V \times V \mid \exists e : E \text{ s.a. } noder(e) = \{u, v\}\}.$$

Det totale rommet som vi skal kvantevandre over defineres til å være:

$$\mathcal{H} = \mathbb{C}\vec{E} \subseteq \mathbb{C}(V \times V) = \mathbb{C}V \otimes \mathbb{C}V = \mathcal{H}_V \otimes \mathcal{H}_C.$$

A priori vet vi ikke om vi kan bruke færre qubits enn $lg_2(dim(\mathbb{C}(V^2)))$. Posisjonsrommet defineres dermed til å være det første argumentet $\mathcal{H}_V = \mathbb{C}V$ og myntrommet er det andre argumentet $\mathcal{H}_C = \mathbb{C}V$. Merk at grafen er representert ved $\mathcal{H}$, men rommet $\mathcal{H}_V \otimes \mathcal{H}_C$ tillater tilstander som ikke representerer en kant i grafen.

Skift operatoren $S : \mathcal{H} \to \mathcal{H}$ definerer vi på hver elementær tensor. Vi definerer $S = S'|_{\mathcal{H}}$, hvor S' er definert under. Denne definisjonen av S gjør at den blir en *flip-flop operator*.

$$\begin{aligned} S' : \mathbb{C}V \otimes \mathbb{C}V &\to \mathbb{C}V \otimes \mathbb{C}V \\ u \otimes v &\mapsto v \otimes u \end{aligned}$$

For å definere myntoperatoren trenger vi å dekomponere rommet $\mathcal{H}$.

$$\mathcal{H} = \mathbb{C}\vec{E} = \bigoplus_{v:V} \mathbb{C}\{u : V \mid \exists e : E \; noder(e) = \{v, u\}\}.$$

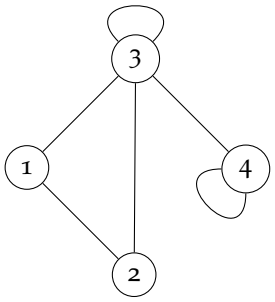
Vi kan nå betrakte rommet $\mathbb{C}\{u : V \mid \exists e : E \; noder(e) = \{v, u\}\}$ som et lokalt myntrom. Dermed for enhver $v : V$ definerer vi en lokal myntoperator $C_v : \mathbb{C}\{u : V \mid \exists e : E \; noder(e) = \{v, u\}\} \to \mathbb{C}\{u : V \mid \exists e : E \; noder(e) = \{v, u\}\}$. Den globale myntoperatoren defineres som den blokkdiagonale operatoren:

$$\begin{aligned} C &: \mathcal{H} \to \mathcal{H} \\ C &= \bigoplus_{v:V} C_v. \end{aligned}$$

Et kvantesteg med *arc notation* algoritmen ser ut som $U = SC = S \bigoplus_{v:V} C_v$. Vi illustrerer hvordan dette virker med et eksempel. La figur 9 være grafen $G = (V, E)$. A priori kan vi se at $\mathcal{H} \subseteq \mathbb{C}(V^2) = \mathbb{C}^{16}$, vi trenger dermed 4 qubits for å representere grafen. For å regne ut hva $\mathcal{H} = \mathbb{C}\vec{E}$ er, trenger vi å forstå basisen $\vec{E}$.

$$\vec{E} = \{(1,2), (1,3), (2,1), (2,3), (3,1), (3,2), (3,3), (3,4), (4,3), (4,4)\} \implies \mathcal{H} = \mathbb{C}^{10}$$

Figur 9: Simpel graf med løkker



For å definere myntoperatorene spalter vi opp basisen på følgende måte:

$$\vec{E} = (\{1\} \times \{2,3\}) \sqcup (\{2\} \times \{1,3\}) \sqcup (\{3\} \times \{1,2,3,4\}) \sqcup (\{4\} \times \{3,4\})$$
$$\implies \mathcal{H} = (\mathbb{C} \otimes \mathbb{C}^2) \oplus (\mathbb{C} \otimes \mathbb{C}^2) \oplus (\mathbb{C} \otimes \mathbb{C}^4) \oplus (\mathbb{C} \otimes \mathbb{C}^2) \subseteq \mathbb{C}^4 \otimes \mathbb{C}^4$$

Vi kan dermed definere myntene våre til å være Hadamardmynter $C_\nu = H^{\otimes n}$ hvor $n : \{1,2\}$ ($n = 2$ hvis og bare hvis $\nu = 3$). Vi definerer C som den blokkdiagonale mynten opp til isomorfi (basis):

$$C = \bigoplus_{\nu:V} C_\nu \simeq H \oplus H \oplus H^{\otimes 2} \oplus H.$$

Et kvantesteg er gitt ved operatoren $U = SC \simeq S(H \oplus H \oplus H^{\otimes 2} \oplus H)$ opp til isomorfi. Vi illustrer hvordan man bruker denne operatoren med et eksempel. Anta at vi har en tilstand som er lokalisert i løkken på node 3: $\psi = |3\rangle\,|3\rangle$.

$$\begin{aligned} U(\psi) = SC(|3\rangle \otimes |3\rangle) &\simeq S(H \oplus H \oplus H^{\otimes 2} \oplus H)(|3\rangle \otimes |3\rangle) \simeq S(|3\rangle \otimes H^{\otimes 2}(|3\rangle)) \\ &= S(|3\rangle \otimes (1/2 \Sigma_{n=1}^4 |n\rangle)) = 1/2 \Sigma_{n=1}^4 S(|3\rangle \otimes |n\rangle) = 1/2 \Sigma_{n=1}^4 |n\rangle \otimes |3\rangle \end{aligned}$$

3.2.3 *Kvantemynter*

I litteraturen ([4] og [2]) er det tre kvantemynter som ofte er av stor interesse. Disse tre myntene blir brukt på grunn av at de er enkle å beskrive, og har anvendelser andre steder enn som en kvantemynt. Den første mynten har vi allerede snakket om, nemlig Hadamardmynten. Hadamardmynten kan brukes når dimensjonen til myntrommet er en 2-er potens $\dim(\mathcal{H}_C) = 2^n$. Da beskriver vi Hadamardmynten som:

$$H_C = H^{\otimes n}.$$

Fouriermynten bruker QFT (Quantum Fourier Transform, se [3]) som myntoperatoren. La $\omega_n = e^{2\pi i/n}$ være den n-te roten av enhet, QFT operatoren F_n er definert som følgende:

$$F_n : \mathbb{C}^n \to \mathbb{C}^n$$
$$F_n = \frac{1}{\sqrt{n}} (\omega_n^{ij})_{(i,j):\{1,\ldots,n\}\times\{1,\ldots,n\}}$$

Denne mynten er mer anvendbar enn Hadamardmynten, ettersom dimensjonen til $\mathcal{H}_C$ kan være arbitrær. For å illustrere mer hvordan F_n ser ut så regner vi ut F_4.

$$F_4 = \frac{1}{\sqrt{4}} \begin{pmatrix} \omega_4^{0\cdot0} & \omega_4^{0\cdot1} & \omega_4^{0\cdot2} & \omega_4^{0\cdot3} \\ \omega_4^{1\cdot0} & \omega_4^{1\cdot1} & \omega_4^{1\cdot2} & \omega_4^{1\cdot3} \\ \omega_4^{2\cdot0} & \omega_4^{2\cdot1} & \omega_4^{2\cdot2} & \omega_4^{2\cdot3} \\ \omega_4^{3\cdot0} & \omega_4^{3\cdot1} & \omega_4^{3\cdot2} & \omega_4^{3\cdot3} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

Den siste mynten som nevnes kalles for Grovers mynt. Denne mynten har sitt navn fra Grover's algoritme, ettersom det er operatoren som reflekterer om diagonal tilstanden. La $\mathcal{H}_C = \mathbb{C}^n$ og $u = {}^1/\sqrt{n}\Sigma_{i=0}^{n-1} |i\rangle$, da definerer vi Grover's mynt til å være:

$$G : \mathbb{C}^n \to \mathbb{C}^n$$
$$G = 2uu^* - I$$

For å illustrere hvordan mynten ser ut så regner vi ut G når $n = 4$.

$$G = 2uu^* - I = \frac{2}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}$$

3.3 Umyntede kvantevandringer

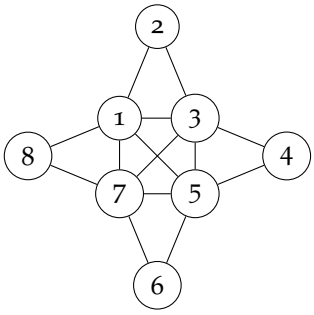
Myntede kvantevandringsmetoder hermer etter klassiske tilfeldige vandringer ved at man tar et valg og følger denne. Valget man tar blir oversatt til et kvantemyntkast, hvilket som gjør at vi kan velge en superposisjon av valg. I motsetning til det klassiske valget, er valget fra kvantmyntkastet forhåndsbestemt ved operatoren som beskriver den. Dette gjør at selve myntkastet er en abstraksjon man ikke nødvendigivis trenger. Det finnes derfor flere metoder som utnytter andre strukturer hos grafer for å utføre kvantevandringer. Blant disse metodene har vi Portugal sin forskyvede metode (Staggered model, [5], [2]) og Szegedy sin metode ([3], [2]).

3.3.1 Staggered model

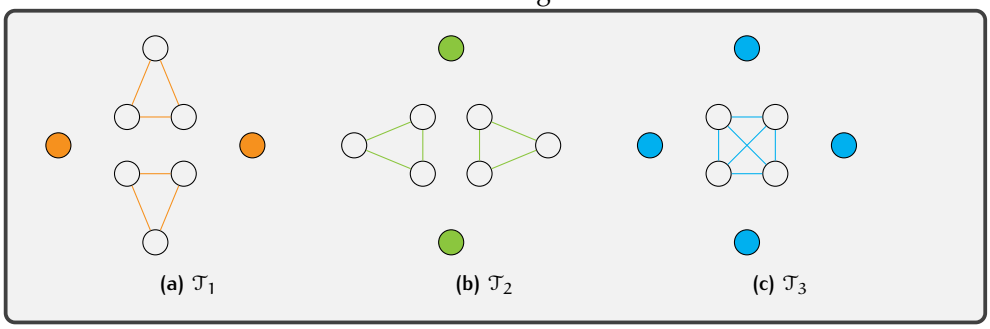
Den første umyntede metoden som vi skal se på er den forskyvede metoden. La $G = (V, E)$ være en graf. Vandringen baserer seg på å finne graf tesselleringer $\mathcal{T}_i$ og en assosiert graf tesselleringsdekke $\{\mathcal{T}_i\}$ til G. En graftessellering er en clique partisjon av nodene til G, og et element av $\mathcal{T}_i$ kalles for et polygon. $\mathcal{E}(\mathcal{T}_i)$ defineres som mengden av kanter tilhørende til polygonene i $\mathcal{T}_i$. En graf tesselleringsdekke er en samling av graf tesselleringer slik at alle kantene i grafen er dekket. Dette vil si at unionen av alle kantene i tesseleringene blir mengden av kanter.

$$\bigcup \mathcal{E}(\mathcal{T}_i) = E$$

Figur 10: Shuriken graf



Graf tesselleringsdekke



Vi illustrerer graf tesselleringsdekke med et eksempel. La $G = (V, E)$ være shuriken grafen som beskrevet i figur 10. Vi kan dermed finne 3 graf tesseleringer $\mathcal{T}_1$, $\mathcal{T}_2$ og $\mathcal{T}_3$.

$$\mathcal{T}_1 = \{\{1,2,3\},\{4\},\{5,6,7\},\{8\}\}$$
$$\mathcal{T}_2 = \{\{1,7,8\},\{2\},\{3,4,5\},\{6\}\}$$
$$\mathcal{T}_3 = \{\{1,3,5,7\},\{2\},\{4\},\{6\},\{8\}\}$$

Her kan vi f.eks. se at mengden $\{1,3,5,7\}$ er et polygon i $\mathcal{T}_3$. Mengden $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$ er en graf tesselleringsdekke, ettersom enhver kant i G opptrer i et polygon fra $\mathcal{T}_i$ for en eller annen i. Her sier vi at tildekningsnummeret til graf tesselleringsdekket $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$ er 3. Merk at graf tesselleringsdekker ikke er unike.

Valget av en graf tesselleringsdekke vil bestemme hvordan kvantestegsoperatoren vil se ut. Hilbert rommet som kvantevandringen skal foregå i er $\mathcal{H} = \mathbb{C}V$. For enhver graf tessellering $\mathcal{T}$ assosierer vi en operator $H_{\mathcal{T}} : \mathcal{H} \to \mathcal{H}$, og ethvert polygon $\alpha : \mathcal{T}$ assosierer vi med en vektor $\vec{\alpha} = {}^{1}/_{\sqrt{|\alpha|}} \Sigma_{v:\alpha} |v\rangle$ i $\mathcal{H}$. Vi kan nå definere operatoren $H_{\mathcal{T}}$ og kvantestegs operatoren U som under.

$$H_{\mathcal{T}} = 2\Sigma_{\alpha:\mathcal{T}} \alpha\alpha^* - I$$
$$U = \circ_{\mathcal{T}:\mathrm{Cover}} H_{\mathcal{T}}$$

Vi illustrer videre hvordan denne operatoren kan se ut med figur 10. Siden grafen G har 8 noder følger det at $\mathcal{H} = \mathbb{C}^8$. For å finne kvantestegsoperatoren U trenger vi å regne ut alle vektorene α. La $\alpha_1 : \mathcal{T}_1$ være vektoren definert av polygonet $\{1,2,3\}$,

$\alpha_2 : \mathcal{T}_2$ være vektoren definert av polygonet $\{4\}$, osv. Vi velger at β_i tilhører $\mathcal{T}_2$ og at γ_i tilhører $\mathcal{T}_3$.

$$
\begin{aligned}
\alpha_1 &= 1/\sqrt{3}(|1\rangle + |2\rangle + |3\rangle) \\
\alpha_2 &= |4\rangle \\
\alpha_3 &= 1/\sqrt{3}(|5\rangle + |6\rangle + |7\rangle) \\
\alpha_4 &= |8\rangle
\end{aligned}
$$

$$
\begin{aligned}
\beta_1 &= 1/\sqrt{3}(|1\rangle + |7\rangle + |8\rangle) \\
\beta_2 &= |2\rangle \\
\beta_3 &= 1/\sqrt{3}(|3\rangle + |4\rangle + |5\rangle) \\
\beta_4 &= |6\rangle
\end{aligned}
$$

$$
\begin{aligned}
\gamma_1 &= 1/2(|1\rangle + |3\rangle + |5\rangle + |7\rangle) \\
\gamma_2 &= |2\rangle \\
\gamma_3 &= |4\rangle \\
\gamma_4 &= |6\rangle \\
\gamma_5 &= |8\rangle
\end{aligned}
$$

De hermitiske operatorene $H_{\mathcal{T}_i}$ kan sees til å være.

$$
H_{\mathcal{T}_1} = 2\Sigma_{i=1}^{4} \alpha_i \alpha_i^* - I = \begin{pmatrix}
-1/3 & 2/3 & 2/3 & 0 & 0 & 0 & 0 & 0 \\
2/3 & -1/3 & 2/3 & 0 & 0 & 0 & 0 & 0 \\
2/3 & 2/3 & -1/3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1/3 & 2/3 & 2/3 & 0 \\
0 & 0 & 0 & 0 & 2/3 & -1/3 & 2/3 & 0 \\
0 & 0 & 0 & 0 & 2/3 & 2/3 & -1/3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

$$
H_{\mathcal{T}_2} = 2\Sigma_{i=1}^{4} \beta_i \beta_i^* - I = \begin{pmatrix}
-1/3 & 0 & 0 & 0 & 0 & 0 & 2/3 & 2/3 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1/3 & 2/3 & 2/3 & 0 & 0 & 0 \\
0 & 0 & 2/3 & -1/3 & 2/3 & 0 & 0 & 0 \\
0 & 0 & 2/3 & 2/3 & -1/3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
2/3 & 0 & 0 & 0 & 0 & 0 & -1/3 & 2/3 \\
2/3 & 0 & 0 & 0 & 0 & 0 & 2/3 & -1/3
\end{pmatrix}
$$

$$
H_{\mathcal{T}_3} = 2\Sigma_{i=1}^{5} \gamma_i \gamma_i^* - I = \begin{pmatrix}
-1/2 & 0 & 1/2 & 0 & 1/2 & 0 & 1/2 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/2 & 0 & -1/2 & 0 & 1/2 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & -1/2 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & 1/2 & 0 & -1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Veldig simple beregninger vil vise at $U = H_{\mathcal{T}_3} H_{\mathcal{T}_2} H_{\mathcal{T}_1}$ er:

$$U = \begin{pmatrix} -1/18 & 4/9 & 5/18 & 2/3 & -7/18 & -2/9 & 5/18 & 0 \\ 2/3 & -1/3 & 2/3 & 0 & 0 & 0 & 0 & 0 \\ 5/18 & 4/9 & -1/18 & 0 & 5/18 & -2/9 & -7/18 & 2/3 \\ 4/9 & 4/9 & -2/9 & -1/3 & -2/9 & 4/9 & 4/9 & 0 \\ -7/18 & -2/9 & 5/18 & 0 & -1/18 & 4/9 & 5/18 & 2/3 \\ 0 & 0 & 0 & 0 & 2/3 & -1/3 & 2/3 & 0 \\ 5/18 & -2/9 & -7/18 & 2/3 & 5/18 & 4/9 & -1/18 & 0 \\ -2/9 & 4/9 & 4/9 & 0 & 4/9 & 4/9 & -2/9 & -1/3 \end{pmatrix}.$$

Merk at rekkefølgen vi komponerer operatorene våre gir nye kvantevandringer over grafen. Ved å anvende en permutering på graf tesselleringsdekket ender vi opp med en ny graf tesselleringsdekke, dette gjør at alle permutasjoner gir opphav til en ny gyldig kvantevandring. Vi kan i tillegg vekte hvor mye hver hermitiske operator skal bidra til vandringen. Gitt at vi har en permutasjon σ på graf tesselleringsdekket kan vi definere U som under.

$$U = \Pi_{n=1}^{k} e^{i\theta H_{\mathcal{T}_{\sigma(n)}}} = \Pi_{n=1}^{k} (\cos(\theta) I + i \sin(\theta) H_{\mathcal{T}_{\sigma(n)}})$$

En forskyvet metode basert på en tesselleringsdekke med størrelse k kalles for en k-tessellerbar kvantevandring.

3.3.2 Grensepunktet og quasi-periodisitet

Hva skjer med en kvantevandring ved uendelige steg? Hvis vi antar at operatoren U er en kvantevandring og vi har starttilstand ψ, hvordan ser $\varprojlim_{t\to\infty} U^t\psi$ ut? Det som vi først må vite er om denne grensen i det hele tatt konverger. Ettersom operatoren U kreves til å være unitær, følger det at $\|U^{t+1}\psi - U^t\psi\|$ er konstant for alle t. Denne grensen vil kun eksistere om ψ er et fikspunkt av U til å starte med.

Et annet interessant spørsmål som vi kan spørre oss er om kvantevandringen tilbakestiller seg selv. Med andre ord, gitt U og ψ som over, finnes det en t slik at $U^t\psi = \psi$? Vi spør om U har periode t, og generelt sett finner vi ikke unitære operatorer med en slik egenskap, men vi kan i det minste at de er quasi-periodiske. Quasi-periodisk i denne konteksten vil si at de er periodisk opp til en epsilon ball med feil. For å motivere dette vil vi se på analysen til en 2 dimensjonal unitær operator.

La U være 2 dimensjonal og diagonaliserbar, da er den similær til en matrise på formen

$$U = \begin{pmatrix} e^{2\pi i\lambda_1} & 0 \\ 0 & e^{2\pi i\lambda_2} \end{pmatrix}.$$

Vi kan arbitrært tilnærme λ_j med rasjonalle tall a_j/b_j. Vi kan da skrive opp følgende

$$U^{b_1 b_2} \approx \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Dette er ikke et bevis på at alle kvantecandringer er quasi-periodiske. For å fullføre det må vi først definere hva quasi-periodisk er litt mer rigorøst og komme med et induksjonbevis for å dekke alle de endelige dimensjonene. Det følger av dette at vi ikke kan garantere eller burde forvente at kvantevandringer på uendelige grafer er quasi-periodiske, og at dette er kun en egenskap vi finner hos endelige grafer.

3.4 Kvantesøk

En anvendelse av kvantevandringer er klassen av algoritmer kalt *Quantum spatial search* (QSS). Disse algoritmene har blitt detaljert beskrevet av Portugal [2] og Tulsi [6]. QSS algoritmene bruker en vandrer for å finne en merket node i en database eller graf.

Vi vil beskrive en QSS algoritme som søker gjennom en graf hvor nøyaktig en node er merket. Anta at $G = (V, E)$ er en graf og at det finnes en kvantestegsoperator $U : \mathcal{H} \to \mathcal{H}$. La $\psi : \mathcal{H}$ være den merkede noden. Siden vi antar at grafen G er endelig følger det at $\mathcal{H}$ er endelig dimensjonal og det finnes en ortonormal basis $\{\psi_1, \psi_2, ..., \psi_n\}$, hvor $\psi = \psi_1$. Vi definerer deretter operatoren R på følgende vis:

$$\begin{aligned} R : \mathcal{H} &\to \mathcal{H} \\ \psi_1 &\mapsto -\psi_1 \\ \psi_k &\mapsto \psi_k \end{aligned}$$

En kvantevandringsbasert QSS operator $S = UR$, denne kaller vi også for den modifiserte kvantestegsoperatoren. En generell QSS operator $S = DR$ trenger å tilfredstille at den initielle tilstanden s overlapper med den merkede noden ψ_1, aka $\langle s, \psi_1 \rangle \neq 0$. Vi krever også at s er en egenvektor til D. I dette rammeverket kan vi se at $D = 2uu* - I$, gir Grover's algoritme. Faktisk er Grover's algoritme den optimale QSS algoritmen [7]. Ettersom den kvantevandringsbaserte QSS algoritmen har noe enklere form velger vi å fokusere på den.

Et kvantesøk over grafen G vil være en operator på formen $\mathcal{S} = MS^tP$, her er M en måling av tilstanden i basisen til nodene, t er antallet ganger det modifiserte kvantesteget skal kjøre, og P er en operator som sender $|0\rangle$ til den initielle tilstanden. La $s = P(|0\rangle)$ være den initielle tilstanden, og $p(t)$ sjansen for å måle ψ_1. For å finne ut hvilken t vi burde bruke ønsker vi å maksimere $p(t)$.

$$p(t) = |\langle \psi_1 | S^t | s \rangle|^2$$

For alle praktiske anvendelser er det best å tilnærme en t ved å bruke numeriske metoder for å maksimere t. Gitt noen flere betingelser kan man bruke resultatene til Tulsi og Portugal for å estimere en t. For dette er det tilstrekkelig å vite spektraldekomposisjonen til operatoren U. La egenvektorene til U være $\vec{\phi_k}$, med egenverdiene $e^{i\phi_k}$. Med disse vektorene kan vi regne ut tre konstanter:

$$\begin{aligned} A &= 2\Sigma_{\phi_k = 0} |\langle \psi_1 | \vec{\phi_k} \rangle|^2 \\ B &= \Sigma_{\phi_k \neq 0} \frac{|\langle \psi_1 | \vec{\phi_k} \rangle|^2 \sin(\phi_k)}{1 - \cos(\phi_k)} \\ C &= \Sigma_{\phi_k \neq 0} \frac{|\langle \psi_1 | \vec{\phi_k} \rangle|^2}{1 - \cos(\phi_k)} \end{aligned}$$

Gitt at $B = 0$, kan man velge $\lambda = \sqrt{A}/\sqrt{B}$. Da kan man enkelt finne en optimal t. Hvis U er en matrise som kun har reelle innlegg, så følger det at $B = 0$.

$$\begin{aligned} t_{opt} &= \lfloor \pi/2\lambda \rfloor \\ p(t_{opt}) &= \frac{|\langle \psi_1 | s \rangle|^2}{AC} \sin^2(\lambda t_{opt}) \end{aligned}$$

4 Q# OG IMPLEMENTASJON AV KVANTEVANDRINGER

I dag finnes det flere alternativer for å programmere kvantealgoritmer med høynivå språk. I Python skrives det flere moduler som IBM sitt *qiskit* og *openql*. I Haskell finnes det flere domene spesifikke språk som *quipper* og *quantum IO monad*. Microsoft har utviklet *Q#* som er et dotnet basert språk, inspirert av *quantum IO monad*, men har semantikk fra C# og F#. Noen av disse språkene som *qiskit* og *Q#* er skrevet slik at man kan bruke bedriftene sine Kvanteberegnings løsninger.

I dette prosjektet har jeg valgt å bruke Q# for å eksperimentere med kvantealgoritmene. Figur 7, 9 og 10 får alle implementert en kvantevandringsalgoritme. Vi ser litt hvordan algoritmene ser ut i et annet perspektiv, sammen med noe eksperimentell data. Implementasjonene av algoritmene tar i tillegg å fremhever noen interessante problemstillinger ved slike implementasjoner.

4.1 Q# intro

Q# er et sterkt typet språk, dette vil si at man ikke kan gjøre implisitt casting mellom typer, og alle funksjoner skal ha en entydig type signatur. De typene som er primitive for språket er String, Int, Double, =>, Qubit, Result, Unit og Array. String, Int, Double og => er kjent fra andre programmeringsspråk, nemlig tekst, heltall, flyttall og funksjoner. Typene som skiller Q# fra andre språk er Qubit, Result og Unit. En term av typen Qubit representerer en qubit i et kvantesystem. På disse kan vi utføre operasjoner som har typen Qubit => Unit. Unit typen fanger effekter som utføres på qubits eller filer. Typen Unit kan ligne på Void typen i C, men den er analog til IO monaden i Haskell. Typen Result har termene Zero og One, og denne representerer en måling av et system. Den siste typen Array er en funktor av typer. Dette vil si at det er en abstrakt type som opererer på andre typer. F.eks. kan vi lage Int[] og Qubit[], som er lister av heltall og lister av qubits.

En forklaring av Q# er vedlagt i appendix A som instruksjonskode. Denne koden forklarer hvordan man konstruerer funksjoner, deklarerer variabler og bruker kontroll uttrykk som if-else.

All koden som er skrevet til dette prosjektet ligger opplastet på `https://github.com/celestialcry/quantum_walks`. Den relevante koden ligger i Program.qs. Bemerk at koden er kjørt fra Python via filen host.py.

4.1.1 Grover's algoritme og Amplitude forsterkning

Vi vil starte med å beskrive Grover's algoritme i Q#. For dette trenger vi å kunne konstruere faseorakler. Dette gjør vi med teknikken som er beskrevet med figur 4. Akkurat som i figuren konjugerer vi en hjelpequbit med HX, og anvender oraklet.

Beginning of code

```
1          // Oracle Conversion
2          operation MarkingAsPhase(register : Qubit[],
3                                   oracle : (Qubit[], Qubit) => Unit is Adj)
```

```
                                : Unit is Adj {
        use target = Qubit();
        within {
            X(target);
            H(target);
        }
        apply {
            oracle(register, target);
        }
    }

    function MarkingToPhase(oracle : (Qubit[], Qubit) => Unit is Adj)
                                : (Qubit[] => Unit is Adj) {
        return MarkingAsPhase(_, oracle);
    }
```

End of code

R operatoren er definert som $2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I$. Siden å modifisere den globale fasen ikke endrer utfallet endrer vi operatoren R til $I - 2|0\rangle^{\otimes n}\langle 0|^{\otimes n}$ for simpelhetens skyld.

Beginning of code

```
    // Zero Oracle
    operation MarkingZero(register : Qubit[], target : Qubit) : Unit
    is Adj + Ctl {
        within {
            ApplyToEachCA(X, register);
        }
        apply {
            Controlled X(register, target);
        }
    }

    operation R(register : Qubit[]) : Unit is Adj {
        MarkingToPhase(MarkingZero)(register);
    }
```

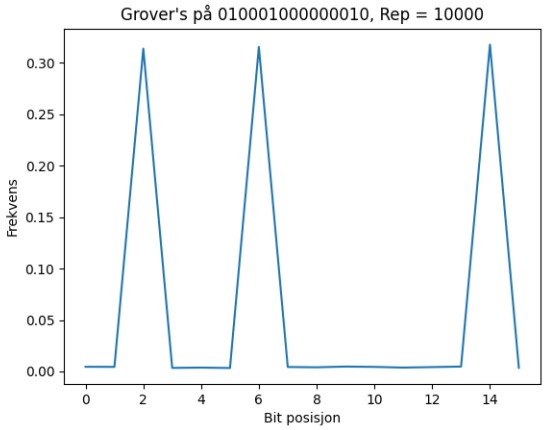
End of code

Nå har vi alle byggeklossene til å definere en Grover's iterate og dermed bygge Grover's algoritme.

Beginning of code

```
    // One Grover's iterate
    operation GroversIterate(register : Qubit[],
                                phaseOracle : Qubit[] => Unit is Adj)
                                : Unit is Adj {
        phaseOracle(register);
        within {
            ApplyToEachCA(H,register);
        }
        apply {
            MarkingToPhase(MarkingZero)(register);
        }
    }
```

Figur 12: Grover's algoritme kjørt på 0100010000000010




```
13
14          // Grover's algorithm
15          operation GroversAlgorithm(register : Qubit[],
16                                     phaseOracle : Qubit[] => Unit is Adj,
17                                     time : Int)
18                                     : Unit {
19              // Initiate initial state
20              ApplyToEachCA(H,register);
21
22              // Running Grover's algorithm
23              for i in 1..time {
24                  GroversIterate(register, phaseOracle);
25              }
26          }
```

End of code

Denne funksjonen tester vi på bitstringen 0100010000000010. Målet er dermed å oppdage at den 2e, 6e og 14e bitten er en 1er. Vi kan regne ut antallet iterasjoner vi skal bruke er $k = 1$. Grover's algoritme presterer veldig bra, som man kan se i figur 12

Vi kan generalisere Grover's algoritme til amplitudeforsterkningsteknikken ved å ha alle operatorene som input. Den implementeres helt analogt som Grover's algoritme.

Beginning of code

```
1           // Amplitude amplification technique
2           operation AmplitudeAmplification(register : Qubit[],
3                                            outputOperator : (Qubit[] => Unit is Adj),
4                                            phaseOracle : (Qubit[] => Unit is Adj),
5                                            time : Int)
6                                            : Unit is Adj {
7               for i in 1..time {
8                   outputOperator(register);
9                   phaseOracle(register);
10              }
11            }
```

End of code

4.2 Kvantevandringer

Vi vil se på noen simuleringer av de kvantevandringene som vi har laget tidligere i rapporten. Den initielle tilstanden til kvantevandringen påvirker utfallet sterkt, og dette er en av effektene vi vil se på. I position-coin notation vil vi også se på effekten av å bruke forskjellige mynter. Til slutt skal vi gjøre et kvantesøk eksperiment. Merk at de grafene vi ser på er endelige og har nødvendigvis ikke en symmetri som gjør vandringene enklere å studere.

Diskrete kvantevandringer på ubegrensede grafer som linjen og gitteret i 2 dimensjoner har blitt omfattende studert i andre verk (se [2] og [4]). Under rimelige antagelser kan man finne analytiske løsninger av posisjonen til kvantepartikkelen. Dette er gjort på den ubegrensede linjen, det ubegrensede 2 dimensjonale gitteret, sirkelen, det begrensede 2 dimensjonale gitteret og hyperkuben.

Videre legges det ikke til spesifikke kodesnutter, men mer abstrakt kode. For å finne konkretiseringen av disse kodesnuttene så finner man resten av koden i github repoet nevnt over.

4.2.1 Position-coin notation

Vi starter med å se på position-coin notation. Denne kvantevandringen foregår over Hilbertrommet $\mathcal{H}_V \otimes \mathcal{H}_C$, disse rommene er representert som *position* og *color*. Den abstrakte koden for denne kvantevandringen kan man se under.

Beginning of code

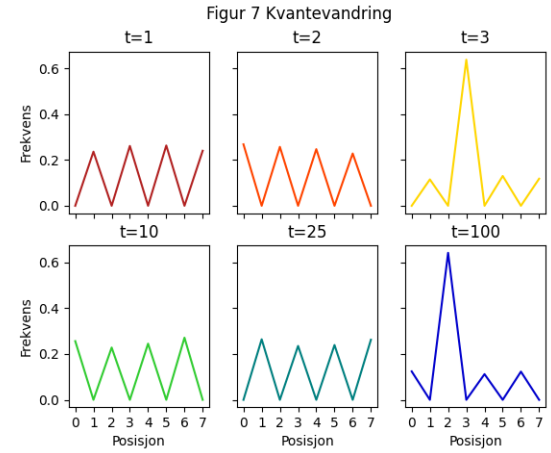
```
operation PoistionCoinWalk(position : Qubit[],
                           color : Qubit[],
                           Shift : ((Qubit[],Qubit[]) => Unit is Adj),
                           Coin : (Qubit[] => Unit is Adj),
                           steps : Int)
                           : Unit is Adj {
    for i in 1..steps {
        Coin(color);
        Shift(position, color);
    }
}
```

End of code

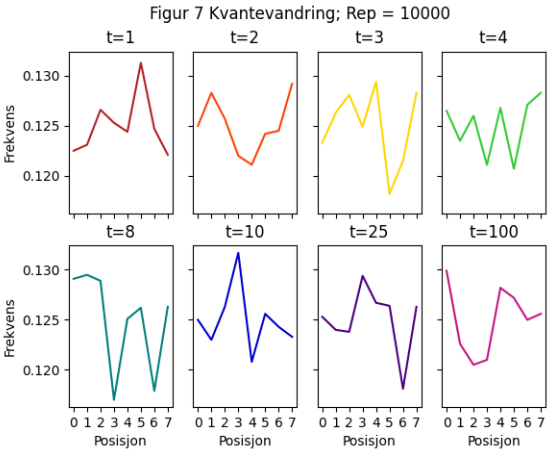
For å implementere enn konkret kvantevandring trenger vi å bestemme hva operatorene Shift og Coin er. Vi starter med å studer Coin. Dimensjonen som denne operatoren operer over er d, det kant kromatiske tallet. Mynten kan være en vilkårlig operator som virker på dette rommet. Naturlige valg er mynter som har uniforme sannsynlighetsfordelinger, derav Hadamardmynten, Grovermynten og Fouriermynten.

Figur 13: Kvantevandring over 7

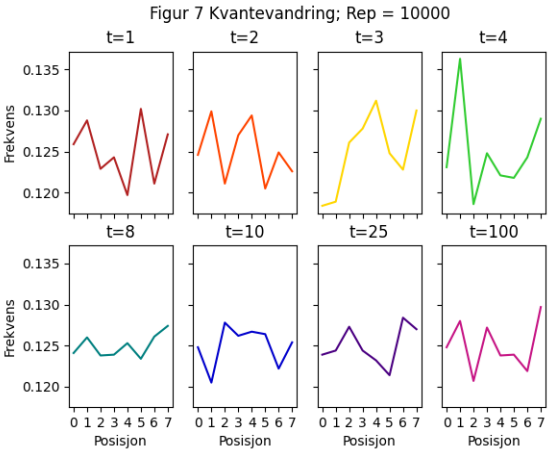
(a) start i node 0, Hadamardmynten



(b) start uniformt, Hadamardmynten

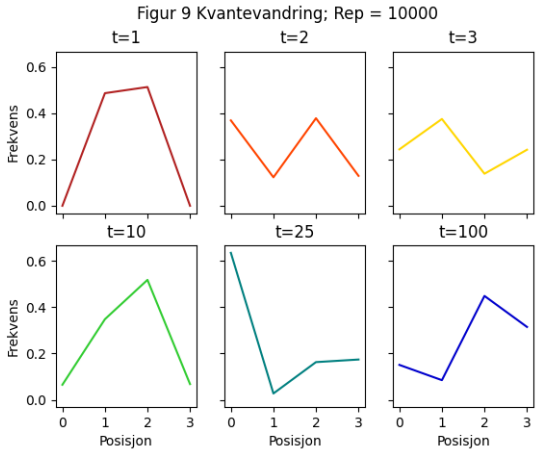


(c) start uniformt, Grovermynten

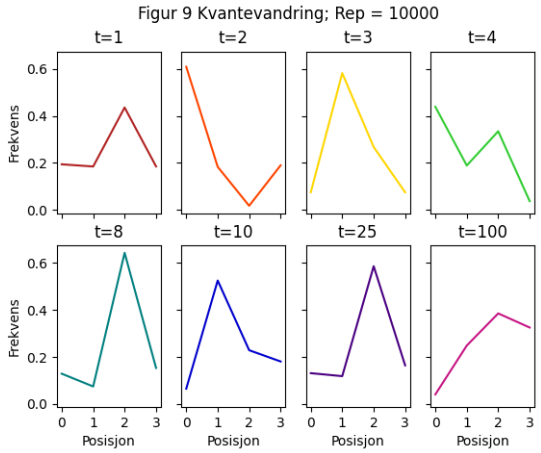


Figur 15: Kvantevandring over 9

(a) start i node 0



(b) start uniformt



Shift operatoren er mer interessant for denne strukturen. Siden vi bruker flip-flop operatorene har vi faktisk at Shift operatoren er unikt bestemt av fargeleggingen av kantene. Dette gjør at strukturen til grafen er totalt fanget av implementasjonen til Shift operatoren.

Som vi ser i figur 13, så har kvantevandringene en del tilfeldige egenskaper etter hvordan de beveger seg. Vi ser f.eks. at i del (a) hvordan vandringen er nesten uniform i de to første stegene, men i steg 3 for den en topp. Denne toppen skyldes nok kansellering og konstruksjon av de forskjellige fasene som blir sendt rundt. I figur (b) og (c) er disse effektene enda tydeligere, ettersom grafene er mer eller mindre tilfeldig. Det er vanskelig å si om disse effektene skyldes av interferens, eller om de skyldes av feil i implementasjon eller kjørefeil under simuleringen.

I figur 13 (b) bruker vi Hadamardmynten og (c) bruker vi Grovermynten. Til sammenligning kan vi se at interferensen opptrer forskjellig. Karakteristikken til kvantevandringene er vanskelig å skille på, ettersom alt virker tilfeldig. I figur 13 (a) er det mye tydligere å se at enkelte mønstre gjentar seg. Dette motiverer quasi-periodisiteten til kvantevandringene.

4.2.2 Arc notation

I arc notation foregår kvantevandringen over Hilbertrommet $\mathcal{H}_V \otimes \mathcal{H}_V$, dette er representert som *present* og *past*. Den abstrakte definisjonen for en slik kvantevandring er gitt under.

Beginning of code

```
// Arc flip-flop operator (MultiSWAP gate)
operation ArcFlipFlop(present : Qubit[], past : Qubit[]) : Unit
is Adj {
    // Get qubit length
    let presentSize = Length(present);
    let pastSize = Length(past);

    // past and present qubit size should be the same
    Fact(presentSize == pastSize, "Not a valid format");

    // Switch wires
    for i in 0..(presentSize-1) {
        SWAP(present[i], past[i]);
    }
}

// The coin operator is determined by each nodes neighbors
operation ArcWalk(present : Qubit[],
                  past : Qubit[],
                  Coin : ((Qubit[],Qubit[]) => Unit is Adj),
                  steps : Int)
                  : Unit is Adj {
    for i in 1..steps {
        Coin(present, past);
        ArcFlipFlop(present, past);
    }
}
```

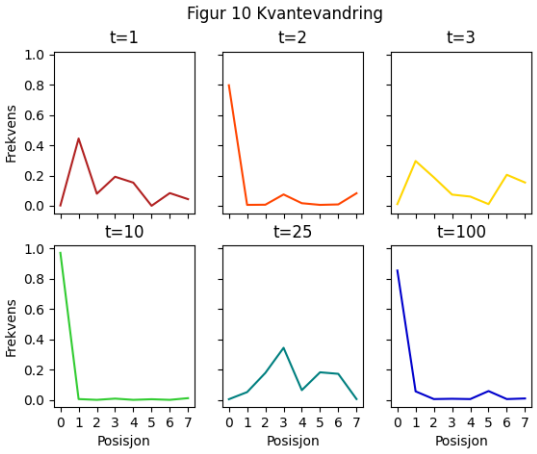
End of code

Som vi kan se i den abstrakte beskrivelsen av arc notation kvantevandringen under trenger vi ikke å eksplisitt oppgi en Shift operator. Denne operatoren er definert som ArcFlipFlop, og er ekvivalent med den symmetriske isomorfien av tensorproduktet. Dette medfører at strukturen til grafen må fanges i Coin operatoren. Tidligere har vi beskrevet denne Coin operatoren som en direktesum av Coin operatorer, hvor hver summand tilsvarer en node, og dimensjonen til operatorene er graden til noden. Her har vi også mer valg for Coin operatoren, ettersom den kan sammensveises av forskjellige typer mynter, som kan gi opphav til mer eller mindre optimale kvantevandringer.

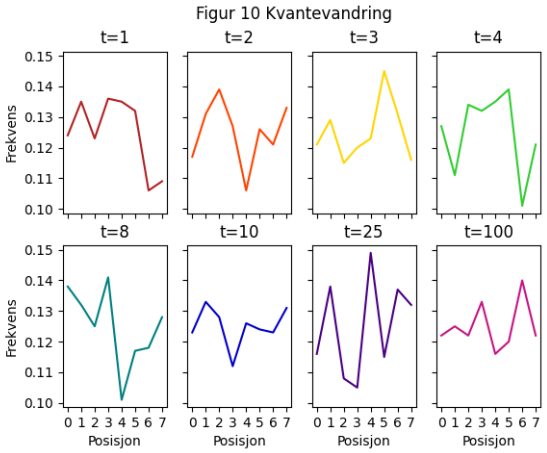
I figur 15 kan vi se mange av de samme interferens effektene som i det d regulære tilfellet. For å illustrere dette enda tydeligere ser vi på $t = 1$ og $t = 2$ i (b) figuren. Vi vet at node 2, 3 og 4 har kanter til node 3. Selv om nesten alle nodene går inn i node 3, ser vi hvordan denne interferensen ødelegger sjansen for å ende opp i node 3 når $t = 2$. For å illustrere quasi-periodisiteten ser vi at i (a) er $t = 1$ og $t = 10$ nesten den samme, og i (b) ser vi det for $t = 3$ og $t = 10$.

Figur 17: Kvantevadnring over 10

(a) start i node 0



(b) start uniformt



4.2.3 Staggered model

For kvantevandringen vi utfører på denne modellen har jeg ikke klart å lage en implementasjon som gir meg ønsket kvantevandring. Den implementasjonen som man kan finne på repoet, gir desverre en annen kvantevandring, enn den som er spesifisert tidligere i rapporten. Dette eksempelet er derimot beholdt, ettersom vi bruker denne vandringen for å illustrere QSS. Den abstrakte implementasjonen ligger fremdeles under, og her foregår kvantevandringen over Hilbertrommet $\mathcal{H}_V$, representert som *position*.

Beginning of code

```
// The staggering is determined by the chosen graph tessellation cover
operation StaggeredWalk(position : Qubit[],
                        Staggering : (Qubit[] => Unit is Adj),
                        steps : Int)
                        : Unit is Adj {
    for i in 1..steps {
        Staggering(position);
    }
}
```

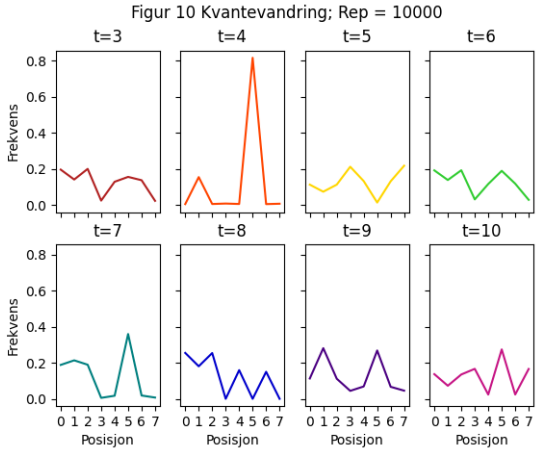
End of code

Denne vandringen er definert unikt utifra graftesselleringsdekket som er valgt. Det er operatoren Staggering som skal være den unitære vandringsoperatoren som er definert slik. Vi kan se på kvantevandringen som er implementert. Vi ser i dataen for den i figur 17 at vi har de samme tilfeldige mønstrene. I (a) hinter det sterkt til at vi har en quasi-periodisitet på 2, ettersom den samler seg tilbake i node 0 for hvert partall.

I (b) av figur 17 blir det åpenbart at kvantevandring som er implementert ikke stemmer overens med den som er spesifisert. Hvis vi husker operatoren U som er komposisjonen av de tre hermitiske operatorene, så er det denne operatoren som skal implementeres. For å implementere U trenger vi å beskrive U som en komposisjon av de universelle unitære operatorene. Dette problemet kalles for unitær dekomponering. På denne fronten er det funnet mange resultater, blant annet at en generell dekomponeringsalgoritme kan ikke være bedre enn $\mathcal{O}(4^n)$ porter, hvor n er antallet qubits. ZYZ-dekomponering er en simpel algoritme, men har ikke så god ytelse i forhold til denne nedre begrensingen. Optimalisert Shannon dekomponering ligger derimot på den teoretiske nedre begrensingen for små qubits [8]. For denne implementeringen bruker vi Python pakken utviklet av Dmytro [9]. Denne pakken har som hovedfunksjonalitet ved å generere Q# kode gitt en unitær matrise.

Måten vi ser at denne kvantevandringen ikke følger spesifikasjonene sine gjør vi ved å finne fikspunkter til operatoren U. En av disse fikspunktene er vektoren $H^{\otimes 3}|0\rangle$, altså at $U \circ H^{\otimes 3}(|0\rangle) = H^{\otimes 3}|0\rangle$. Vi kan enkelt observere at i figur 17 (b) at $t = 1$ og $t = 2$ ikke er det samme plottet, og dette bryter med observasjonen av fikspunktet. Tulsi sin beskrivelse av kvantesøk [6] sier at vi fremdeles kan bruke denne operatoren som en kvantevandrings operator, og dette er grunnen for at vi overseer denne feilen.

Figur 19: QSS søk over 10, med tidligere kvantevandring



4.3 Kvantesøk

Vi illustrer et eksempel av QSS algoritmen. Denne delen har som formål med å illustrere hvordan man kan gjøre den nødvendige forhåndsanalysen, og vise en mulig implementasjon av en slik søke algoritme. Den abstrakte søke algoritmen er gitt under.

Beginning of code

```
// Quantum Search
operation QuantumSpatialSearch(register : Qubit[],
                                    quantumWalk : (Qubit[], Int) => Unit is Adj,
                                    phaseOracle : Qubit[] => Unit is Adj,
                                    steps : Int,
                                    weight : Int)
                                    : Unit is Adj {
    // Spatial Search
    for i in 1..steps {
        phaseOracle(register);
        quantumWalk(register, weight);
    }
}
```

End of code

Vi skal illustrere QSS algoritmen på figur 10, med graftesselleringsdekket som er gitt. For dette QSS eksperimentet skal vi anta at den 5 noden er merket, det vil si at det er den noden som vi skal nå fram til. La $\psi(t) = U^t(\psi(0)) = U^t(H^{\otimes 3}|0\rangle)$ være tilstanden til systemet ved tid t. Sjansen for å måle node 5 er dermed $p(t) = |\langle 5|\psi(t)\rangle = \langle 5|U^t(\psi(0))\rangle$. Om vi finner den t-en som maksimerer denne funksjonen, så finner vi hvor lenge vi skal kjøre algoritmen. Her er det mange metoder man kan bruke, men det anbefales å numerisk tilnærme seg maksima. Vi går videre ved å bruke simplexalgoritmen, dette avslører at $t = 5$ gir tilnærmet maksima, med $p(t) \approx 0,27$. For å forbedre denne sannsynligheten opp til $0,81$ kan vi bruke amplitudeforsterkningsteknikken. Ved å anta at $p(t) \approx 0,25$, får vi at $k = \lfloor \pi/4\sqrt{p} \rfloor \approx \lfloor 2\pi/4 \rfloor = 1$.

I figur 19 illustrerer vi dette med kvantevandringen fra figur 17. Siden denne operatoren er litt annerledes enn den som vi har spesifisert ender vi opp med maksimum på $t = 4$. Ved å amplifisiere amplituden klarer vi øke den til ≈ 0.8. Merk at på $t = 7$ har vi enda en lokal maksima, og her kan man også bruke amplitudeforsterkningsteknikken for å gjøre det veldig sannsynlig å treffe node 5 etter måling.

REFERANSER

[1] R. L. Jaffe. Supplementary notes on dirac notation, quantum states and etc. http://web.mit.edu/8.05/handouts/jaffe1.pdf, 2007.

[2] Renato Portugal. *Quantum Walks and Search Algorithms*. Springer, 2 edition, 2019.

[3] Ronald de Wolf. Quantum computing: Lecture notes, 2021.

[4] Salvador Elías Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum Information Processing*, 11(5):1015–1106, Jul 2012.

[5] R. Portugal, R. A. M. Santos, T. D. Fernandes, and D. N. Gonçalves. The staggered quantum walk model. *Quantum Information Processing*, 15(1):85–101, Oct 2015.

[6] Avatar Tulsi. General framework for quantum search algorithms. *Phys. Rev. A*, 86:042331, Oct 2012.

[7] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746–2751, Oct 1999.

[8] A. M. Krol, A. Sarkar, I. Ashraf, Z. Al-Ars, and K. Bertels. Efficient decomposition of unitary matrices in quantum circuit compilers, 2021.

[9] Dmytro Fedoriaka. Decomposition of unitary matrix into quantum gates. 06 2019.

[10] Bradben, dime10, geduardo, cjgronlund, rmshaffer, and gillenhaalb. Q# user guide. https://docs.microsoft.com/en-us/azure/quantum/user-guide/0, 2021.

A Q# INTRO KODE

Beginning of code

```
namespace ReportQ {
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;

    // Functions are made using the function keyword.
    // Functions are pure, i.e. given the input, the output should always be the sa
    function HelloWorld() : Unit {
        Message("Hello World!"); // Message : String -> Unit
    }

    // Arguments and output needs to be typed.
    function Add(x : Int, y : Int) : Int {
        return x+y;
    }

    // The operation keywords allows impure functions.
    // This is necessary in order to allow quantum weirdness.
    operation HelloQuantumWorld() : Unit {
        // "use" keyword creates qubits.
        // Qubits are initialized in the state |0⟩.
        // Qubit() means to create exactly 1 qubit.
        use register = Qubit();

        // H is the Hadamard transform, apply H to the register
        H(register);

        // Controlled statements are made using if-else
        // To measure a qubit we use the M function, M : Qubit -> Result.
        if M(register) == Zero {
            Message("Hello Quantum World");
        }
        else {
            Message("Bonjour le monde quantique");
        }

        // Before the operation is ended,
        // every qubit created in the operation needs to be released.
        // A qubit is released if it is in the state it was initialized in.
        // We use Reset to send a qubit to the state |0⟩.
        Reset(register);
    }

    operation HelloMultiQuantumWorld() : Unit {
        // The keyword Qubit[n] creates n qubits.
        use register = Qubit[2];

        // This function applies H to each qubit, see documentation for more info.
        ApplyToEachCA(H, register);

        // There are two ways to initialize variables.
        // let keywords creates immutable objects, they may not be changed.
```

```
52              let a = 2;
53
54              // mutable keywords creates mutable objects,
55              // they may be changed using the set keyword.
56              mutable num = 0;
57
58              // For loops are used to iterate over an iterable object
59              // Repeat loops are used instead of while loops
60              // for conditional iteration, see documentation for more info.
61              // While loops may be used safely within function statements.
62              for i in 0..1 {
63                  if M(register[i]) == One {
64                      // We use set to change num
65                      set num = num + 2^i;
66                  }
67              }
68
69              // Bigger if-else blocks may be constructed with elif.
70              if num == 0 {
71                  Message("Hello Quantum World!");
72              }
73              elif num == 1 {
74                  Message("Bonjour le monde quantique!");
75              }
76              elif num == 2 {
77                  Message("Hallo kvanteverden!");
78              }
79              else {
80                  Message("Kon’nichiwa ryōshi sekai!");
81              }
82          }
83
84          operation MoreOnOperators() : Unit {
85              // Single qubit operators.
86                  // Create a qubit in state |0⟩.
87                  use q = Qubit();
88
89                  // We have the Pauli matrices, X, Y, Z.
90                  X(q);
91                  Y(q);
92                  Z(q);
93
94                  // The Hadamard transform.
95                  H(q);
96
97                  // Rotation transforms,
98                  // around the x, y and z axis respectively.
99                  // We need a double to set the operator.
100                 let theta = 1.57;
101                 Rx(theta, q);
102                 Ry(theta, q);
103                 Rz(theta, q);
104
105                 // The pi/4 and pi/8 phase operator.
106                 S(q);
107                 T(q);
```

```
108
109             // Multi qubit operators.
110                 // Create qubits in state |00⟩.
111                 use qs = Qubit[3];
112
113                 // Controlled not gate.
114                 CNOT(qs[0],qs[1]);
115
116                 // SWAP gate.
117                 SWAP(qs[0],qs[1]);
118
119                 // Some operator have the Controlled or Ctl property.
120                 // These operators may be controlled .
121                 // or multi-controlled with the keyword Controll.
122
123                 // Controlled Hadamard.
124                 Controlled H([qs[0]],qs[1]);
125
126                 // Multicontrolled Pauli X.
127                 // This is the same as CCNOT.
128                 Controlled X([qs[0],qs[1]],qs[2]);
129
130             // Conjugated operators.
131                 // Some operators have the Adjoint or Adj property.
132                 // Operators with this property may be auto inverted .
133                 // (see docs to see how to implement this property by hand)
134
135                 // This auto-inversion allows us to conjugate an operator.
136                 // An operator A is conjugated by B whenever we do BAB^-1.
137                 // Conjugation is done in Q# with a within-apply block.
138                 // Within represents B and apply represents A.
139                 within {
140                     X(q);
141                 }
142                 apply {
143                     H(q);
144                 }
145
146                 // This is the same as:
147                 // (Note that X is hermitian, so X = X^-1)
148                 X(q);
149                 H(q);
150                 X(q);
151         }
152
153         // An operation is defined to have the Adjoint property
154         // if it returns type Unit is Adj
155         // It is defined to have the Controlled property
156         // if it returns the type Unit is Ctl
157         // It has both property if it returns the type Unit is Adj + Ctl
158         operation AdjointAndControlled(target : Qubit) : Unit
159         is Adj + Ctl {
160             H(target);
161         }
162     }
```

End of code