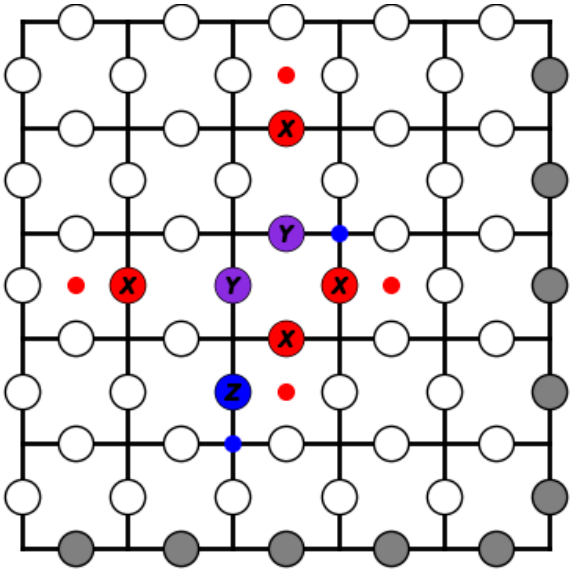
CHALMERS



Monte Carlo-baserat belöningsprotokoll till djup Q-inlärningsdekoder för torisk kod

Ett kandidatarbete i fysik

KARL HAMMAR
WILLE LINDBERG
ALEXEI OREKHOV
ISAK PETTERSON
PATRIK WALLIN HYBELIUS
ANNA KATARIINA WISAKANTO

Institutionen för fysik
Chalmers Tekniska högskola
Göteborg, Sverige 2020

KANDIDATARBETE TIFX04-20-18

Monte Carlo-baserat belöningsprotokoll till djup Q-inlärningsdekoder för torisk kod

KARL HAMMAR
WILLE LINDBERG
ALEXEI OREKHOV
ISAK PETTERSON
PATRIK WALLIN HYBELIUS
ANNA KATARIINA WISAKANTO

CHALMERS

Institutionen för fysik
Chalmers tekniska högskola
Göteborg, Sverige 2020

Monte Carlo-baserat belöningsprotokoll till djup Q-inlärningsdekoder för torisk kod

Monte Carlo based reward scheme for deep Q-learning decoder for the toric code

KARL HAMMAR
WILLE LINDBERG
ALEXEI OREKHOV
ISAK PETTERSON
PATRIK WALLIN HYBELIUS
ANNA KATARIINA WISAKANTO

Handledare: Mats Granath, Institutionen för fysik
Examinator: Lena Falk, Institutionen för fysik

KANDIDATARBETE TIFX04-20-18
Institutionen för fysik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon +46 31 772 1000

Omslag: En $d = 5$ torisk kod som visar de grundläggande operatorerna. Ett syndrom illustreras med röda och blå punkter, och en möjlig felkedja visas med x-, y- och z-fel.

Typsatt i LaTeX
Göteborg, Sverige 2020

Sammandrag

Kvantdatorer förväntas vara betydligt snabbare än konventionella datorer på att lösa vissa typer av problem, med tillämpningar inom bland annat optimering och simuleringar av kemiska processer. En av de största utmaningarna med att bygga en kvantdator är att hålla kvantbitarna stabila, då de är känsliga för störningar. Ett intressant sätt att angripa problemet är med en torisk kod, som kan felkorrigeras med hjälp av en dekoder. I detta arbete utförs en detaljerad studie av en dekoder baserad på *Markov Chain Monte Carlo*, MCMC. En sådan dekoder har hög korrektionsfrekvens på bekostnad av beräkningstid. I ett försök att begränsa beräkningstiden används MCMC-dekodern i ett modifierat belöningsprotokoll för en existerande dekoder baserad på *Deep Reinforcement Learning*, DRL. Den tidigare DRL-dekodern antar, ibland felaktigt, att lösningen med minst antal steg är den mest sannolika. Det nya belöningsprotokollet baseras på sannolikhetsfördelningar och belönar utefter dessa den mest sannolikt korrekta lösningen. Då kan även lösningar som kräver fler steg men är mer sannolika övervägas vid felkorrigeringen. För de undersökta systemstorlekarna $d = 3, 5, 7$ observerades för MCMC-dekodern en korrektionsfrekvens högre än för referensalgoritmen *Minimum Weight Perfect Matching*, MWPM. Undersökningen gjordes vid felsannolikheter $p \in [0.05, 0.2]$ med *depolarizing noise*, för vilket x-, y- och z-fel är lika sannolika. För MCMC-dekodern mättes en felttröskel på 18.5%, vilket är mycket nära den teoretiska gränsen vid 18.9% och motsvarar alltså en nästintill optimal dekoder. Därefter jämfördes MWPM-, MCMC+DRL- och MCMC-dekodrarna för systemstorlekar $d = 3, 5$. Felttröskeln för MCMC+DRL mättes till 14% jämfört med 14.5% för MWPM. Resultaten motiverar därmed fortsatta studier med belöningsprotokoll baserade på sannolikhetsfördelningar.

Abstract

Quantum computers are expected to outperform conventional computers in fields such as optimization and simulations of chemical processes. One of the biggest challenges with constructing a viable quantum computer is the instability of qubits due to their susceptibility to noise. An interesting approach to the problem is to use toric code, which can be corrected using a decoder. This thesis is an in-depth study of a decoder based on *Markov Chain Monte Carlo*, MCMC. Such a decoder has a high error correction rate at the expense of long runtime. In an attempt solve this issue, the MCMC-decoder was applied in the reward scheme of an existing decoder, based on *Deep Reinforcement Learning*, DRL. This decoder sometimes incorrectly assumes that the solution with the least number of steps is the most probable. The new reward scheme is based on probability distributions and gives a reward for the most probable solution. This allows more probable solutions, that require a larger number of steps, to be considered. A higher error correction rate was observed for the MCMC-decoder compared to the reference algorithm *Minimum Weight Perfect Matching*, MWPM, for the studied system sizes $d = 3, 5, 7$. The study was carried out for error probabilities $p \in [0.05, 0.2]$ and *depolarizing noise*, where x-, y- and z-errors are equally likely. The error threshold for the MCMC-decoder was measured to be 18.5%, which is very close to the theoretical limit of 18.9% for error correction. Lastly the MWPM-, MCMC+DRL- and MCMC-decoders were compared for system sizes $d = 3, 5$. The error threshold for MCMC+DRL was measured to be 14% compared to 14.5% for MWPM. The result motivates further research of reward schemes based on probability distributions.

Keywords: quantum error correction, toric code, reinforcement learning,
Markov Chain Monte Carlo, parallel tempering

Förord

Datagenerering och träning av neurala nätverk utfördes till stor del med resurser på Chalmers Centre for Computational Science and Engineering (C3SE) tillhandahållet av Swedish National Infrastructure for Computing (SNIC). Ett speciellt tack riktar vi till Mikael Öhman på C3SE för hjälp med anpassning av aktuell toolchain för att köra vår kod på C3SE resurser.

Ett tack till Wallenberg Centre for Quantum Technology (WACQT), Anton Frisk Kockum och David Fitzek för den praktiska introduktionen till kvantdatorer. David bidrog också tidigt med genomgång av kod.

Ett stort tack utgår också till Mattias Eliasson för generering av MWPM-data och Gabriel Lindeby för initiala tips kring klusterkörnigar.

Ett speciellt tack utgår även till vår handledare, Mats Granath, som väglett oss under detta projekt.

Göteborg, maj 2020

Innehåll

1 Bakgrund

På grund av väsentliga funktionella skillnader förväntas kvantdatorer bli bättre än klassiska datorer på att lösa vissa typer av problem, med tillämpningar inom bland annat optimering, databassökning och simuleringar av kemiska processer [1]–[3]. Särskilt simuleringar av komplexa kemiska system har direkta samhälleliga konsekvenser, då det kan leda till bland annat mer energieffektiv kvävefixering och nya läkemedel [4], [5].

De fundamentala egenskaperna som utnyttjas i en kvantdator är sammanflätning och superposition, där en kvantbit till skillnad från en klassisk bit kan representera `0` och `1` samtidigt. Superpositionen är viktig då det innebär att n kvantbitar kan beskriva 2^n tillstånd samtidigt, vilket ger kvantdatorer deras teoretiskt överlägsna beräkningskraft, medan sammanflätningen innebär att man kan agera på flera tillstånd samtidigt. Ett exempel på denna beräkningskraft är att Google med sin 54-kvantbitars Sycamore-processor under hösten 2019 kunde uföra beräkningar, som skulle ha tagit världens snabbaste superdator $1 \cdot 10^4$ år att utföra,[1] på 200 sekunder [7].

En komplikation vid användningen av kvantdatorer är att de kvantbitar som kvantdatorn består av är fundamentalt instabila och påverkas av dekoherens [8]. Denna instabilitet leder till att information i systemet går förlorad. Ett sätt att hantera detta är att låta en uppsättning fysiska kvantbitar representera en logisk kvantbit [9]. De fel som uppstår på fysiska kvantbitar kan då korrigeras innan ett logiskt fel hinner uppstå, genom att använda en felkorrektionskod.

Ett problem med felkorrektionen är att de fysiska kvantbitarnas tillstånd inte kan mätas direkt, eftersom direkt mätning leder till att tillståndet kollapsar. Istället kan en indirekt mätning göras av fyra kvantbitar, vilket kan ge upphov till en defekt som kan mätas utan att kollapsa tillståndet. Flera olika felkedjor, det vill säga uppsättningar av fysiska fel, kan ge upphov till samma uppsättning av defekter. En sådan samling defekter benämns syndrom. Problemet kan då omformuleras till att hitta de felkedjor som med störst sannolikhet har orsakat syndromet.

Antalet möjliga syndrom för en felkorrektionskod av storlek d är 2^{2d^2-2}.[2] På grund av det stora antalet syndrom går det inte att lagra den optimala felkorrektionen för varje syndrom. Ett alternativ är att använda en felkorrektionsalgoritm som uppskattar den mest sannolika felkedjan utifrån ett givet syndrom. Vid design av algoritmer måste dock en avvägning alltid göras mellan noggrannhet och snabbhet. En metod som har potential att vara både noggrann och snabb är att använda förstärkningsinlärning för att träna ett neuralt nätverk som uppskattar den optimala korrigeringen av syndromet. Nätverket kan då lära sig att korrigera liknande syndrom på liknande sätt, och därmed minska antalet beräkningar som krävs.

I tidigare forskning har ett neuralt nätverk tränats för att hitta och korrigera felkedjor [10], [11]. Träningen av det neurala nätverket gjordes med djup Q-inlärning. Ett belöningsprotokoll som belönade eliminering av enstaka defekter och fullständiga elimineringar av syndromen. Ett sådant belöningsprotokoll skapar ett bias där korta felkedjor föredras. Syftet med detta arbete är att inled-

[1] Detta påstående har blivit kritiserat av IBM, som påstår att den klassiska beräkningen bara tar 2.5 dagar [6].
[2] Där storlekar som typiskt behandlas är $d = 5, 7, 9$ så som i [10].

ningsvis konstruera en dekoder baserad på *Markov Chain Monte Carlo*, MCMC, sampling för att hitta den mest sannolika typen av korrektionskedja. Detta för att sedan bygga vidare på den tidigare forskningen genom att förändra belöningsprotokollet så att det tar hänsyn till sannolikheten för den föreslagna korrektionskedjan. Sannolikheten kommer bestämmas med hjälp av den konstruerade MCMC-dekodern [11]. Specifikt kommer rapporten slutligen ha målet att jämföra den uppnådda korrektionsfrekvensen för de båda dekodrarna med referensalgoritmer. Rapporten kommer huvudsakligen behandla kvantfelskorrektion ur ett algoritmiskt perspektiv. Den faktiska implementeringen av korrektionskoden och andra fysikaliska detaljer såsom implementering av kvantbitar, behandlas ej.

2 Teori

I detta kapitel behandlas den teoretiska bakgrunden för studien. Inledningsvis presenteras den matematiska formalismen för de tre möjliga kvantfelen, x, y och z. Därefter förklaras syftet med kvantfelskorrektion och hur den möjliggörs av den toriska koden. Slutligen förklaras översiktligt principen bakom algoritmen *Minimum Weight Perfect Matching*, MWPM, och djup Q-inlärning, samt hur de appliceras på den toriska koden för kvantfelskorrektion.

2.1 Formalism för kvantbitfel

Här presenteras den matematiska formalism som behövs för att få en inledande förståelse för kvantbitar, det brus de utsätts för och de relevanta operatorerna. Detta utgör grunden för vad som krävs för att bygga upp och förstå logiken för en korrektionskod.

Den dekoherens som påverkar kvantbitarna kan beskrivas med olika brusmodeller, där en kvantbit i varje tidpunkt har en viss sannolikhet att få ett av de tre felen x, y och z. Exakt vad dessa fel fysikaliskt innebär behandlas inte här, enbart deras matematiska egenskaper.

Felen x, y och z beskrivs av operatorer som är både hermiteska och unitära. Det vill säga

$$\begin{aligned} a &= a^\dagger, \\ a^\dagger a &= a a^\dagger = \hat{1}, \end{aligned}$$

där $a^\dagger$ är adjunkten till operatorn a, och $\hat{1}$ är identitetsoperatorn. Därmed är x, y och z även självinversa och har egenvärden ± 1. Dessutom antingen kommuterar eller antikommuterar de. Det gäller att

$$\begin{aligned} xy &\propto yx \propto z, \\ yz &\propto zy \propto x, \\ zx &\propto xz \propto y, \end{aligned}$$

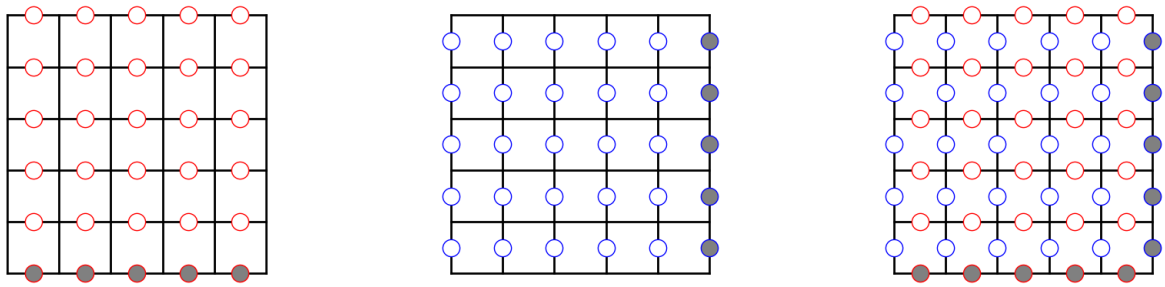
där proportionalitetskonstanterna är ± 1 eller $\pm i$. Att operatorerna är hermiteska innebär att de även definierar mätningar, som resulterar i ett av egenvärdena, ± 1. Proportionalitetskonstanterna behövs egentligen för att fullständigt beskriva alla möjliga fel som sker, och därmed för att beskriva kvantbitarnas tillstånd. Däremot påverkar de inte resultatet vid mätning, eftersom alla egenvektorer som kan relateras via en nollskild proportionalitetskonstant har samma egenvärde. Därmed försummas

proportionalitetskonstanterna, och förenklingen $y = xz$ införs. Mätningar på en enskild kvantbit kollapsar dess tillstånd, varför mätningar istället görs på en grupp av fyra kvantbitar. Detta beskrivs i avsnitt 2.2.

Som nämnt i avsnitt 1 påverkas kvantmekaniska system av störningar både från omgivningen och från interaktioner mellan kvantbitarna. Det ställs därför höga krav på att skydda den kvantinformation som är lagrad i systemet. I den forskning som sker idag läggs det särskild fokus på tre brusmodeller, som på olika sätt beskriver hur sannolika x-, y- och z-fel är att ske. Dessa är *depolarizing noise*, *uncorrelated noise* och *biased noise* [12]. Detta arbete behandlar endast *depolarizing noise*, där sannolikheten att ett fel sker på en given kvantbit är p, och att x-, y- och z-fel är lika sannolika. Det gäller alltså att $p_x = p_y = p_z = p/3$. För att hantera olika sorters brus och störningar i det kvantmekaniska systemet tillämpas en kvantfelskorrigerande kod.

2.2 Kvantfelskorrektion

En felkorrigerande kod är en struktur för att representera någon information på ett sådant sätt att oväntade fel som uppstår kan detekteras och korrigeras. Ett enkelt exempel på en felkorrigerande kod är den så kallade repetitionskoden. I en enkel repetitionskod repeteras varje bit, `1` eller `0`, för att skapa redundans. Exempelvis kan då informationen `101` representeras som tripletterna `111 000 111`. Om ett fel skulle uppstå på en av bitarna, exempelvis `111 010 111`, kan detta hanteras av dekodern genom att ta ett majoritetsbeslut över vilken siffra `0` eller `1` som finns representerad i varje triplett av bitar. I det kvantmekaniska fallet är inte den klassiska repetitionskoden lämplig eftersom mätningar inte kan göras på enskilda bitar. Istället behövs andra typer av strukturer.

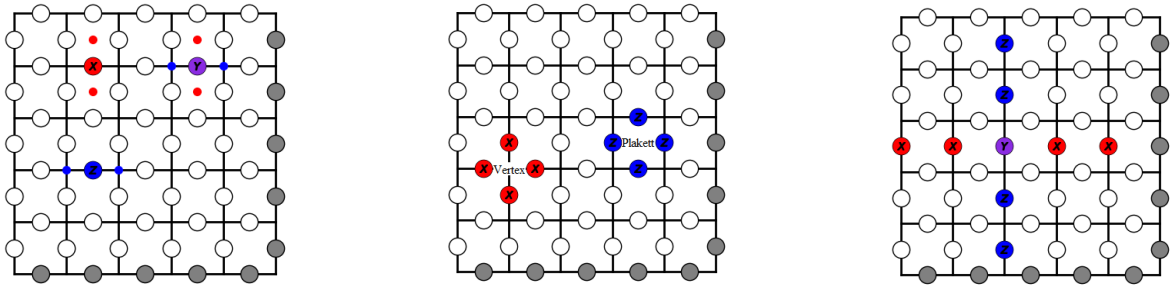


(a) (b) (c)

Figur 1: Uppställningsskiss för torisk kod med storlek $d = 5$. I (a) representeras lager 1 med kvantbitar på de horisontella kanterna och i (b) lager 2 med kvantbitar på de vertikala kanterna. Tillsammans representerar de ett $2 \times 5 \times 5$-gitter med fysiska kvantbitar i form av dubbelgittret i (c), som i sin helhet representerar en logisk kvantbit. De grå cirklarna representerar de periodiska randvillkoren.

En lovande klass felkorrigerande koder för kvantberäkningar är topologiska koder, av vilka den toriska koden är ett konceptuellt intressant exempel [13]. I den toriska koden arrangeras $2d^2$ fysiska kvantbitar i ett kvadratiskt $2 \times d \times d$ gitter med periodiska randvillkor, vilket illustreras i figur 1. Detta ger koden topologin av en torus, och därav sin benämning. Gittret kan betraktas som en överlagring av gitter i

två lager. I det ena lagret ligger kvantbitarna på de horisontella kanterna enligt figur 1(a) och i det andra lagret på de vertikala enligt figur 1(b). Dessa två lager formar tillsammans ett gitter enligt figur 1(c) och representerar en logisk kvantbit. En logisk kvantbit innebär här en representation av en kvantbit som med hjälp av felkorrigerande algoritmer kan skyddas från dekoherens. Det är dessa logiska kvantbitar som används för att lagra information och utföra kvantdatorberäkningar.



(a) (b) (c)

Figur 2: En torisk kod med storlek $d = 5$. Cirklarna representerar fysiska kvantbitar, där de gråa cirklarna representerar de periodiska randvillkoren. I (a) syns x-fel i rött, z-fel i blått och y-fel i lila. Där visas även defekter, som representeras av röda prickar på plaketter och blåa punkter på vertex. I (b) visas plakettoperatorerna som blåa ringar av z-fel som omringar en plakett, och vertexoperatorerna som röda ringar av x-fel som omringar en vertex. Plakett- och vertexoperatorerna benämns kollektivt som stabilisatorer. I (c) visas de logiska felen X och Z, som representeras av en rad eller kolumn fysiska x- eller z-fel som bildar en loop runt torusen.

För att illustrera de olika felen som kan uppstå på gittret, samt hur relevanta operatorer på den logiska kvantbiten ser ut, används figur 2. I figur 2(a) illustreras x-, y- och z-felen som ifyllda röda, lila respektive blå ringar. De grå ifyllda ringarna markerar de periodiska randvillkoren och representerar samma kvantbitar som visas på motsatt sida av gittret. Vi definierar en felkedja som den fullständiga uppsättningen fysiska fel. En felkedja ger upphov till defekter, som illustreras som röda prickar på plaketter och blå prickar på vertex. Slutligen definierar vi ett syndrom som en uppsättning av defekter.

I figur 2(b) illustreras stabilisatorer, som antingen är en loop av fyra x-operatorer runt en vertex, eller en loop av fyra z-operatorer runt en plakett. Defekterna i figur 2(a) är resultaten av paritetsmätningar som beskrivs av stabilisatoroperatorer. Med paritetsmätning menas en mätning på det gemensamma tillståndet för en grupp av fyra kvantbitar, och ger således endast begränsad information om de fysiska kvantbitarnas tillstånd. Därför kollapsar inte heller kvantbitarnas tillstånd. Paritetsmätningen blir $+1$ (ingen defekt) om ett jämnt antal fel skett på de fyra kvantbitarna, och -1 (defekt) om ett udda antal fel skett. Notera också att eftersom stabilisatoroperatorerna inte ger upphov till några defekter, kan de appliceras och användas till att deformera en felkedja utan att ändra syndromet.

Utöver x-, y- och z-operatorerna som verkar på fysiska kvantbitar, definierar vi de logiska operatorerna X, Y och Z som globala operatorer över gittret, som verkar på gittrets logiska tillstånd, den logiska kvantbiten. En logisk X- eller Z-operator består av en rad eller kolumn av x- respektive z-operatorer som omringar torusen utan att ge upphov till defekter. En logisk Y-operator består av en logisk X och en logisk Z i samma lager, vilket bildar ett kors och illustreras i figur 2(c). Om en logisk operator bildas till följd av fysiska fel säger vi att ett logiskt fel sker. Eftersom logiska operatorer inte ger upphov till några defekter är det omöjligt att detektera och korrigera logiska fel. För att den felkorrigerande koden ska fungera är det alltså av yttersta vikt att fysiska fel kan korrigeras snabbt nog att logiska fel inte hinner ske.

Utmaningen är alltså att från ett uppmätt syndrom hitta den bakomliggande kedjan av fysiska fel. Genom att sedan applicera samma felkedja, det vill säga att använda den som korrektionskedja, skulle alla fysiska fel och defekter elimineras, eftersom x-, y- och z-operatorerna är självinversa. Eftersom defekter endast innehåller information om de kringliggande fyra kvantbitarnas gemensamma, inte enskilda, tillstånd är det dock omöjligt att veta exakt vilken den underliggande felkedjan är. Faktum är att det för ett givet syndrom finns 2^{2d^2+2} möjliga underliggande felkedjor.[3] För gittret i figur 2 med $d = 5$ finns det alltså ca 10^{15} kedjor att undersöka. Dessa gitter är dock små jämfört med andra gitterstorlekar som studeras [14]. För exempelvis $d = 25$ är antalet felkedjor av storleksordning 10^{375}, vilket skulle vara omöjligt att hantera om varje kedja behövde undersökas. Problemet förenklas dock väsentligt om hänsyn tas till stabilisatorernas egenskaper. Eftersom applicering av en stabilisator inte ändrar syndromet, kommer den korrekta felkedjan även lösa syndromet efter applicering av en godtycklig mängd stabilisatorer.[4] Sammanslagningen av felkedja och korrektionskedja resulterar då i ett tillstånd utan defekter. Eftersom stabilisatorerna inte påverkar det logiska tillståndet är en sådan lösning ekvivalent med en lösning som använder en korrektionskedja identisk med ursprungskedjan. Eventuella skillnader mellan den applicerade felkedjan och den korrekta kommer att bilda så kallade triviala loopar. En trivial loop är en uppsättning stabilisatorer som inte ger upphov till något syndrom, och inte påverkar det logiska tillståndet.

En samling felkedjor som kan deformeras till varandra genom applicering av stabilisatorer definieras som en ekvivalensklass. För ett givet syndrom finns det 16 ekvivalensklasser, och de kan identifieras på följande sätt. Vi definierar först pariteten π_{ij} av antalet N i-fel i lager j som $\pi_{ij} = N \bmod 2$, där $i = x, z$ och $j = 1, 2$. Vi betraktar y-fel som ett x- och ett z-fel på samma kvantbit. Eftersom en stabilisator består av exakt 2 x- eller z-operatorer i båda gitterlager, kan applicering av en stabilisator aldrig ändra pariteten av antalet x- eller z-fel som finns i respektive lager. Om en felkedja har ett udda antal x-fel i lager 1, har alla andra kedjor i samma ekvivalensklass också det. Pariteten på antalet x- och z-fel i lager 1 och 2 identifierar därför en ekvivalensklass. Ekvivalensklasserna kan då numreras enligt följande. För varje kombination av operator och lager beräknas pariteten, $\pi_{x1}, \pi_{z1}, \pi_{x2}$ och π_{z2}. De fyra talen har två möjliga värden var, och kan tillsammans representera $2^4 = 16$ olika värden. Hur de kombineras är godtyckligt, men vi väljer en kombination som motsvarar fyra bitar i ett binärt tal. Resultatet blir ett decimalt heltal mellan 0 och 15, som vi låter representera varsin ekvivalensklass.

[3] Det finns 2^{2d^2-2} oberoende stabilisatorer och därmed lika många felkedjor i varje av de 16 ekvivalensklasserna; se ekvation (1).

[4] Med den korrekta felkedjan menas här den exakta felkedja som gav upphov till syndromet.

Från en felkedja kan ekvivalensklassen då beräknas enligt ekvation (1).

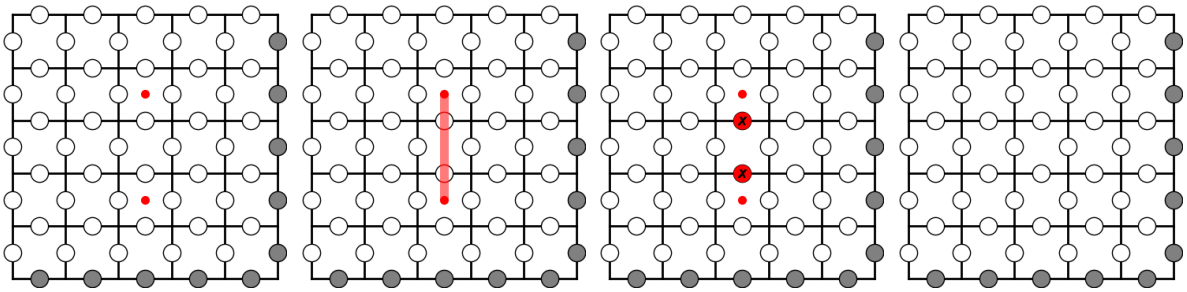
$$Eq = 8\pi_{z2} + 4\pi_{x2} + 2\pi_{z1} + 1\pi_{x1} \tag{1}$$

Till skillnad från stabilisatorerna ändras pariteten π_{ij} av logiska operatorer, givet att d är udda. Detta innebär att applicering av en logisk operator på en felkedja överför den till en felkedja i en annan ekvivalensklass, som ger upphov till samma syndrom. Om den logiska identitetsoperatorn I räknas med finns det 16 unika kombinationer av logiska operatorer, I, X, Y och Z i lager 1 kombinerat med samma uppsättning i lager 2. Givet en viss ekvivalensklass kan samtliga 16 ekvivalensklasser nås via applicering av någon av de 16 unika kombinationerna av logiska operatorer. Om en korrektionskedja i en ekvivalensklass appliceras på en felkedja i en annan ekvivalensklass elimineras syndromet, men resultatet blir ett logiskt fel som motsvarar den logiska operatorn som sammankopplar de två ekvivalensklasserna. Slutsatsen är alltså att en korrektionskedja i samma ekvivalensklass som den underliggande felkedjan alltid eliminerar syndromet utan logiska fel, medan en korrektionskedja i en annan ekvivalensklass alltid resulterar i ett logiskt fel. Problemet är därmed förenklat till att, givet ett visst syndrom, hitta den mest sannolika ekvivalensklassen.

För att eliminera felen i den topologiska koden krävs en algoritm som givet ett syndrom kan hitta en korrektionskedja som eliminerar syndromet utan att introducera logiska fel. För att undvika logiska fel bör en korrektionskedja från den mest sannolika ekvivalensklassen användas. En bra algoritm väljer därför med stor sannolikhet en lösning inom den ekvivalensklass som har störst sannolikhet att lyckas. Ett sätt att jämföra olika algoritmer för kvantfelskorrektion är att jämföra deras feltröskel. Dekodrar har större sannolikhet att lösa ett syndrom ju större gittret görs om vissa krav uppfylls. I analogi med repetitionskoden leder större redundans till större säkerhet att bibehålla informationen. Skulle däremot sannolikheten för ett fel i repetitionskoden vara över 50%, kommer repetitionskoden att prestera sämre med ökad redundans. En liknande feltröskel existerar för den toriska koden. Feltröskeln definieras som den maximala felsannolikhet vid vilken sannolikheten att en korrektion lyckas konvergerar mot 1 vid oändlig systemstorlek. Feltröskeln är även den felsannolikhet för vilken sannolikheten att en korrektion lyckas är oberoende av systemstorlek. Den teoretiska övre gränsen på feltröskeln för den toriska koden är omkring 18.9% vid *depolarizing noise* [15]. Alla praktiskt realiserbara dekodrar har en feltröskel under det teoretiska värdet. I nästa avsnitt beskrivs en sådan dekoder vars feltröskel ofta används som riktvärde i litteraturen.

2.2.1 MWPM

MWPM, är en kvantfelskorrektionsalgoritm som baseras på parvis matchning och eliminering av defekter. Algoritmen är relativt simpel och används därför ofta som en referens, vilken nya algoritmer jämförs mot. MWPM överför i grunden elimineringen av syndrom från den toriska koden till ett graf-problem, och löser detta. För *depolarizing noise* har den en feltröskel vid ungefär 14.5%, vilket är betydligt lägre än den teoretiska gränsen 18.9% [15]. För att förstå varför beskrivs här den grundläggande principen bakom MWPM.



(a) Initialt syndrom med två defekter.

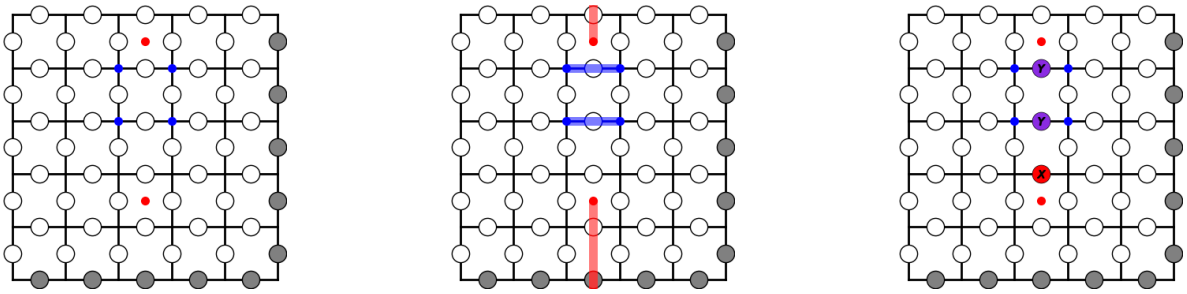
(b) MWPM kopplar ihop de två defekterna.

(c) Motsvarande lösningskedja.

(d) Efter applicering av lösningkedja är syndromet löst.

Figur 3: Lösningsgång för MWPM-algoritmen på ett simpelt syndrom syns i figur (a) - (d). Kortast möjliga felkedja som binder ihop defekterna appliceras och eliminerar syndromet.

Utgångspunkten för MWPM är att defekter parvis elimineras mot varandra tills inga defekter kvarstår. Grundantagandet är att den kortaste korrektionskedjan har en hög sannolikhet att vara i rätt ekvivalensklass,[5] och därför paras defekterna ihop med så få steg som möjligt. Betrakta det enkla exemplet med endast två defekter i figur 3. I detta fall paras defekterna genom de två mellanliggande kvantbitarna. Vi behöver alltså utföra operationer på två kvantbitar för att lösa syndromet. Det skulle också gå att para ihop defekterna genom att gå andra hållet runt torusen, men då skulle tre operationer behövas. De kvantbitar som ingår i kortaste vägen motsvarar själva korrektionskedjan. Den operator som ska appliceras längs korrektionskedjan bestäms av vilken typ av defekt den parar ihop.



(a) Initialt syndrom med 6 st defekter.

(b) MWPM kopplar parvis ihop defekterna.

(c) Den kortaste korrektionskedjan innehåller y-fel.

Figur 4: Lösningsgång för MWPM på syndrom i (a). MWPM behandlar x- och z-fel som oberoende av varandra, och använder korrektionskedjan i (b) som består av 4 operatorer. Den missar däremot den kortaste möjliga kedjan i (c) med 3 operatorer då den inte tar hänsyn till y-fel.

Betrakta nu det mer komplicerade syndromet i figur 4. De parade defekterna och strecken dem emellan motsvarar lösningen enligt MWPM. Notera att eftersom MWPM parvis eliminerar syndrom, så kan

[5]Med kortaste korrektionskedja menas här den korrektionskedja som innehåller minimalt antal x- och z-operatorer.

den inte hantera y-fel, utan betraktar dem som ett x-fel och ett z-fel på samma kvantbit. Detta innebär att MWPM inte kan ta hänsyn till korrelationer mellan de olika felen, varför den presterar suboptimalt för *depolarizing noise*, där felen är korrelerade. En annan anledning till det suboptimala beteendet är antagandet om att den kortaste felkedjan har hög sannolikhet att korrigera felet. Det finns exempel på syndrom som har flera lika korta lösningar, alla inom olika ekvivalensklasser. I detta fall är det inte uppenbart att MWPM väljer den mest sannolika lösningskedjan.

2.2.2 Deep reinforcement learning

I detta avsnitt presenteras den grundläggande teori som behövs för att föra en diskussion kring övergripande koncept och problematik vad gäller djup Q-inlärning. För en mer djupgående förklaring av principerna bakom neurala nätverk och specifikt djup Q-inlärning, se [16].

Deep Q-Network, DQN, är en typ av *deep reinforcement learning*, DRL-algoritm, baserad på Q-inlärning och djupa neurala nätverk. I Q-inlärning beskrivs ett problem med hjälp av en omgivning som befinner sig i ett tillstånd. Genom att en agent i omgivningen utför en handling ändras tillståndet. Till varje par av tillstånd s_t och handling a_t hör en belöning r_t, där t är en diskret tidpunkt. Agentens syfte är att utföra den serie handlingar som maximerar den totala belöningen. Som ett mått på förväntad framtida belöning definieras Q-funktionen,

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a), \tag{2}$$

där $0 \leq \gamma < 1$ är en diskontineringsfaktor som viktar kortsiktiga belöningar över de som fås senare. Ett $\gamma = 0$ betyder att man inte bryr sig om möjliga framtida tillstånd och $\gamma \approx 1$ innebär att framtida belöning viktas lika mycket som omedelbar belöning. DQN bygger på att omgivningen som agenten agerar i kan beskrivas av en så kallad *Markov Decision Process*, MDP. Det innebär att belöningen r_t endast är beroende på tillståndet s_t och handlingen a_t. Vilka steg som agenten tidigare tagit för att komma till tillståndet s_t påverkar varken belöningen eller utfallet av en handling.

I traditionell Q-inlärning representeras Q-funktionen som en tabell över tillstånd, handlingar och Q-värden. Dock blir detta problematiskt när antalet tillstånd och handlingar blir stort. För komplicerade problem används därför neurala nätverk för att approximera Q-värdestabellen. Neurala nätverk interpolerar tillstånds/handlings-rummet och generaliserar till tillstånd som möjligen inte har besökts under träningen. Q-värden uppdateras varefter agenten utforskar handlingar, tills nätverket med Q-värden konvergerat och uppfyller ekvation (2).

Belöningen r_t måste vara definierad för varje tillstånd-handlingspar (s_t, a_t). Ibland är detta val uppenbart av problemet, om agentens mål är att handla på aktiemarknaden, så ges belöningen naturligtvis som vinsten/förlusten av respektive handling. För många problem måste den dock definieras manuellt. DQN kommer att maximera den totala ackumulerade belöningen, och det är därför viktigt att maximal ackumulerad belöning motsvarar optimalt beteende. Det är inte ett trivialt problem, och utformningen av belöningsfunktionen behöver skräddarsys för varje problem. Ett optimalt belöningsschema skulle bara ge belöning när algoritmen lyckats lösa problemet fullständigt. För de flesta komplexa problem är dock sannolikheten att detta händer av slumpen mycket liten. Utan några belöningar lär sig inte nätverket vad som är rätt och fel och kommer därför inte konvergera. För att leda nätverket i rätt

riktning kan belöningar även ges för mindre delmål. Detta gör att nätverket kan lära sig i steg, och uppnå ett delmål i taget. En nackdel med detta är att nätverket lär sig optimera för heuristik bakom delmålen, vilket inte nödvändigtvis innebär att den även presterar optimalt för slutmålet. Valet av belöningsfunktion är således en avvägning mellan en optimalt presterande algoritm och att effektivt kunna träna nätverket.

3 Metod

I detta avsnitt presenteras projektets arbetsgång i två delar. Inledningsvis beskrivs i avsnitt 3.1 konstruktionen av vad som härefter benämns vår MCMC-dekoder. Algoritmen är en Metropolis-Hastings algoritm, som snabbas upp med hjälp av så kallad *parallel tempering* beskriven i avsnitt 3.1.2, inspirerat av vad som gjordes i [14]. Vilka krav som ställs för konvergens undersöks. Hur konvergensen analyseras för att få fram parametrar för tillräcklig precision och hastighet presenteras i avsnitt 3.1.3. Slutligen förklaras i avsnitt 3.2 hur data från MCMC-algoritmen genereras, sparas och sedan används för att träna det neurala nätverket med ett belöningsprotokoll baserat på denna data.

3.1 MCMC-dekoder

För att framgångsrikt eliminera fel i den toriska koden behövs en metod för att givet ett syndrom hitta en korrektionskedja som eliminerar de bakomliggande faktiska kvantbitsfelen utan att introducera logiska fel. Till detta används ekvivalensklasserna som introducerades i avsnitt 2.2, där den föreslagna korrektionskedjan bör tillhöra den mest sannolika ekvivalensklassen för det aktuella syndromet, vilket en bra dekoder bör uppfylla. För att bestämma vilken ekvivalensklass som är mest sannolik används en MCMC metod för att sampla en stor mängd underliggande felkedjor givet ett syndrom, och utifrån dessa föra statistik över vilken ekvivalensklass som är vanligast. Vi kallar detta för en MCMC-dekoder. MCMC-dekodern tar en felkedja som input, genererar det motsvarande syndromet och ger en sannolikhetsfördelning över ekvivalensklasserna för underliggande felkedjor som output. MCMC-dekodern behöver alltså hjälp av någon annan dekoder för att skapa en möjlig initial felkedja, men denna behöver inte tillhöra en sannolik ekvivalensklass och är således inte en intressant del av MCMC-dekoderns funktion och kommer därför inte behandlas.

3.1.1 Metropolis-Hastings algoritmen

För varje syndrom finns det en underliggande fördelning av felkedjor. Vi kan också definiera en underliggande fördelning av ekvivalensklasserna för nämnda felkedjor. Det är denna fördelning vi vill hitta. Problemet är att det inte är enkelt att sampla en slumpmässig felkedja från fördelningen. För att komma runt denna problematik utnyttjas Metropolis-Hastings algoritm, som är en typ av MCMC-algoritm som syftar till att med korrekt sannolikhet sampla data utifrån en sannolikhetsfördelning som annars är svår att sampla ifrån. För sammanhanget av felkorrektion på en torisk kod gäller det att sampla felkedjor givet ett syndrom och sannolikheterna för respektive fel p_x, p_y och p_z. I vårat fall intresserar vi oss enbart för fallet då $p_x = p_y = p_z = p/3$, alltså *depolarizing noise*.

Ekvivalensklassfördelningen genereras enligt följande. Först genereras en felkedja med felsannolikheten p. Den initiala felkedjan kallas frö och betecknas e_0. Sannolikheten för e_0 ges av

$$P(e_0) = (1-p)^{2d^2-n_e}(p/3)^{n_e},$$

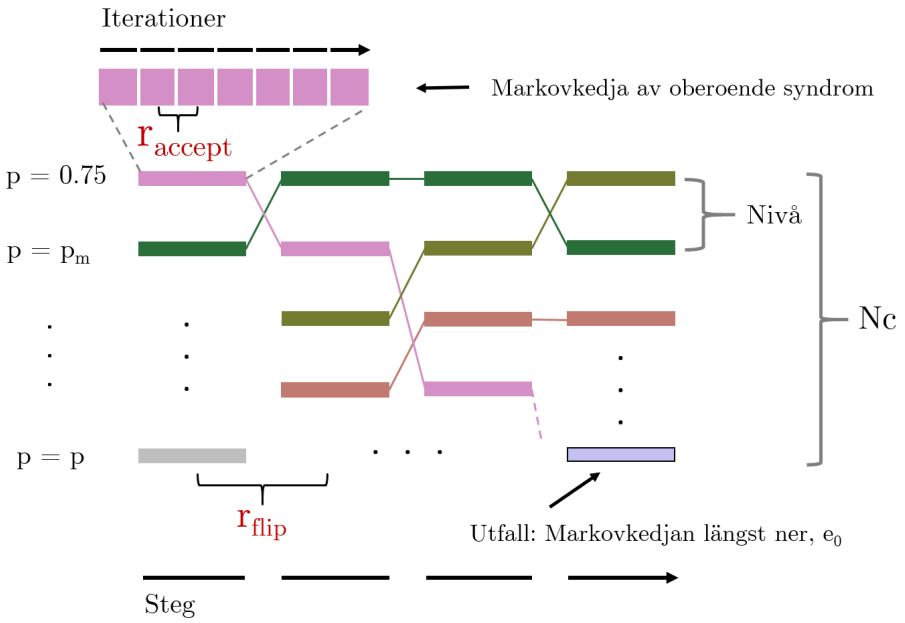
där n_e är antalet fel i e_0. Metropolis-Hastings algoritm används sedan för att upprepade gånger från e_0 sampla en ny felkedja som är konsekvent med syndromet. Detta kan göras genom att applicera en slumpvis operator som bevarar syndromet, alltså en stabilisator eller logisk operator. En ny felkedja e_0' bildas, som har samma syndrom men inte nödvändigtvis samma ekvivalensklass eller antal fel som den ursprungliga kedjan e_0. Sedan kan en kvot av sannolikheten för respektive felkedja bildas som

$$r_{accept}(e_0, e_0') = \frac{P(e_0')}{P(e_0)} = \left(\frac{p/3}{1-p}\right)^{n'-n}, \tag{3}$$

där n och n' är antalet fel i e_0 respektive e_0' och p är sannolikheten att ett fel uppstår. Om $r_{accept} > 1$ accepteras förändringen i den aktuella iterationen och e_1 sätts till $e_1 = e_0'$. Annars sätts $e_1 = e_0'$ med sannolikheten r_{accept} och $e_1 = e_0$ med sannolikheten $(1 - r_{accept})$. I de fall e_1 sätts till $e_1 = e_0'$ säger vi att förändringen accepterades. Notera att förändringen alltid accepteras om $n' < n$ och $p \leq 0.75$, alltså om den föreslagna felkedjan är mer sannolik än den befintliga felkedjan. Vi betraktar fördelningen över ekvivalensklasser för alla genererade e_k, och fortsätter sampla nya felkedjor e_k till dess att denna har konvergerat. När fördelningen har konvergerat har vi fått en sannolikhetsfördelning över ekvivalensklasserna, som motsvarar det givna syndromet.

3.1.2 Parallel tempering

För att kunna approximera den faktiska sannolikhetsfördelningen är det viktigt att sampla felkedjor från alla möjliga felkedjor som svarar mot samma syndrom och därmed från alla ekvivalensklasser. Problemet med Metropolis-Hastings algoritmen är att antalet fel efter applicering av en logisk operator, n', ofta är större än n och att $n' - n$ är av storleksordning d, där d är gittrets storlek. Detta leder till att sannolikheten att acceptera ett sådant fel går som p^d för små p och sannolikheten är alltså mycket liten för allt större gitter. Detta leder i sin tur till att det tar exponentiellt lång tid för ett sådant fel att godkännas av Metropolis-Hastings algoritmen, då kvoten r_{accept} blir liten.



Figur 5: Illustration av *parallel tempering*. En torisk kod med tillhörande felkedja förändras genom slumpmässig applicering av operatorer i varje iteration. Ett utfall är felkedjan för den lägsta nivån när ett steg är slutfört, r_{accept} är sannolikheten att applicering av en operator accepteras, och r_{flip} är sannolikheten att två kedjor byter plats efter ett steg. Antalet nivåer benämns `Nc`.

För att minska tiden till konvergens används utbytesmetoden för kopior av markovkedjor, *parallel tempering*,[6] som beskrivs i detalj i [14]. En schematik för *parallel tempering*, se figur 5, illustrerar de `Nc` stycken instanser av Metropolis-Hastings algoritmen, organiserade i så kallade nivåer, som körs samtidigt. Felsannolikheten i den m:te nivån ges av $p_m = p + m \cdot \frac{0.75-p}{\texttt{Nc - 1}}$, där p är felsannolikheten som studeras och $m = 0, 1, 2, ..., \texttt{Nc} - 1$. I den översta markovkedjan där $p_{\texttt{Nc}-1} = 0.75$ appliceras både stabilisatorer och logiska operatorer slumpvis medan det bara appliceras stabilisatorer i de lägre kedjorna. För $p = 0.75$ blir alltid $r_{accept} = 1$ och alla ändringar godkänns. Efter ett visst antal iterationer byter närliggande kedjor plats enligt samma princip som i Metropolis-Hastings, utifrån kvoten

$$r_{flip}(e_m, e_{m+1}) = \frac{P(e_{m+1})}{P(e_m)} = \left(\frac{p_m}{p_{m+1}} \frac{1-p_{m+1}}{1-p_m} \right)^{n_{e_{m+1}} - n_{e_m}}, \tag{4}$$

där e_m är det aktuella tillståndet för markovkedjan i den m:te nivån, och n_{e_m} är antalet fel i markovkedja e_m.

Bytena sker konsekutivt från det översta kedjeparet ($m = \texttt{Nc} - 1$) och nedåt, vilket innebär att en felkedja har möjlighet att propagera flera nivåer nedåt under en bytesomgång. En omgång av `iters` antal iterationer följt av kedjeutbyten benämns som ett steg, och hela processen utförs under ett

[6] Namnet *parallel tempering* kommer ifrån liknelsen att ekvivalensklasserna är potentialgropar, och att en högre temperatur, i detta fallet sannolikheten p, krävs för att ta sig mellan dessa.

lämpligt antal steg, `steps`. Logiska operatorer genererar nya ekvivalensklasser i översta kedjan som sedan kan propagera ner till den nedersta nivån. På detta sätt undviks problemet där det är svårt för stora förändringar, logiska operatorer, att accepteras. De felkedjor som är i lägsta kedjan i slutet av varje steg kallar vi för utfall. Ekvivalensklasserna för de genererade utfallen kan slutligen bestämmas och fördelningen över ekvivalensklasser approximeras.

3.1.3 Konvergenskriterier

Vid approximering av ekvivalensklassfördelningen behöver en avvägning göras mellan beräkningstid och fördelningens precision. Detta görs med hjälp av konvergenskriterier, som avbryter MCMC-samplingen när tillräcklig precision uppnåtts. Precisionen avgörs implicit av hur mycket fördelningen förändras mellan varje steg i samplingen. När fördelningen konvergerat sker ingen förändring då fler steg tas, och därför antar vi att precisionen efter ett visst antal steg beror på hur lite fördelningen ändras. Då förändringen per steg är tillräckligt liten kan alltså konvergens anses vara uppnådd.

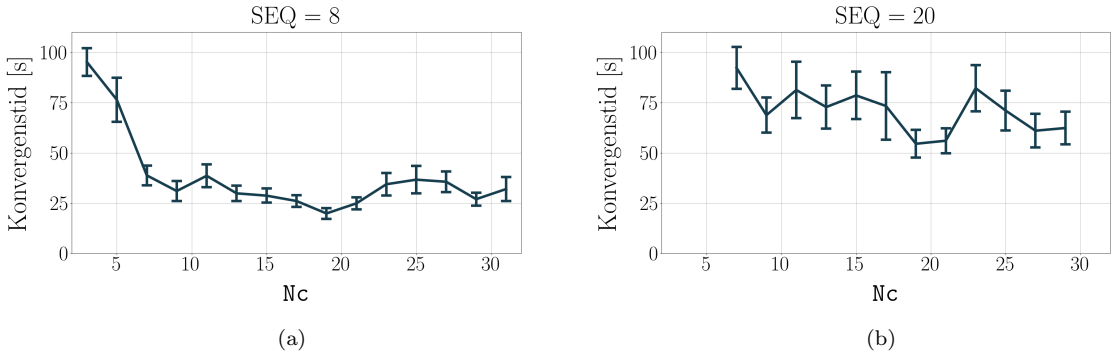
Som mått på denna förändring har vi valt att titta på hur medelvärdet av antalet fel i serier av utfall, det vill säga felkedjan i den nedersta nivån efter varje steg, skiljer sig mellan olika delar av datan. Utifrån antagandet att datan vid konvergens består av utfall som representerar den sökta fördelningen, kommer medelvärdet av antalet fel i utfallen bli lika i olika delmängder av datan. Utöver att förändringen per steg är tillräckligt liten behöver det även undersökas antalet möjligheter utfallen fått att byta ekvivalensklass. Som mått på detta används antalet felkedjor som propagerat från översta markovkedjan till nedersta, vilket benämns `tops0`. Förändringen mellan olika steg och `tops0` används tillsammans för att avgöra konvergens i ett konvergensekriterie kallat felkriteriet.

Felkriteriet är inspirerat av [14], och beskrivs sammanfattat i algoritm 1 [14]. I detta kriterium måste `tops0` först uppnå ett visst värde, `tops_burn`, varvid antalet fel i nedersta felkedjan efter varje steg börjar sparas. Om skillnaden i medelvärdet av antalet fel mellan andra och fjärde kvartilerna av datan är under en viss tolerans, `eps`, under tiden som `tops0` ökar med ett visst antal, `SEQ`, anses fördelningen vara konvergerad. Då det är medelvärdena av kvartilerna som jämförs för konvergens är det nödvändigt att en minsta mängd data samlas in innan konvergens evalueras. Detta implementeras genom villkoret att $\texttt{tops0} \geq \texttt{TOPS}$, för en parameter $\texttt{TOPS} > \texttt{tops\_burn}$. Först efter att villkoret uppfyllts jämförs differensen med `eps`. Eftersom kvartilerna förändras väldigt lite med varje enskilt extra steg, och för att bättre utnyttja beräkningstiden, undersöks konvergens endast var tionde steg.

Algoritm 1 Sammanfattning av felkriteriet

procedure BESTÄM KONVERGENS
`since_burn` = 0
for *step* = 1, 2, 3, ..., `steps` **do**
ett MCMC-steg
if `tops0` ≥ `tops_burn` **then**
`data[since_burn]` = antal fel i nedersta felkedjan efter MCMC-steg nr `step`
`since_burn` += 1
end if
if `tops0` ≥ `TOPS` **and** `since_burn` % 10 == 0 **then**
`fel` = abs(medelvärde(`data`[2:a kvartilen]) − medelvärde(`data`[4:e kvartilen]))
if `fel` ≥ `eps` **then**
`tops0` = `TOPS`
end if
if `fel` < `eps` **and** `tops0` − `TOPS` ≥ `SEQ` **then**
Konvergens uppnådd
end if
end if
end for
end procedure

För att säkerställa att felkriteriet ger en bra balans mellan noggrannhet och snabbhet behövs en analys av de inblandade parametrarna. Precisionen ökar med minskande `eps` och ökande `SEQ`. Precisionen påverkar i sin tur antalet steg som behövs för konvergens, som dessutom påverkas av antalet markovkedjor `Nc`. `Nc` påverkar dessutom tiden det tar att gå ett steg. På grund av detta kommer det för givna `eps` och `SEQ` finnas ett värde på `Nc` som minimerar konvergenstiden. Därmed behöver konvergenstiden för olika värden på `Nc` undersökas.



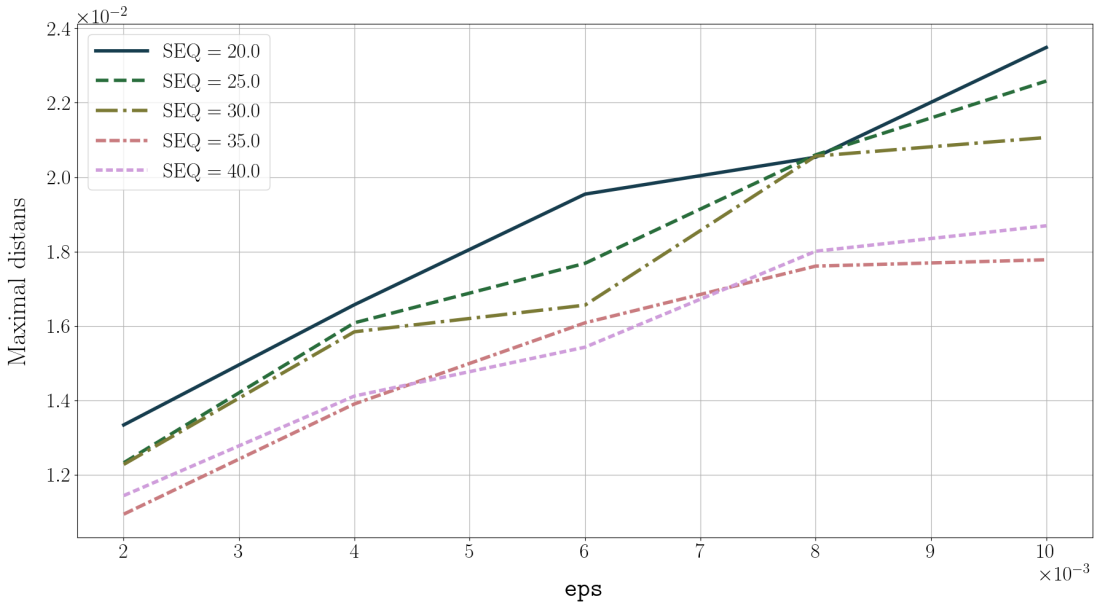
Figur 6: Konvergenstiden för algoritmen över antalet nivåer av markovkedjor, `Nc`, med `eps`=0.008 och `SEQ`=8 i (a) samt `SEQ`=20 i (b). Kurvorna visar medelvärdet av konvergenstiden för respektive `Nc` med standardavvikelsen markerad med felstaplar.

För att hitta ett bra värde på `Nc` gjordes inledningsvis ett antal testkörningar med olika parametrar. Detta för att intressanta områden för `SEQ` och `eps` skulle kunna identifieras. För `eps`= 0.008 och `SEQ`=8, 20 varierades sedan `Nc`, för att urskilja det antal nivåer som minimerar körtiden. Ett minimum hittades kring `Nc`= 19, enligt figur 6.

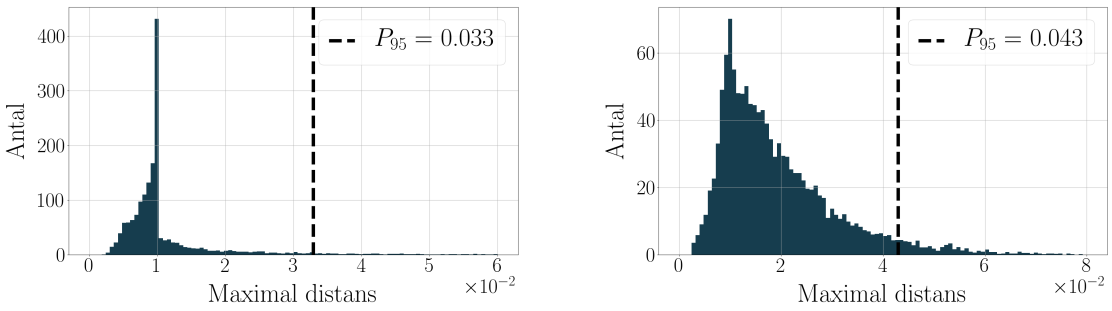
Efter att `Nc` bestämts till 19 gjordes en mer noggrann optimering av `eps` och `SEQ`. Basen i utrycket för $\mathtt{r_{flip}}$ i ekvation 4 antar ett värde som närmar sig 1 för ett ökande värde på `Nc`, samtidigt som exponentens väntevärde närmar sig 0. Detta innebär att utbyten mellan kedjorna i *parallel tempering* kommer ske oftare och därmed att `tops0` kommer öka fortare. Med detta i åtanke gjordes en mätning av konvergensen med `eps` i intervallet [0.002, 0.01] och `SEQ` i intervallet [20, 40]. Som ett mått på skillnaden mellan två sannolikhetsfördelningar använder vi oss av det vi benämner maximal distans, MD, som mäter maximala skillnaden mellan två fördelningar. Om vi har två sannolikhetsfunktioner σ och $\tilde{\sigma}$ definierar vi $\mathrm{MD}(\sigma, \tilde{\sigma})$ som

$$\mathrm{MD}(\sigma, \tilde{\sigma}) = \max_{E} |\sigma(E) - \tilde{\sigma}(E)|,$$

där E är en av de 16 ekvivalensklasserna. MD blir då den största skillnaden i sannolikhet för en given ekvivalensklass. Vid undersökningen antog vi att fördelningen konvergerat efter ett stort antal steg. Detta bestämdes empiriskt till 10^6; se till exempel figur 9. Med detta antagande mättes MD mellan den fördelning som uppnåtts efter 10^6 steg, σ_{10^6} och den som uppnåtts enligt felkriteriet, σ_{felkrit}. Då vi under mätningen av MD kunde observera att tidsåtgången var betydligt högre för höga värden på `SEQ` och små värden på `eps` gjordes en avvägning av tidsåtgången jämfört med precision. De parametrar som därmed valdes var `eps`= 0.006 och `SEQ`= 30 för vilka vi i figur 7 kan observera att MD < 0.018.



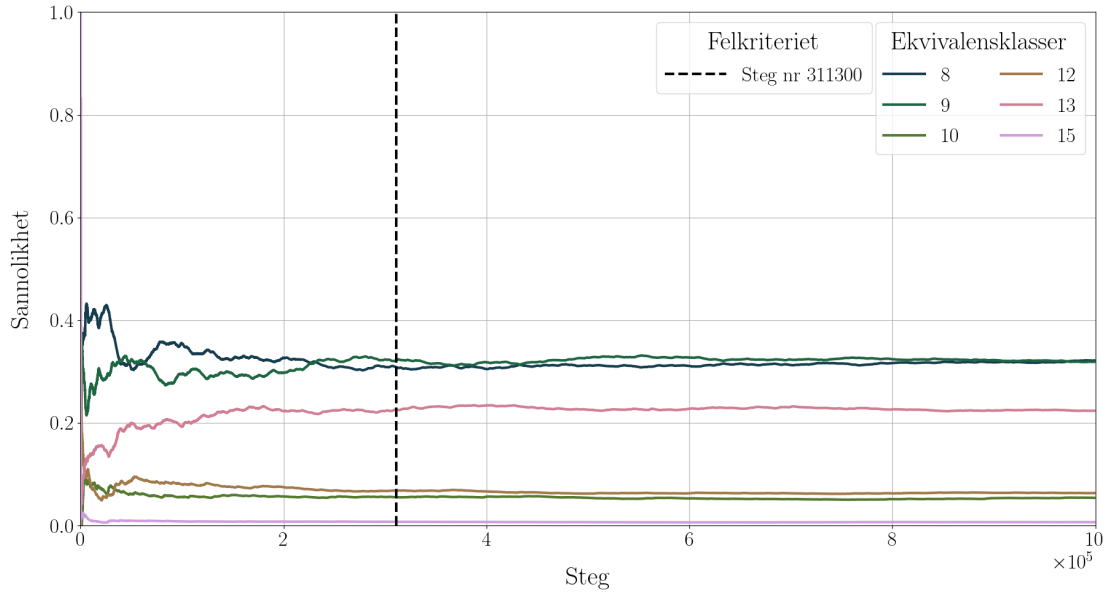
Figur 7: Medelvärdet av maximal distans, $\mathrm{MD}(\sigma_{\text{felkrit}}, \sigma_{10^6})$ som funktion av toleransen `eps`. Observera att de valda parametrarna `SEQ` = 30 och `eps` = 0.006 ger en maximal distans MD< 0.018.



(a) $p = 0.1$ (b) $p = 0.185$

Figur 8: Fördelning över maximal distans för felsannolikheter $p = 0.1$ i (a) och $p = 0.185$ i (b). I båda bilderna syns även värdet på MD för den 95:e precentilen. Vi har då P_{95} är 0.033 och 0.043 för $p = 0.1$ respektive $p = 0.185$.

När parametrarna `SEQ` $= 30$ och `eps` $= 0.006$ bestämts undersöktes $\mathrm{MD}(\sigma_{\text{felkrit}}, \sigma_{10^6})$ för de givna parametrarna nogrannare. Detta gjordes genom att undersöka hur fördelningen över MD såg ut för felsannolikheterna $p = 0.1$ och $p = 0.185$, se figur 8. Denna mätning visade att 95% av de frön som mätningen utgick från gav en fördelning σ_{felkrit} som maximalt skilde sig med 3.3 och 4.3 procentenheter för $p = 0.1$ respektive $p = 0.185$ från den korrekta fördelningen.



Figur 9: Sannolikheten för ekvivalensklasserna i olika steg av *parallel tempering* för syndromet på titelsidan. För tydlighetens skull visas endast de 6 mest sannolika ekvivalensklasserna. En streckad vertikal linje visar var konvergens uppnås enligt felkriteriet, vilket är efter 311300 steg. Observera att ekvivalensklasserna 8 och 9 verkar vara lika sannolika.

En bild av utvecklingen av ekvivalensklassfördelningen över tid fås från figur 9, där konvergensförloppet för ett specifikt syndrom över tid finns då 10^6 steg tagits med *parallel tempering* algoritmen. I figuren finns markerat med en sträckad vertikallinje när felkriteriet anser att konvergens är uppnådd för parametrarna `SEQ`= 30 och `eps`= 0.006, som i detta fallet skett vid 311300 steg.

När parametrar som ger MCMC-dekodern hög precision och kort körningstid hade identifierats, erhölls approximativa sannolikhetsfördelningar för vilken ekvivalensklass olika felkedjor tillhör. Dessa fördelningar kunde sedan utnyttjas i DRL-dekodern för att träna nätverket till att genomföra den kvantfelskorrektionen för torisk kod som eftersöktes.

3.2 DRL-dekoder

I denna del presenteras hur MCMC-dekodern används för att modifiera den existerande DRL-dekodern baserad på djup Q-inlärning i [10]. Detta görs genom att förändra befintligt belöningsprotokoll så att det utgår från sannolikhetsfördelningen genererad av MCMC-algoritmen.

Belöningsprotokollet för den tidigare DRL-dekodern gav belöning både för eliminering av enskilda defekter och för eliminering av syndromet. På detta sätt tränas nätverket att likt MWPM-dekodern hitta den kortaste korrektionskedjan, men presterar bättre då det även tar hänsyn till y-operatorn [10]. Belöningsprotokollet bildar dock ett bias att använda y-operatorn, eftersom denna operator har möjlighet att eliminera flest defekter,[7] detta är inte alltid optimalt. Både MWPM-dekodern och DRL-dekodern utgår ifrån antagandet att den kortaste felkedjan är den mest sannolika. Detta antagande är ofta välmotiverat, men fallerar exempelvis när det finns två kortaste felkedjor i två olika sannolika ekvivalensklasser. För att agera optimalt i en sådan situation behöver en dekoder kunna estimera sannolikheten att den underliggande felkedjan till syndromet tillhör en viss ekvivalensklass.

För att uppnå detta introduceras ett annat belöningsprotokoll. Istället för att ge belöning för varje borttagen defekt ges endast belöning när syndromet eliminerats, vilket undviker att specifikt träna nätverket att använda den kortaste korrektionskedjan, och även motverkar problemet med bias. Belöningen för eliminering av syndromet ändras till en belöning som är proportionell mot hur sannolik en föreslagen korrektionskedja är att lyckas. Denna sannolikhet är ekvivalent med sannolikheten att korrektionskedjans ekvivalensklass är korrekt. Den bestäms utifrån ett facit som förgenererats med hjälp av en MCMC-dekoder. Den stora fördelen med att använda ett neuralt nätverk gentemot MCMC-dekodern är att den kan korrigera fel mycket snabbare, vilket är avgörande för felkorrektion i praktiken.

3.2.1 Generering av data med MCMC-dekoder

MCMC-algoritmen är formellt inte en dekoder eftersom den okända underliggande felkedjan används som frö, men den kan trivialt göras om till en dekoder genom att som frö använda en godtycklig felkedja konsekvent med syndromet. Målet i detta projekt var att använda MCMC-dekodern för att generera vad som kommer att betraktas som ett facit för en optimal dekoder, och sedan träna ett nätverk att efterlikna detta facit. Eftersom körtiden för MCMC-algoritmen är lång, se avsnitt 3.1.3, är

[7] Två plakettdefekter och två vertexdefekter.

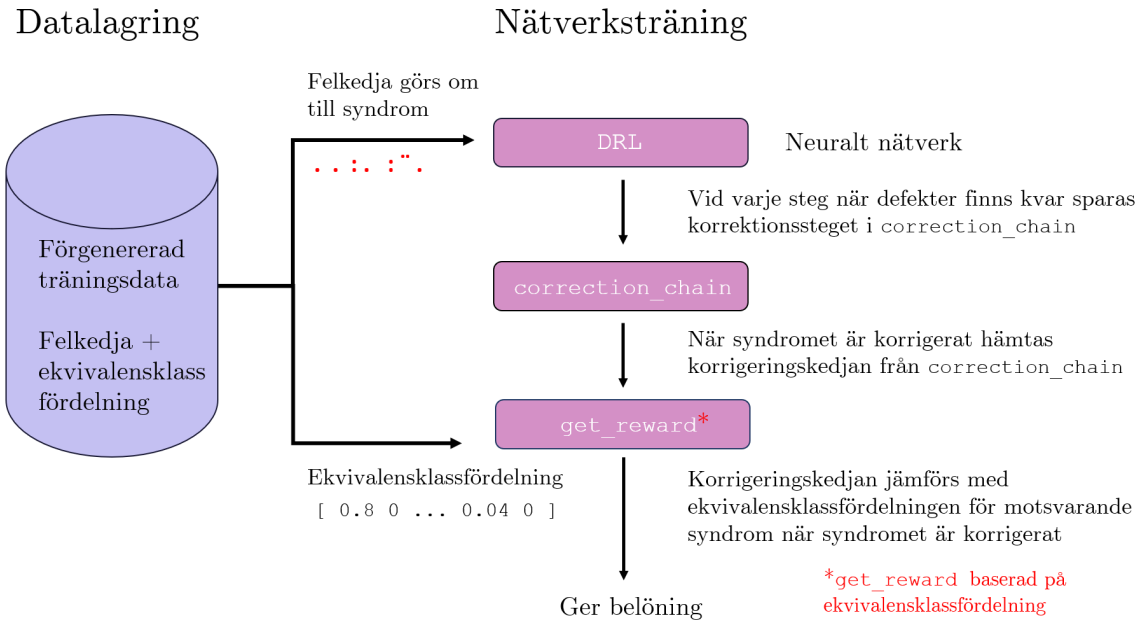
det inte praktiskt att köra denna för varje syndrom under träningen av nätverket. För att lösa detta problem genererades istället data i förväg, då datagenerering kan parallelliseras mångfaldigt. Detta tillvägagångsätt har också fördelen att samma data kan återanvändas för att träna flera nätverk.

Den data som behöver genereras är par av felkedjor och ekvivalensklassfördelningar. Felkedjan, som entydigt definierar ett syndrom, matas in i nätverket som därefter föreslår en korrektionskedja inom en viss ekvivalensklass. Ekvivalensklassen jämförs med ekvivalensklassfördelningen i facit och en belöning r_t ges utifrån detta.

För att generera tillräckligt många datapunkter att träna på användes Chalmers datakluster VERA, där algoritmen körs parallellt på många processorkärnor samtidigt. När tillräckligt många datapunkter samlats in sparas och sammanställs all data i ett dataset. Ett minimum på 10^5 datapunker genereras vid $p = 0.1$ för att träna nätverket, både för $d = 3$ och $d = 5$. För $d = 3$ användes inget konvergenskriterium. De parametrar som användes var `steps` $= 10^5$, `Nc` $= 11$ samt `iters` $= 10$. Inget konvergenskriterium användes då ekvivalensklassfördelningarna konvergerade snabbt, och 10^5 steg är mer än ett tillräckligt antal för konvergens. Det bedömdes inte tidseffektivt att använda konvergenskriterium här. För $d = 5$ genererades data enligt de parametrar som bestämdes i avsnitt 3.1.3. Data kan på samma sätt genereras för att utvärdera prestandan för MCMC-dekodern. Eftersom att datan som sparas innehåller par av felkedjor och ekvivalensklassfördelningar, så kan prestandan för MCMC-dekodern bestämmas från hur ofta den mest sannolika ekvivalensklassen stämmer överens med ekvivalensklass för tillhörande felkedja. Detta görs genom att generera 10^5 datapunker från MCMC-dekodern för $p = 0.05, 0.06, ..., 0.19, 0.20$ för $d = 3$ och $d = 5$. För $d = 7$ genereras endast ett minimum på 2300 datapunker för varje p, då algoritmen tar längre tid för denna storlek.

3.2.2 Träning av neuralt nätverk

För den aktuella problemformuleringen representeras ett tillstånd som en $2 \times d \times d$ tensor, där de två $d \times d$ matriserna representerar det fullständiga sydromet. Denna tensor används som indata till nätverket som presenteras i [10]. När nätverket tränas lär det sig att approximera Q-värden för applikation av x-, y- respektive z-operatorerna för alla kvantbitar som gränsar till en defekt. Genom att jämföra vilket par av kvantbit/operation som har högst Q-värde kan aktuell operator appliceras på tillhörande kvantbit.



Figur 10: Flödesschema för den nya inlärningsmetoden, där förgenererad träningsdata nyttjas i belöningen för DRL-dekodern. Varje datapunkt består av en felkedja och dess tillhörande ekvivalensklassfördelning. Felkedjan görs om till ett syndrom och används som input till det neurala nätverket, som löser syndromet och sparar varje korrektionsteg i `correction_chain`. När syndromet är löst jämförs korrektionskedjan med ekvivalensklassfördelningen för motsvarande syndrom, och en proportionell belöning ges till nätverket enligt detta.

Ett flödesschema för den nya inlärningsmetoden syns i figur 10, där förgenererad data med felkedjor och motsvarande ekvivalensklassfördelningar har sparats i en databas och används vid träningen av nätverket med det nya belöningsprotokollet. Korrektionskedjor sparas genom en ny funktion `correction_chain` varje gång nätverket utför ett korrektionssteg. Denna korrektionskedjas ekvivalensklass jämförs sedan med ekvivalensklassfördelningen i databasen för att avgöra hur sannolik den är. Sedan får nätverket en belöning genom funktionen `get_reward` baserat på detta.

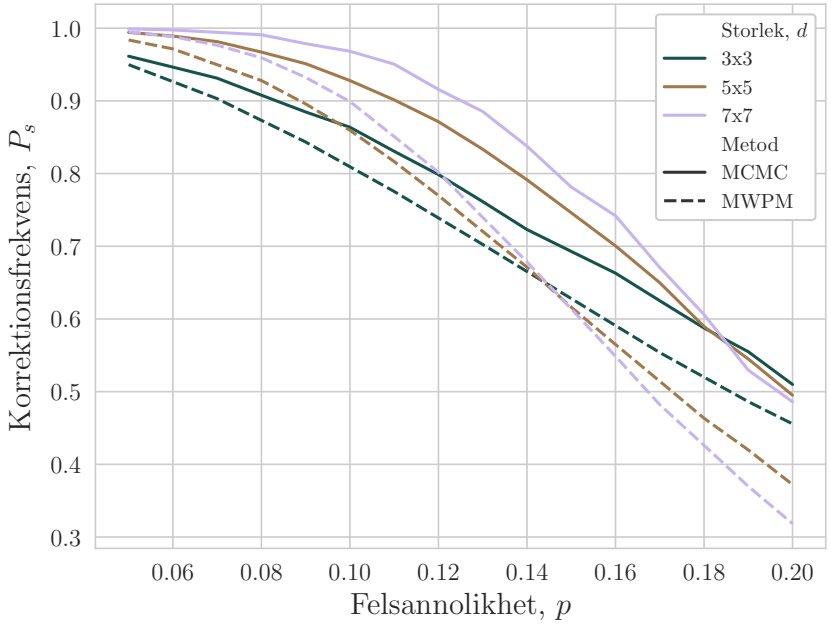
Belöningen som ges från `get_reward` är alltid 0 om syndromet inte är löst. Om syndromet löses ges en belöning mellan 0 och 100, som är sannolikheten i procent att korrektionskedjan löser syndromet utan att ett logiskt fel uppstår. Detta innebär att belöningen bara beror på syndromet och inte det underliggande felets verkliga utseende. Givet ett syndrom får nätverket alltid samma belöning för samma korrektionskedja, även om den någon gång råkar inducera ett logiskt fel.

För träningen på $d = 3$ och $d = 5$ användes två olika nätverksarkitekturer som beskrivs i detalj i bilaga B. För att minska träningstiden infördes en parameter `max_nbr_steps`, vilket är en begränsning på antalet handlingar som nätverket kan utföra för att lösa syndromet under ett träningssteg. För $d = 3$ användes `max_nbr_steps = 5` och för $d = 5$ användes `max_nbr_steps = 10`. Fullständiga träningsparametrar kan ses i bilaga A.

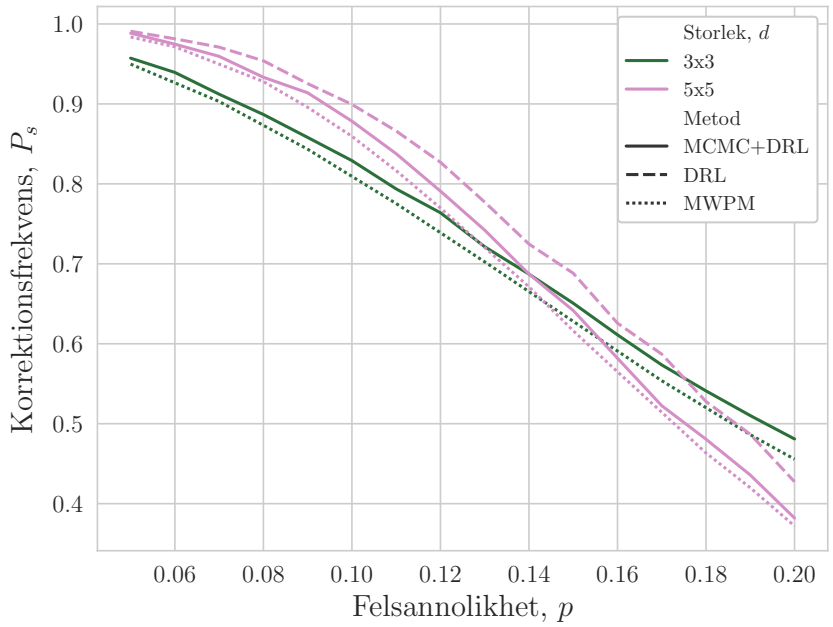
4 Resultat

I detta avsnitt presenteras hur MCMC-dekodern och den anpassade DRL-dekodern presterat jämfört med referensalgoritmer. De algoritmer som har använts som referens är MWPM, beskriven i avsnitt 2.2.1, och den tidigare algoritmen baserad på djup Q-inlärning enligt [10], vilken benämns DRL. Vår DRL-dekoder tränad med belöningsschemat från MCMC-dekodern benämns MCMC+DRL.

Prestandan visas i termer av korrektionsfrekvensen P_s, det vill säga hur stor andel syndrom som korrigeras utan att ge upphov till ett logiskt fel. I figur 11 jämförs P_s mellan MCMC-dekodern och MWPM. För de systemstorlekar som testats har MCMC-dekodern ett högre P_s för alla felsannolikheter. I figur 12 jämförs P_s för MCMC+DRL mot P_s för den tidigare DRL-dekodern. För $d = 3$ tog träningen 40 minuter över $6 \cdot 10^4$ iterationer. För $d = 5$ tränades nätverket i 14 timmar i $3 \cdot 10^5$ iterationer.



Figur 11: Korrektionsfrekvens P_s som funktion av felsannolikheten p för gitterstorlekarna $d = 3, 5, 7$. MCMC-dekodern jämförs mot MWPM som beskrivs i 2.2.1. Dessa benäms i figuren som MWPM respektive MCMC. MCMC-dekodern har bättre korrektionsfrekvens än MWPM för alla undersökta p och får en feltröskel vid $p = 0.185$, vilket är där korsningen sker mellan de kurvorna för $d = 3, 5, 7$.



Figur 12: Korrektionsfrekvens P_s som funktion av felsannolikheten p för gitterstorlekarna $d = 3, 5$. MCMC+DRL-dekodern jämförs mot MWPM som beskrivs i 2.2.1 samt DRL-dekodern som tränades med det tidigare belöningsschemat i [10]. Dessa benäms i figuren som MCMC+DRL, MWPM respektive RL. MCMC+DRL har något bättre korrektionsfrekvens än MWPM för alla undersökta p och får en feltröskel vid $p \approx 0.14$ vilket är där korsningen sker mellan de två kurvorna för $d = 3, 5$.

Figur 11 och 12 ger även information om tröskelvärdet för dekodrarna. Detta är en punkt där P_s och p är lika för alla gitterstorlekar. För MWPM ligger denna punkt vid $p = 0.145$ och för MCMC+DRL-dekodern sker detta vid $p = 0.14$. MCMC-dekodern har ett tröskelvärde på ungefär 18.5%. För en beskrivning av feltröskel se avsnitt 2.2.

5 Analys och diskussion

I detta avsnitt analyseras och diskuteras de resultat som erhållits och de metoder som använts under projektet. Inledningsvis analyseras den korrektionsfrekvens och feltröskel som erhållits utifrån figur 11 och 12. Därefter diskuteras metoderna, vidare forskning, potentiella tillämpningar och slutligen även etiska aspekter av projektet.

5.1 Korrektionsfrekvens och feltröskel

Då MCMC-dekodern genererar sannolikhetsfördelningen utifrån samma felkedja som den ämnar att korrigera är det viktigt att kontrollera att det inte finns ett bias med avseende på frö-kedjan och dess ekvivalensklass. Detta eftersom dekodern egentligen bara ska baseras på syndromet, och inte på någon specifik tillhörande kedja. Om en bias existerar leder det till en överskattning av korrektionsfrekvensen

P_s och tröskelvärdet. En enkel kontroll är att jämföra värdet för P_s då olika felkedjor med samma syndrom används som frö. Om P_s blir densamma oavsett frö är det ett tecken på att MCMC-dekodern inte påverkas av bias. En kontroll gjordes för $d = 5$, med $p = 0.1$, $p = 0.185$ och $p = 0.189$ och visas i tabell 1. För varje p genererades 10000 felkedjor enligt *depolarizing noise*-modellen. Från varje sådan ursprungskedja genererades sedan en deformerad kedja genom att likformigt applicera en av de 16 möjliga logiska operatorkombinationerna. Därefter applicerades, var och en med sannolikhet 50%, samtliga $2 \cdot 5^2 = 50$ möjliga stabilisatorer på den deformerade kedjan, varefter den motsvarade en uniformt samplad felkedja ur alla felkedjor konsekventa med ursprungskedjans syndrom. Både ursprungskedjan och den deformerade kedjan användes sedan som frö i MCMC, och P_s beräknades som andelen gånger den enligt MCMC-dekodern mest sannolika ekvivalensklassen var densamma som ursprungskedjans ekvivalensklass.

Tabell 1: Medelvärde av P_s för olika frökedjor. I det ena fallet används den ursprungskedja som orsakat syndromet. I det andra fallet används en helt slumpmässig felkedja med samma syndrom, som genereras från ursprungskedjan med en kombination av stabilisatorer och logiska operatorer. MCMC-algoritmen uppvisar för $p = 0.1$ en svag bias med avseende på ursprungskedjan.

	Ursprungskedja	Deformerad kedja
p	P_s	P_s
0.1	0.9279 ± 0.0051	0.9220 ± 0.0053
0.185	0.5731 ± 0.0097	0.5710 ± 0.0097
0.189	0.5539 ± 0.0097	0.5511 ± 0.0097

I tabell 1 visas de beräknade värdena för P_s samt ett 95% konfidensintervall för de olika frökedjorna. Av tabellen framgår det att MCMC-dekodern har en liten bias för frökedjans ekvivalensklass, då P_s är lite högre då ursprungskedjan används som frö än då den deformerade kedjan används. Tabellen antyder även ett beroende av bias på felsannolikheten p, då skillnaden i korrektionsfrekvens ΔP_s är cirka 0.006 för $p = 0.1$ och 0.002 för $p = 0.185$ och $p = 0.189$. Dessa skillnader är dock små i förhållande till korrektionsfrekvensen, så MCMC-dekodern kan fortfarande anses prestera väl, och dess bias borde inte avsevärt påverka träningen av DRL+MCMC-dekodern.

Vi kan observera från figur 11 att MCMC-dekoderns feltröskel på cirka 18.5% är nära den teoretiska gränsen på 18.9% i [15]. Resultatet bekräftar MCMC-dekoderns feltröskel i [14], men nu applicerad på en torisk kod. MCMC-dekodern har under tröskelvärdet en betydligt högre korrektionsfrekvens än både MWPM och den tidigare DRL-dekodern från [10]. Med anledning av den höga korrektionsfrekvensen och tröskelvärdet, samt det låga biaset med avseende på valet av frö, är det välmotiverat att använda MCMC-dekodern för generering av träningsdata till det neurala nätverket.

Träningen med MCMC-belöning redovisas i figur 12, där prestandan för MCMC+DRL jämförs med MWPM och den tidigare DRL-dekodern. Feltröskeln på 14% för MCMC+DRL är lägre än för den tidigare DRL-dekodern [10] och MWPM. Den har däremot högre korrektionsfrekvens på ett $d = 3$

gitter i jämförelse med MWPM. Även för $d = 5$ presterar den bättre än MWPM för alla p, men har lägre korrektionsfrekvens än den tidigare DRL-dekodern. En orsak till att feltröskeln är så låg kan vara att nätverket inte konvergerade tillräckligt väl för storleken $d = 5$, då träningen var tidsbegränsad. Det bör även noteras att tröskelvärdet har en stor osäkerhet eftersom det uppskattas från enbart två kurvor, och konvergensen till viss del är slumpmässig från träningsförsök till träningsförsök. Fler systemstorlekar behöver testas för att kunna dra en pålitlig slutsats om feltröskeln. Resultatet som uppnåddes är inte tillräckligt för att påvisa fördelen med vårt belöningsprotokoll i relation till den tidigare DRL-dekodern. Däremot har vi visat att metoden leder till högre korrektionsfrekvens än MWPM för de studerade systemstorlekarna, vilket kan betraktas som en konceptvalidering.

Feltröskeln är ett vanligt mått på prestanda för en dekoder, men i fallet med DRL-dekodrar kan den diskuteras. För MCMC+DRL har olika nätverksstrukturer använts för systemstorlekarna $d = 3$ och $d = 5$, och för var och en av dem beror konvergensen på en stor mängd hyperparametrar. Då algoritmen på sätt och vis ändras för olika d är feltröskeln ett relativt opålitligt mått på prestanda. Notera exempelvis i figur 12 att om prestandan för vår $d = 3$ DRL-dekoder försämras, så kommer korsningen med $d = 5$ DRL-dekodern att ske senare, och feltröskeln ökar. Vår dekoder har högre korrektionsfrekvens än MWPM, både för $d = 3$ och $d = 5$, men har trots det en sämre feltröskel. Felkorrektion i praktiken kommer inte att ske för felsannolikheter nära feltröskeln [11] eller med oändliga systemstorlekar. Istället är det den faktiska korrektionsfrekvensen för en given systemstorlek som är intressant.

5.2 MCMC-dekoder

MCMC-dekodern som diskuterades i detta arbete är som nämnt inte en fullständig dekoder eftersom felkedjan som skapar syndromet också agerar som frö. En verklig dekoder skulle skapa en frökedja genom att till exempel para ihop defekter som i avsnitt 2.2.1. Efter att ha hittat den mest sannolika ekvivalensklassen behöver en korrektionskedja inom den ekvivalensklassen appliceras på syndromet. Detta kan exempelvis göras genom att lägga på en unik kombination av logiska operatorer på frökedjan. En MCMC-dekoder skulle dock inte vara praktisk i verkligheten eftersom det tar mycket lång tid att generera fördelningen. Den är ineffektiv eftersom den måste sampla över alla möjliga felkedjor som motsvarar ett syndrom även om syndromet är mycket enkelt. Med ett neuralt nätverk är fördelen att även om det tar lång tid att tränas, så sker själva korrektionen på extremt kort tid, vilket är avgörande inom praktiska applikationer.

Hur konvergenskriterium ska väljas är inte uppenbart, och kräver i praktiken empiriska undersökningar såväl som teoretiska argument för att uppnå önskade resultat. Det är troligtvis möjligt att uppnå en representativ sannolikhetsfördelning på kortare tid med andra parametrar eller konvergenskriterier. Alternativt, om endast den mest sannolika ekvivalensklassen söks, kan konvergensen avgöras tidigare. Ett exempel på ett sådant konvergenskriterium presenteras i [14].

5.3 DRL-dekoder

En av anledningarna till att MCMC+DRL-dekodern inte presterar bättre än den tidigare DRL-algoritmen kan vara att belöningen är av en icke-markoviansk karaktär, alltså beskrivs inte processen av en MDP, se avsnitt 2.2.2. Nätverket får bara belöning i sista steget, när den löst hela syndromet. Denna belöning är dock inte oberoende av tidigare handlingar. Belöningen är beroende på alla tidigare steg som nätverket tagit för det aktuella syndromet, då det är dessa som definierar en ekvivalensklass. Det insågs under projektets gång att detta kan orsaka problem med konvergens, men valet att försöka gjordes ändå. För att förstå problematiken med en icke-markoviansk belöning kan vi betrakta följande enkla exempel. När nätverket har lyckats eliminera alla defekter i ett syndrom utom två, så är kvantbiten dem emellan den uppenbara lösningen för att slutföra felkorrektionen. Problemet är att belöningen för att eliminera denna defekt kommer att variera från gång till gång, beroende på hela sekvensen av handlingar som utfördes vid lösningen av syndromet. Detta är ingenting nätverket kan ta hänsyn till, då nätverket inte "minns" något av systemets tidigare tillstånd. Detta leder i sin tur till att nätverket inte kan lära sig att uppskatta Q-värdet för denna handling. Nätverket lär sig istället att minimera någon slags medelfel till den verkliga belöningen, som varierar från gång till gång. Detta gör det omöjligt för nätverket att identifiera och hålla sig inom den mest sannolika ekvivalensklassen som önskat, utan det måste i varje steg göra det som maximerar Q-värdet, och detta är som nämnt snarare ett försök till att minimera ett medelfel än att välja rätt ekvivalensklass. För att lösa detta skulle det vara möjligt att omdefiniera problemet till en MDP, som egentligen är en förutsättning för DQN. Den tidigare DRL-dekodern hade markoviansk belöning eftersom den konsistent fick samma belöning för ändring i antalet defekter. Detta skulle kunna förklara varför MCMC+DRL har lägre korrektionsfrekvens än den tidigare DRL-dekodern för $d = 5$ enligt figur 12.

Ett sätt att omformulera problemet till en MDP är att omdefiniera tillståndsrummet. Detta skulle kunna göras genom att till exempel alltid inkludera de tidigare steg nätverket tagit, eller det initiala syndromet, som en del av tillståndsrummet. Detta skulle resultera i att informationen om tidigare handlingar blir inbyggd i tillståndsrummet. Problemet med denna lösning är att man kvadrerar storleken på tillståndsrummet, vilket i sin tur leder till ett mycket mer komplicerat problem att lösa för nätverket. Ett sådant nätverk skulle därför möjligtvis behöva tränas längre.

I det nuvarande belöningsprotokollet belönas nätverket proportionerligt mot hur sannolik den föreslagna lösningens ekvivalensklass var. Valet av detta belöningsprotokoll gjordes för att den maximala belöningen ska fås genom att lösa syndromet optimalt, samtidigt som träningen underlättades av mindre belöningar för mindre sannolika ekvivalensklasser. Det finns dock flera varianter på belöningsprotokoll som hade kunnat användas. En variant skulle kunna vara att alltid ge en konstant belöning för att lösa syndromet med den optimala ekvivalensklassen, och 0 belöning annars. En fördel med detta är att belöningen alltid blir lika stor om nätverket gör rätt, vilket skulle kunna underlätta konvergensen. Möjligtvis skulle träningstiden behöva ökas eftersom antalet utdelade belöningar minskas. Ett annat alternativ skulle vara att normera belöningen, så att den mest sannolika ekvivalensklassen alltid gav 100 poäng, och de andra ekvivalensklasserna viktas proportionellt mot detta. Ett sådant protokoll är mer konsekvent, då likt det övre förslaget alla optimala lösningar kommer belönas lika mycket. Till skillnad från förslaget med belöning för mest sannolika klassen skulle detta förslag också ge belöning vid eliminering av syndromet vilket i så fall inte borde

leda till samma potentiella ökning av träningstiden. En annan fördel med ett normerat protokoll, och det protokoll vi använde, är att det kan ta hänsyn till degenerade ekvivalensklasser, alltså då det finns flera lika och mest sannolika ekvivalensklasser för ett syndrom. Ett exempel på degenererade ekvivalensklasser kan ses i figur 9, där ekvivalensklass 8 och 9 är ungefär lika sannolika.

Val av dekoder kan även påverkas av vilken sorts brus som önskas att hantera under felkorrektionen. I de flesta samtidiga dekodrar hanteras brusmodeller var för sig, det vill säga att en dekoder tränas att hantera en sorts brus där den presterar väl. En dekoder tränad till att hantera *depolarising noise* kan prestera suboptimalt för *biased noise*, och vice versa.

5.4 Vidare forskning och tillämpningar

Vidare forskning behövs för att kontrollera om det är motiverat att använda förstärkningsinlärning för detta problem. Ett alternativ skulle kunna vara att istället använda väglett lärande, *supervised learning*, med MCMC-dekodern som referens, genom att mata in syndrom till ett nätverk som sedan skulle föreslå den mest sannolika av 16 ekvivalensklasser. Denna information skulle sedan enkelt kunna tillämpas för att hitta en passande korrektionskedja. Djup förstärkningsinlärning har bland annat en fördel över väglett lärande om nätverket kan lära sig mer om omgivningen genom att interagera med den. Detta gäller dock ej för vårat nätverk eftersom all information om den mest sannolika ekvivalensklassen är lagrad i ursprungssyndromet och ingen information om den underliggande felkedjan kan återfås genom mätningar på fysiska kvantbitar.En till fördel med att använda väglett lärande hade varit att man inte fått problem med icke-markoviansk belöning. Utöver det hade man undkommit antagandet om att den kortaste korrektionskedjan är den mest sannolika.

En ytterligare möjlighet för vidare forskning är att anpassa den MCMC-dekodern som konstruerats till att kunna ta hänsyn till godtyckligt brus. Då bruset i fysiska kvantbitar är av olika karaktär och inte enkelt kan beskrivas med en typ av brusmodell, såsom *depolarizing noise*, skulle ett nätverk kunna tränas för att hantera olika kombinationer av brus. En väsentlig svårighet infinner sig i att typen av brusmodell har en verkan på hur snabbt konvergens uppnås, och därtill ställs det högre krav på konvergensanalysen. Taget till en praktisk nivå i framtida kvantdatorer skulle en tillämpning av detta kunna leda till individuellt konstruerade brusmodeller för kvantprocessorer som används för att korrigera de defekter som uppstår, en form av kalibrering för kvantdatorn.

Trots de fysiska begränsningar som idag finns för praktiskt realiserbara kvantdatorer är specialfallet av topologisk kod, den toriska koden, med olika brusmodeller ett högst intressant och viktig specialfall att studera. I praktiken innebär en tvådimensionell arkitektur hos kvantprocessorer att torisk kod med sina periodiska randvillkor kan vara illa lämpad till syftet, och någon form av plankod måste eventuellt tillämpas istället. Likaså återstår praktiska svårigheter med hopsättningen av fysiska kvantbitar till processorer, och vidare kvantdatorer, vilket inte är ett trivialt problem för ingenjörskonsten. Högst sannolikt kommer kvantdatorer användas i kombination med konventionella datorer, där kvantdatorernas överlägsna beräkningskraft används till att hantera de resursintensiva simuleringarna av bland annat de komplexa kemiska system nämnda i avsnitt 1. I väntan på de praktiska tillämpningarna återstår det för det teoretiska arbetet att fortlöpa för att utveckla den optimala dekodern för den bäst lämpade felkorrigeringskoden.

5.5 Etiska aspekter

I framtiden då kvantadatorer möjligen kommer finnas tillgängliga i betydligt större skala kan detta bidra till både positiva och negativa effekter. Att kunna utföra simuleringar av kemiska system kan som nämnt exempelvis leda till effektivare kvävefixering i marken eller nya läkemdel. Sådana effekter är klart positiva för både samhället i stort och för den enskilde individen. Effektivare kvävefixering kan ge samhället möjlighet att lägga mindre fysiska och ekonomiska resurser på matproduktion. För den enskilde individen som blir sjuk kan nya läkemdel ge denne möjligheten att bli frisk. Kvantdatorer ger även möjlighet att med hjälp av Shor's algoritm för primtalsfaktorisering bryta vissa typer av kryptering som just bygger på att det är svårt att faktorisera stora primtal. Om vissa typer av kryptering knäcks kan detta leda till att säkerheten för vissa system bryts eller att hemlig information sprids, vilket i sin tur hade kunnat påverka samhället negativt.

Även om genomförandet av detta projekt inte har några etiska dilemman eller direkt påverkan på samhället så finns möjligheten att resultatet kan bidra. Som nämnts i bakgrunden 1 av rapporten så är det av yttersta vikt att utveckla en så generell och snabb algoritm för kvantfelskorrektion som möjligt. Utan att kvantfelskorrektion kan man inte realisera kvantdatorer i praktiken. Vad vårt arbete kan bidra med till forskningen kring kvantfelskorrektion är insikter gällande vilka angreppssätt som kan komma att vara de mest framgångsrika.

Då det även existerar andra typer av krypteringsmetoder vilka inte kan knäckas genom primtalsfaktorisering anser vi att kvantdatorer troligtvis kommer skapa fler positiva användningsområden än negativa. Vi anser därför att det inte finns någon direkt anledning att tro att resultatet av detta projekt har möjlighet att bidra negativt till samhället.

6 Slutsatser

Vi har implementerat en MCMC-dekoder enligt [14], applicerad på den toriska koden. Våra resultat bekräftar den höga och nästintill optimala feltröskeln vid approximativt 18.5%. Dessutom har felkorrektionsfrekvensen för MCMC-dekodern konstaterats vara betydligt högre än både för MWPM och DRL. Genom att träna nätverket som presenteras i [10] med ett belöningprotokoll baserat på MCMC-algoritmen, uppnås bättre korrektionsfrekvens jämfört med MWPM för systemstorlekar $d = 3, 5$ och en feltröskel vid 14%. För $d = 5$ ses dock en markant försämring jämfört med den tidigare DRL-dekodern. Framgången gentemot MWPM visar dock att det är möjligt att framgångsrikt träna ett nätverk med vårt belöningsschema, men svårigheter med konvergens förhindrar nätverket att uppnå önskad prestanda. Vidare undersökning av nätverksträningen, och variationer på belöningsschemat bör i framtiden undersökas för att kunna avgöra metodens framgång.

Referenser

[1] N. Moll *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, 030503, Jun. 2018. DOI: 10.1088/2058-9565/aab822.

[2] A. Ambainis, "Quantum Search Algorithms," *SIGACT News*, vol. 35, nr. 2, ss. 22–35, Jun. 2004. DOI: 10.1145/992287.992296.

[3] K. Sugisaki *et al.*, "Quantum chemistry on quantum computers: quantum simulations of the time evolution of wave functions under the S2 operator and determination of the spin quantum number S," *Phys. Chem. Chem. Phys.*, vol. 21, ss. 15 356–15 361, Jul. 2019. DOI: 10.1039/C9CP02546D.

[4] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, och M. Troyer, "Elucidating reaction mechanisms on quantum computers," *Proceedings of the National Academy of Sciences*, 2017. DOI: 10.1073/pnas.1619152114. eprint: https://www.pnas.org/content/early/2017/06/30/1619152114.full.pdf.

[5] Y. Cao, J. Romero, och A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, vol. 62, nr. 6, 6:1–6:20, Dec. 2018. DOI: 10.1147/JRD.2018.2888987.

[6] E. Pednault, J. Gunnels, D. Maslov, och J. Gambetta, "On 'Quantum Supremacy'," *IBM Research Blog*, Okt. 2019. [Online]. Tillgänglig: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/, hämtad: 2020-05-13.

[7] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, nr. 7779, ss. 505–510, Okt. 2019. DOI: 10.1038/s41586-019-1666-5.

[8] M. Schlosshauer, "Quantum decoherence," *Physics Reports*, vol. 831, ss. 38–39, Okt. 2019. DOI: 10.1016/j.physrep.2019.10.001.

[9] D. Lu-ming och G. Guang-can, "Scheme for Reducing Decoherence in Quantum Computer Memory by Transformation to the Coherence-Preserving States," *Chinese Physics Letters*, vol. 14, nr. 7, ss. 488–491, Jul. 1997. DOI: 10.1088/0256-307X/14/7/003.

[10] D. Fitzek, M. Eliasson, A. F. Kockum, och M. Granath, "Deep Q-learning decoder for depolarizing noise on the toric code," 2019. arXiv: 1912.12919 [quant-ph].

[11] P. Andreasson, J. Johansson, S. Liljestrand, och M. Granath, "Quantum error correction for the toric code using deep reinforcement learning," *Quantum*, vol. 3, s. 183, Sep. 2019. DOI: 10.22331/q-2019-09-02-183.

[12] A. Hutter, J. R. Wootton, och D. Loss, "Efficient Markov chain Monte Carlo algorithm for the surface code," *Phys. Rev. A*, vol. 89, 022326, Feb. 2014. DOI: 10.1103/PhysRevA.89.022326.

[13] A. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, nr. 1, ss. 2–30, Jan. 2003. DOI: 10.1016/S0003-4916(02)00018-0.

[14] J. R. Wootton och D. Loss, "High Threshold Error Correction for the Surface Code," *Physical Review Letters*, vol. 109, 160503, Okt. 2012. DOI: 10.1103/physrevlett.109.160503.

[15] H. Bombin, R. S. Andrist, M. Ohzeki, H. G. Katzgraber, och M. A. Martin-Delgado, "Strong Resilience of Topological Codes to Depolarization," *Phys. Rev. X*, vol. 2, 021004, Aug. 2012. DOI: 10.1103/PhysRevX.2.021004.

[16] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and more*. Birmingham, UK: Packt Publishing, 2018.

Bilaga A Träningsparametrar

Parametrar till träningen av MCMC+DRL-dekodern valdes utefter erfarenhet om vad som gav bäst konvergens. Parametrarna presenteras i tabell 2. För de parametrar som inte nämns här användes samma värden som i [10].

Tabell 2: Träningsparametrar för MCMC+DRL för systemstorlekar $d = 3, 5$.

gitterstorlek, d	nätverk	# träningssteg	learning-rate	max_nbr_steps
3	NN_6	60000	0.00025	5
5	NN_11	300000	0.000025	10

Bilaga B Nätverksarkitekturer

Olika nätverksstrukturerer användes för olika systemstorlekar. De två nätverk som användes för att träna på $d = 3$ och $d = 5$ syns i tabell 3 respektive tabell 4.

Tabell 3: Nätverksarkitektur för $d = 3$, NN_6.

lager	typ	storlek	steglängd	kernel	vaddering
1	2d Konv.	128	1	3	periodisk
2	2d Konv.	128	1	3	1
3	2d Konv.	120	1	3	1
4	2d Konv.	111	1	3	1
5	2d Konv.	91	1	3	1
6	2d Konv.	64	1	3	1
7	Lin.	3	-	-	-

Tabell 4: Nätverksarkitektur för $d = 5$, NN_11.

lager	typ	storlek	steglängd	kernel	vaddering
1	2d Konv.	128	1	3	periodisk
2	2d Konv.	128	1	3	1
3	2d Konv.	120	1	3	1
4	2d Konv.	111	1	3	1
5	2d Konv.	104	1	3	1
6	2d Konv.	103	1	3	1
7	2d Konv.	90	1	3	1
8	2d Konv.	80	1	3	1
9	2d Konv.	73	1	3	1
10	2d Konv.	71	1	3	1
11	2d Konv.	64	1	3	1
12	Lin.	3	-	-	-